



MACHINE LEARNING

A BAYESIAN AND OPTIMIZATION PERSPECTIVE

SERGIOS THEODORIDIS



Machine Learning

A Bayesian and Optimization Perspective

This page intentionally left blank

Machine Learning

A Bayesian and Optimization Perspective

Sergios Theodoridis



AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO
Academic Press is an imprint of Elsevier



Academic Press is an imprint of Elsevier
125 London Wall, London, EC2Y 5AS, UK
525 B Street, Suite 1800, San Diego, CA 92101-4495, USA
225 Wyman Street, Waltham, MA 02451, USA
The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, UK

Copyright © 2015 Elsevier Ltd. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or any information storage and retrieval system, without permission in writing from the publisher. Details on how to seek permission, further information about the Publisher's permissions policies and our arrangements with organizations such as the Copyright Clearance Center and the Copyright Licensing Agency, can be found at our website: www.elsevier.com/permissions.

This book and the individual contributions contained in it are protected under copyright by the Publisher (other than as may be noted herein).

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data

A catalog record for this book is available from the Library of Congress

ISBN: 978-0-12-801522-3

For information on all Academic Press publications
visit our website at <http://store.elsevier.com/>

Publisher: Jonathan Simpson

Acquisition Editor: Tim Pitts

Editorial Project Manager: Charlie Kent

Production Project Manager: Susan Li

Designer: Greg Harris

Typeset by SPi Global, India

Printed and bound in The United States

15 16 17 18 19 10 9 8 7 6 5 4 3 2 1



Working together
to grow libraries in
developing countries

www.elsevier.com • www.bookaid.org

Contents

Preface	xvii
Acknowledgments	xix
Notation	xxi
CHAPTER 1 Introduction	1
1.1 What Machine Learning is About.....	1
1.1.1 Classification.....	2
1.1.2 Regression.....	3
1.2 Structure and a Road Map of the Book	5
References	8
CHAPTER 2 Probability and Stochastic Processes	9
2.1 Introduction	10
2.2 Probability and Random Variables	10
2.2.1 Probability.....	11
2.2.2 Discrete Random Variables.....	12
2.2.3 Continuous Random Variables	14
2.2.4 Mean and Variance	15
2.2.5 Transformation of Random Variables.....	17
2.3 Examples of Distributions	18
2.3.1 Discrete Variables.....	18
2.3.2 Continuous Variables	20
2.4 Stochastic Processes	29
2.4.1 First and Second Order Statistics	30
2.4.2 Stationarity and Ergodicity	30
2.4.3 Power Spectral Density	33
2.4.4 Autoregressive Models	38
2.5 Information Theory	41
2.5.1 Discrete Random Variables.....	42
2.5.2 Continuous Random Variables	45
2.6 Stochastic Convergence	48
Problems.....	49
References	51
CHAPTER 3 Learning in Parametric Modeling: Basic Concepts and Directions 53	
3.1 Introduction	53
3.2 Parameter Estimation: The Deterministic Point of View	54

3.3	Linear Regression.....	57
3.4	Classification.....	60
3.5	Biased Versus Unbiased Estimation	64
3.5.1	Biased or Unbiased Estimation?	65
3.6	The Cramér-Rao Lower Bound	67
3.7	Sufficient Statistic.....	70
3.8	Regularization.....	72
3.9	The Bias-Variance Dilemma	77
3.9.1	Mean-Square Error Estimation.....	77
3.9.2	Bias-Variance Tradeoff	78
3.10	Maximum Likelihood Method	82
3.10.1	Linear Regression: The Nonwhite Gaussian Noise Case	84
3.11	Bayesian Inference.....	84
3.11.1	The Maximum a Posteriori Probability Estimation Method	88
3.12	Curse of Dimensionality.....	89
3.13	Validation	91
3.14	Expected and Empirical Loss Functions	93
3.15	Nonparametric Modeling and Estimation	95
Problems	97	
References.....	102	
CHAPTER 4	Mean-Square Error Linear Estimation	105
4.1	Introduction	105
4.2	Mean-Square Error Linear Estimation: The Normal Equations	106
4.2.1	The Cost Function Surface	107
4.3	A Geometric Viewpoint: Orthogonality Condition.....	109
4.4	Extension to Complex-Valued Variables	111
4.4.1	Widely Linear Complex-Valued Estimation.....	113
4.4.2	Optimizing with Respect to Complex-Valued Variables: Wirtinger Calculus	116
4.5	Linear Filtering	118
4.6	MSE Linear Filtering: A Frequency Domain Point of View.....	120
4.7	Some Typical Applications	124
4.7.1	Interference Cancellation	124
4.7.2	System Identification	125
4.7.3	Deconvolution: Channel Equalization	126
4.8	Algorithmic Aspects: The Levinson and the Lattice-Ladder Algorithms	132
4.8.1	The Lattice-Ladder Scheme	137
4.9	Mean-Square Error Estimation of Linear Models	140
4.9.1	The Gauss-Markov Theorem.....	143
4.9.2	Constrained Linear Estimation: The Beamforming Case	145

4.10	Time-Varying Statistics: Kalman Filtering	148
	Problems	154
	References.....	158
CHAPTER 5 Stochastic Gradient Descent: The LMS Algorithm and its Family.....		161
5.1	Introduction	162
5.2	The Steepest Descent Method.....	163
5.3	Application to the Mean-Square Error Cost Function	167
	5.3.1 The Complex-Valued Case	175
5.4	Stochastic Approximation	177
5.5	The Least-Mean-Squares Adaptive Algorithm	179
	5.5.1 Convergence and Steady-State Performance of the LMS in Stationary Environments	181
	5.5.2 Cumulative Loss Bounds	186
5.6	The Affine Projection Algorithm.....	188
	5.6.1 The Normalized LMS	193
5.7	The Complex-Valued Case	194
5.8	Relatives of the LMS	196
5.9	Simulation Examples	199
5.10	Adaptive Decision Feedback Equalization	202
5.11	The Linearly Constrained LMS	204
5.12	Tracking Performance of the LMS in Nonstationary Environments	206
5.13	Distributed Learning: The Distributed LMS	208
	5.13.1 Cooperation Strategies.....	209
	5.13.2 The Diffusion LMS	211
	5.13.3 Convergence and Steady-State Performance: Some Highlights	218
	5.13.4 Consensus-Based Distributed Schemes.....	220
5.14	A Case Study: Target Localization	222
5.15	Some Concluding Remarks: Consensus Matrix	223
	Problems	224
	References.....	227
CHAPTER 6 The Least-Squares Family.....		233
6.1	Introduction	234
6.2	Least-Squares Linear Regression: A Geometric Perspective	234
6.3	Statistical Properties of the LS Estimator.....	236
6.4	Orthogonalizing the Column Space of X : The SVD Method.....	239
6.5	Ridge Regression	243
6.6	The Recursive Least-Squares Algorithm	245

6.7	Newton's Iterative Minimization Method	248
6.7.1	RLS and Newton's Method.....	251
6.8	Steady-State Performance of the RLS	252
6.9	Complex-Valued Data: The Widely Linear RLS	254
6.10	Computational Aspects of the LS Solution	255
6.11	The Coordinate and Cyclic Coordinate Descent Methods	258
6.12	Simulation Examples	259
6.13	Total-Least-Squares.....	261
	Problems	268
	References.....	272
CHAPTER 7	Classification: A Tour of the Classics	275
7.1	Introduction	275
7.2	Bayesian Classification	276
7.2.1	Average Risk	278
7.3	Decision (Hyper)Surfaces	280
7.3.1	The Gaussian Distribution Case.....	282
7.4	The Naive Bayes Classifier.....	287
7.5	The Nearest Neighbor Rule	288
7.6	Logistic Regression.....	290
7.7	Fisher's Linear Discriminant	294
7.8	Classification Trees	300
7.9	Combining Classifiers	304
7.10	The Boosting Approach	307
7.11	Boosting Trees	313
7.12	A Case Study: Protein Folding Prediction.....	314
	Problems	318
	References.....	323
CHAPTER 8	Parameter Learning: A Convex Analytic Path.....	327
8.1	Introduction	328
8.2	Convex Sets and Functions	329
8.2.1	Convex Sets	329
8.2.2	Convex Functions	330
8.3	Projections onto Convex Sets	333
8.3.1	Properties of Projections	337
8.4	Fundamental Theorem of Projections onto Convex Sets	341
8.5	A Parallel Version of POCS	344
8.6	From Convex Sets to Parameter Estimation and Machine Learning	345
8.6.1	Regression.....	345
8.6.2	Classification.....	347

8.7	Infinite Many Closed Convex Sets: The Online Learning Case	349
8.7.1	Convergence of APSM	351
8.8	Constrained Learning	356
8.9	The Distributed APSM	357
8.10	Optimizing Nonsmooth Convex Cost Functions	358
8.10.1	Subgradients and Subdifferentials	359
8.10.2	Minimizing Nonsmooth Continuous Convex Loss Functions: The Batch Learning Case	362
8.10.3	Online Learning for Convex Optimization	367
8.11	Regret Analysis	370
8.12	Online Learning and Big Data Applications: A Discussion	374
8.13	Proximal Operators	379
8.13.1	Properties of the Proximal Operator	382
8.13.2	Proximal Minimization	383
8.14	Proximal Splitting Methods for Optimization	385
Problems	389	
8.15	Appendix to Chapter 8	393
References	398	
CHAPTER 9 Sparsity-Aware Learning: Concepts and Theoretical Foundations		403
9.1	Introduction	403
9.2	Searching for a Norm	404
9.3	The Least Absolute Shrinkage and Selection Operator (LASSO)	407
9.4	Sparse Signal Representation	411
9.5	In Search of the Sparsest Solution	415
9.6	Uniqueness of the ℓ_0 Minimizer	422
9.6.1	Mutual Coherence	424
9.7	Equivalence of ℓ_0 and ℓ_1 Minimizers: Sufficiency Conditions	426
9.7.1	Condition Implied by the Mutual Coherence Number	426
9.7.2	The Restricted Isometry Property (RIP)	427
9.8	Robust Sparse Signal Recovery from Noisy Measurements	429
9.9	Compressed Sensing: The Glory of Randomness	430
9.9.1	Dimensionality Reduction and Stable Embeddings	433
9.9.2	Sub-Nyquist Sampling: Analog-to-Information Conversion	434
9.10	A Case Study: Image De-Noising	438
Problems	440	
References	444	
CHAPTER 10 Sparsity-Aware Learning: Algorithms and Applications		449
10.1	Introduction	450
10.2	Sparsity-Promoting Algorithms	450

10.2.1	Greedy Algorithms	451
10.2.2	Iterative Shrinkage/Thresholding (IST) Algorithms	456
10.2.3	Which Algorithm?: Some Practical Hints	462
10.3	Variations on the Sparsity-Aware Theme	467
10.4	Online Sparsity-Promoting Algorithms.....	475
10.4.1	LASSO: Asymptotic Performance	475
10.4.2	The Adaptive Norm-Weighted LASSO.....	477
10.4.3	Adaptive CoSaMP (AdCoSaMP) Algorithm	479
10.4.4	Sparse Adaptive Projection Subgradient Method (SpAPSM)	480
10.5	Learning Sparse Analysis Models	485
10.5.1	Compressed Sensing for Sparse Signal Representation in Coherent Dictionaries.....	487
10.5.2	Cosparsity	488
10.6	A Case Study: Time-Frequency Analysis	490
10.7	Appendix to Chapter 10: Some Hints from the Theory of Frames	497
	Problems	500
	References.....	502
CHAPTER 11	Learning in Reproducing Kernel Hilbert Spaces.....	509
11.1	Introduction	510
11.2	Generalized Linear Models.....	510
11.3	Volterra, Wiener, and Hammerstein Models.....	511
11.4	Cover's Theorem: Capacity of a Space in Linear Dichotomies	514
11.5	Reproducing Kernel Hilbert Spaces	517
11.5.1	Some Properties and Theoretical Highlights	519
11.5.2	Examples of Kernel Functions	520
11.6	Representer Theorem	525
11.6.1	Semiparametric Representer Theorem.....	527
11.6.2	Nonparametric Modeling: A Discussion	528
11.7	Kernel Ridge Regression	528
11.8	Support Vector Regression	530
11.8.1	The Linear ϵ -Insensitive Optimal Regression	531
11.9	Kernel Ridge Regression Revisited	537
11.10	Optimal Margin Classification: Support Vector Machines.....	538
11.10.1	Linearly Separable Classes: Maximum Margin Classifiers	540
11.10.2	Nonseparable Classes.....	545
11.10.3	Performance of SVMs and Applications	550
11.10.4	Choice of Hyperparameters	550
11.11	Computational Considerations	551
11.11.1	Multiclass Generalizations	552

11.12	Online Learning in RKHS	553
11.12.1	The Kernel LMS (KLMS)	553
11.12.2	The Naive Online R_{reg} Minimization Algorithm (NORMA)	556
11.12.3	The Kernel APSM Algorithm	560
11.13	Multiple Kernel Learning	567
11.14	Nonparametric Sparsity-Aware Learning: Additive Models.....	568
11.15	A Case Study: Authorship Identification	570
	Problems	574
	References.....	578
CHAPTER 12 Bayesian Learning: Inference and the EM Algorithm.....		585
12.1	Introduction	586
12.2	Regression: A Bayesian Perspective	586
12.2.1	The Maximum Likelihood Estimator	587
12.2.2	The MAP Estimator	588
12.2.3	The Bayesian Approach	589
12.3	The Evidence Function and Occam's Razor Rule	593
12.4	Exponential Family of Probability Distributions	600
12.4.1	The Exponential Family and the Maximum Entropy Method	605
12.5	Latent Variables and the EM Algorithm	606
12.5.1	The Expectation-Maximization Algorithm	606
12.5.2	The EM Algorithm: A Lower Bound Maximization View	608
12.6	Linear Regression and the EM Algorithm	610
12.7	Gaussian Mixture Models	613
12.7.1	Gaussian Mixture Modeling and Clustering	617
12.8	Combining Learning Models: A Probabilistic Point of View.....	621
12.8.1	Mixing Linear Regression Models	622
12.8.2	Mixing Logistic Regression Models	625
	Problems	628
12.9	Appendix to Chapter 12	631
12.9.1	PDFs with Exponent of Quadratic Form	631
12.9.2	The Conditional from the Joint Gaussian Pdf	632
12.9.3	The Marginal from the Joint Gaussian Pdf	633
12.9.4	The Posterior from Gaussian Prior and Conditional Pdfs.....	634
	References.....	637
CHAPTER 13 Bayesian Learning: Approximate Inference and Nonparametric Models		639
13.1	Introduction	640
13.2	Variational Approximation in Bayesian Learning	640
13.2.1	The Case of the Exponential Family of Probability Distributions	644

13.3	A Variational Bayesian Approach to Linear Regression	645
13.4	A Variational Bayesian Approach to Gaussian Mixture Modeling	651
13.5	When Bayesian Inference Meets Sparsity	655
13.6	Sparse Bayesian Learning (SBL)	657
13.6.1	The Spike and Slab Method	660
13.7	The Relevance Vector Machine Framework	661
13.7.1	Adopting the Logistic Regression Model for Classification	662
13.8	Convex Duality and Variational Bounds	666
13.9	Sparsity-Aware Regression: A Variational Bound Bayesian Path	671
13.10	Sparsity-Aware Learning: Some Concluding Remarks	675
13.11	Expectation Propagation	679
13.12	Nonparametric Bayesian Modeling	683
13.12.1	The Chinese Restaurant Process	684
13.12.2	Inference	684
13.12.3	Dirichlet Processes	684
13.12.4	The Stick-Breaking Construction of a DP	685
13.13	Gaussian Processes	687
13.13.1	Covariance Functions and Kernels	688
13.13.2	Regression	690
13.13.3	Classification	692
13.14	A Case Study: Hyperspectral Image Unmixing	693
13.14.1	Hierarchical Bayesian Modeling	695
13.14.2	Experimental Results	696
Problems	699	
References	702	
CHAPTER 14	Monte Carlo Methods	707
14.1	Introduction	707
14.2	Monte Carlo Methods: The Main Concept	708
14.2.1	Random number generation	709
14.3	Random Sampling Based on Function Transformation	711
14.4	Rejection Sampling	715
14.5	Importance Sampling	718
14.6	Monte Carlo Methods and the EM Algorithm	720
14.7	Markov Chain Monte Carlo Methods	721
14.7.1	Ergodic Markov Chains	723
14.8	The Metropolis Method	728
14.8.1	Convergence Issues	731
14.9	Gibbs Sampling	733
14.10	In Search of More Efficient Methods: A Discussion	735

14.11	A Case Study: Change-Point Detection	737
Problems	740	
References.....	742	
CHAPTER 15 Probabilistic Graphical Models: Part I	745	
15.1 Introduction	745	
15.2 The Need for Graphical Models	746	
15.3 Bayesian Networks and the Markov Condition	748	
15.3.1 Graphs: Basic Definitions	749	
15.3.2 Some Hints on Causality	753	
15.3.3 D -Separation	755	
15.3.4 Sigmoidal Bayesian Networks	758	
15.3.5 Linear Gaussian Models	759	
15.3.6 Multiple-Cause Networks	760	
15.3.7 I-Maps, Soundness, Faithfulness, and Completeness	761	
15.4 Undirected Graphical Models	762	
15.4.1 Independencies and I-Maps in Markov Random Fields	763	
15.4.2 The Ising Model and Its Variants	765	
15.4.3 Conditional Random Fields (CRFs)	767	
15.5 Factor Graphs	768	
15.5.1 Graphical Models for Error-Correcting Codes	770	
15.6 Moralization of Directed Graphs.....	772	
15.7 Exact Inference Methods: Message-Passing Algorithms	773	
15.7.1 Exact Inference in Chains	773	
15.7.2 Exact Inference in Trees	777	
15.7.3 The Sum-Product Algorithm	778	
15.7.4 The Max-Product and Max-Sum Algorithms	782	
Problems	789	
References.....	791	
CHAPTER 16 Probabilistic Graphical Models: Part II	795	
16.1 Introduction	795	
16.2 Triangulated Graphs and Junction Trees.....	796	
16.2.1 Constructing a Join Tree.....	799	
16.2.2 Message-Passing in Junction Trees	801	
16.3 Approximate Inference Methods.....	804	
16.3.1 Variational Methods: Local Approximation	804	
16.3.2 Block Methods for Variational Approximation	809	
16.3.3 Loopy Belief Propagation	813	
16.4 Dynamic Graphical Models	816	

16.5	Hidden Markov Models	818
16.5.1	Inference	821
16.5.2	Learning the Parameters in an HMM	825
16.5.3	Discriminative Learning	828
16.6	Beyond HMMs: A Discussion	829
16.6.1	Factorial Hidden Markov Models	829
16.6.2	Time-Varying Dynamic Bayesian Networks	832
16.7	Learning Graphical Models	833
16.7.1	Parameter Estimation	833
16.7.2	Learning the Structure	837
Problems	838	
References.....	840	
CHAPTER 17	Particle Filtering.....	845
17.1	Introduction	845
17.2	Sequential Importance Sampling.....	845
17.2.1	Importance Sampling Revisited	846
17.2.2	Resampling.....	847
17.2.3	Sequential Sampling	849
17.3	Kalman and Particle Filtering	851
17.3.1	Kalman Filtering: A Bayesian Point of View.....	852
17.4	Particle Filtering	854
17.4.1	Degeneracy	858
17.4.2	Generic Particle Filtering	860
17.4.3	Auxiliary Particle Filtering	862
Problems	868	
References.....	872	
CHAPTER 18	Neural Networks and Deep Learning.....	875
18.1	Introduction	876
18.2	The Perceptron	877
18.2.1	The Kernel Perceptron Algorithm	881
18.3	Feed-Forward Multilayer Neural Networks	882
18.4	The Backpropagation Algorithm.....	886
18.4.1	The Gradient Descent Scheme	887
18.4.2	Beyond the Gradient Descent Rationale	895
18.4.3	Selecting a Cost Function	896
18.5	Pruning the Network.....	897
18.6	Universal Approximation Property of Feed-Forward Neural Networks	899
18.7	Neural Networks: A Bayesian Flavor	902

18.8	Learning Deep Networks	903
18.8.1	The Need for Deep Architectures	904
18.8.2	Training Deep Networks	905
18.8.3	Training Restricted Boltzmann Machines	908
18.8.4	Training Deep Feed-Forward Networks	914
18.9	Deep Belief Networks	916
18.10	Variations on the Deep Learning Theme	918
18.10.1	Gaussian Units	918
18.10.2	Stacked Autoencoders	919
18.10.3	The Conditional RBM	920
18.11	Case Study: A Deep Network for Optical Character Recognition	923
18.12	Case Study: A Deep Autoencoder	925
18.13	Example: Generating Data via a DBN Problems	928
	References.....	929
		932
CHAPTER 19	Dimensionality Reduction	937
19.1	Introduction	938
19.2	Intrinsic Dimensionality	939
19.3	Principle Component Analysis	939
19.4	Canonical Correlation Analysis	950
19.4.1	Relatives of CCA	953
19.5	Independent Component Analysis	955
19.5.1	ICA and Gaussianity	956
19.5.2	ICA and Higher Order Cumulants	957
19.5.3	Non-Gaussianity and Independent Components	958
19.5.4	ICA Based on Mutual Information.....	959
19.5.5	Alternative Paths to ICA.....	962
19.6	Dictionary Learning: The k -SVD Algorithm	966
19.7	Nonnegative Matrix Factorization	971
19.8	Learning Low-Dimensional Models: A Probabilistic Perspective	972
19.8.1	Factor Analysis	972
19.8.2	Probabilistic PCA	974
19.8.3	Mixture of Factors Analyzers: A Bayesian View to Compressed Sensing	977
19.9	Nonlinear Dimensionality Reduction	980
19.9.1	Kernel PCA	980
19.9.2	Graph-Based Methods	982

19.10	Low-Rank Matrix Factorization: A Sparse Modeling Path	991
19.10.1	Matrix Completion.....	991
19.10.2	Robust PCA	995
19.10.3	Applications of Matrix Completion and ROBUST PCA	996
19.11	A Case Study: fMRI Data Analysis.....	998
	Problems	1002
	References.....	1003
APPENDIX A	Linear Algebra	1013
A.1	Properties of Matrices	1013
A.2	Positive Definite and Symmetric Matrices	1015
A.3	Wirtinger Calculus	1016
	References	1017
APPENDIX B	Probability Theory and Statistics	1019
B.1	Cramér-Rao Bound	1019
B.2	Characteristic Functions	1020
B.3	Moments and Cumulants	1020
B.4	Edgeworth Expansion of a Pdf	1021
	Reference	1022
APPENDIX C	Hints on Constrained Optimization	1023
C.1	Equality Constraints	1023
C.2	Inequality Constraints	1025
	References	1029
	Index	1031

Preface

Machine Learning is a name that is gaining popularity as an umbrella for methods that have been studied and developed for many decades in different scientific communities and under different names, such as Statistical Learning, Statistical Signal Processing, Pattern Recognition, Adaptive Signal Processing, Image Processing and Analysis, System Identification and Control, Data Mining and Information Retrieval, Computer Vision, and Computational Learning. The name “Machine Learning” indicates what all these disciplines have in common, that is, to *learn from data*, and then *make predictions*. What one tries to learn from data is their underlying structure and regularities, via the development of a *model*, which can then be used to provide predictions.

To this end, a number of diverse approaches have been developed, ranging from optimization of cost functions, whose goal is to optimize the deviation between what one observes from data and what the model predicts, to probabilistic models that attempt to model the statistical properties of the observed data.

The goal of this book is to approach the machine learning discipline in a unifying context, by presenting the major paths and approaches that have been followed over the years, without giving preference to a specific one. It is the author’s belief that all of them are valuable to the newcomer who wants to learn the secrets of this topic, from the applications as well as from the pedagogic point of view. As the title of the book indicates, the emphasis is on the processing and analysis front of machine learning and not on topics concerning the theory of learning itself and related performance bounds. In other words, the focus is on methods and algorithms closer to the application level.

The book is the outgrowth of more than three decades of the author’s experience on research and teaching various related courses. The book is written in such a way that individual (or pairs of) chapters are as self-contained as possible. So, one can select and combine chapters according to the focus he/she wants to give to the course he/she teaches, or to the topics he/she wants to grasp in a first reading. Some guidelines on how one can use the book for different courses are provided in the introductory chapter.

Each chapter grows by starting from the basics and evolving to embrace the more recent advances. Some of the topics had to be split into two chapters, such as sparsity-aware learning, Bayesian learning, probabilistic graphical models, and Monte Carlo methods. The book addresses the needs of advanced graduate, postgraduate, and research students as well as of practicing scientists and engineers whose interests lie beyond black-box solutions. Also, the book can serve the needs of short courses on specific topics, e.g., sparse modeling, Bayesian learning, probabilistic graphical models, neural networks and deep learning.

Most of the chapters include Matlab exercises, and the related code is available from the book’s website. The solutions manual as well as PowerPoint lectures are also available from the book’s website.

This page intentionally left blank

Acknowledgments

Writing a book is an effort on top of everything else that must keep running in parallel. Thus, writing is basically an early morning, after five, and over the weekends and holidays activity. It is a big effort that requires dedication and persistence. This would not be possible without the support of a number of people—people who helped in the simulations, in the making of the figures, in reading chapters, and in discussing various issues concerning all aspects, from proofs to the structure and the layout of the book.

First, I would like to express my gratitude to my mentor, friend, and colleague Nicholas Kalouptsidis, for this long-lasting and fruitful collaboration.

The cooperation with Kostas Slavakis over the last six years has been a major source of inspiration and learning and has played a decisive role for me in writing this book.

I am indebted to the members of my group, and in particular to Yannis Kopsinis, Pantelis Bouboulis, Simos Chouvardas, Kostas Themelis, George Papageorgiou, and Charis Georgiou. They were beside me the whole time, especially during the difficult final stages of the completion of the manuscript. My colleagues Aggelos Pikrakis, Kostas Koutroumbas, Dimitris Kosmopoulos, George Giannakopoulos, and Spyros Evaggelatos gave a lot of their time for discussions, helping in the simulations, and reading chapters.

Without my two sabbaticals during the spring semesters of 2011 and 2012, I doubt I would have ever finished this book. Special thanks to all my colleagues in the Department of Informatics and Telecommunications of the National and Kapodistrian University of Athens.

During my sabbatical in 2011, I was honored to be a holder of an Excellence Chair in Carlos III University of Madrid and spent the time with the group of Anibal Figueras-Vidal. I am indebted to Anibal for his invitation and all the fruitful discussions and the bottles of excellent red Spanish wine we had together. Special thanks to Jerónimo Arenas-García and Antonio Artés-Rodríguez, who have also introduced me to aspects of traditional Spanish culture.

During my sabbatical in 2012, I was also honored to be an Otto Mønsted Guest Professor at the Technical University of Denmark with the group of Lars Kai Hansen. I am indebted to him for the invitation and our enjoyable and insightful discussions, as well as his constructive comments on reviewing chapters of the book and for the visits to the Danish museums on weekends. Also, special thanks to Jan Larsen and Morten Mørup for the fruitful discussions.

A number of colleagues were kind enough to read and review chapters and parts of the book and come back with valuable comments and criticisms. My sincere thanks to Tulay Adali, Kostas Berberidis, Jim Bezdek, Gustavo Camps-Valls, Taylan Cemgil and his students, Petar Djuric, Paulo Diniz, Yannis Emiris, Georgios Giannakis, Mark Girolami, Dimitris Gunopoulos, Alexandros Katsioris, Evangelos Karkaletsis, Dimitris Katselis, Athanasios Liavas, Eleftherios Kofidis, Elias Koutsoupias, Alexandros Makris, Dimitris Manatakis, Elias Manolakos, Francisco Palmieri, Jean-Christophe Pesquet, Bhaskar Rao, Ali Sayed, Nicolas Sidiropoulos, Paris Smaragdis, Isao Yamada, and Zhilin Zhang.

Finally, I would like to thank Tim Pitts, the Editor in Academic Press, for all his help.

This page intentionally left blank

Notation

I have made an effort to keep a consistent mathematical notation throughout the book. Although every symbol is defined in the text prior to its use, it may be convenient for the reader to have the list of major symbols summarized together. The list is presented below:

- Vectors are denoted with **boldface** letters, such as \mathbf{x} .
- Matrices are denoted with capital letters, such as A .
- The determinant of a matrix is denoted as $\det\{A\}$, and sometimes as $|A|$.
- A diagonal matrix with elements a_1, a_2, \dots, a_l , in its diagonal is denoted as $A = \text{diag}\{a_1, a_2, \dots, a_l\}$.
- The identity matrix is denoted as I .
- The trace of a matrix is denoted as $\text{trace}\{A\}$.
- Random variables are denoted with roman fonts, such as x , and their corresponding values with *mathmode* letters, such as x .
- Similarly, random vectors are denoted with roman **boldface**, such as \mathbf{x} , and the corresponding values as x . The same is true for random matrices, denoted as X and their values as X .
- The vectors are assumed to be column-vectors. In other words,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_l \end{bmatrix} = \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(l) \end{bmatrix}$$

That is, the i th element of a vector can be represented either with a subscript x_i or as $x(i)$.

This is because the vectors may have already been given another subscript; \mathbf{x}_n , and the notation can be cluttered.

- Matrices are written as

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1l} \\ \vdots & \vdots & \ddots & \vdots \\ x_{l1} & x_{l2} & \dots & x_{ll} \end{bmatrix} = \begin{bmatrix} X(1, 1) & X(1, 2) & \dots & X(1, l) \\ \vdots & \vdots & \ddots & \vdots \\ X(l, 1) & X(l, 2) & \dots & X(l, l) \end{bmatrix}$$

- Transposition of a vector is denoted as \mathbf{x}^T and the Hermitian transposition as \mathbf{x}^H
- Complex conjugation of a complex number is denoted as x^* and also $\sqrt{-1} := j$. The symbol “:=” denotes definition.
- The set of real, complex, integer, and natural numbers is denoted as \mathbb{R} , \mathbb{C} , \mathbb{Z} , and \mathbb{N} , respectively.
- Sequences of numbers (vectors) are denoted as x_n (\mathbf{x}_n) or $x(n)$ ($\mathbf{x}(n)$).
- Functions are denoted with lower case letters, e.g., f , or in terms of their arguments, e.g., $f(x)$ or sometimes as $f(\cdot)$, if no specific argument is used.

This page intentionally left blank

$\Sigma\tau o \Delta\epsilon\sigma\pi o\iota\nu\grave{\alpha}\kappa i$
*For Everything
All These Years*

This page intentionally left blank

INTRODUCTION

1

CHAPTER OUTLINE

1.1 What Machine Learning is About	1
1.1.1 Classification	2
1.1.2 Regression	3
1.2 Structure and a Road Map of the Book	5
References	8

1.1 WHAT MACHINE LEARNING IS ABOUT

Learning through personal experience and knowledge, which propagates from generation to generation, is at the heart of human intelligence. Also, at the heart of any scientific field lies the development of models (often, they are called theories) in order to explain the available experimental evidence at each time period. In other words, we always *learn from data*. Different data and different focuses on the data give rise to different scientific disciplines.

This book is about learning from data; in particular, our intent is to detect and unveil a possible hidden structure and regularity patterns associated with their generation mechanism. This information in turn helps our analysis and understanding of the nature of the data, which can be used to make *predictions* for the future. Besides modeling the underlying structure, a major direction of significant interest in Machine Learning is to develop *efficient* algorithms for designing the models and also for analysis and prediction. The latter part is gaining importance in the dawn of what we call the *big data* era, when one has to deal with massive amounts of data, which may be represented in spaces of very large dimensionality. Analyzing data for such applications sets demands on algorithms to be computationally efficient and at the same time *robust* in their performance, because some of these data are contaminated with large noise and also, in some cases, the data may have missing values.

Such methods and techniques have been at the center of scientific research for a number of decades in various disciplines, such as Statistics and Statistical Learning, Pattern Recognition, Signal and Image Processing and Analysis, Computer Science, Data Mining, Machine Vision, Bioinformatics, Industrial Automation, and Computer-Aided Medical Diagnosis, to name a few. In spite of the different names, there is a common corpus of techniques that are used in all of them, and we will refer to such methods as Machine Learning. This name has gained popularity over the last decade or so. The name suggests the use of a machine/computer to learn in analogy to how the brain learns and predicts. In some cases, the methods are directly inspired by the way the brain works, as is the case with neural networks, covered in [Chapter 18](#).

Two problems at the heart of machine learning, which also comprise the backbone of this book, are the classification and the regression tasks.

1.1.1 CLASSIFICATION

The goal in classification is to assign an unknown *pattern* to one out of a number of classes that are considered to be known. For example, in X-ray mammography, we are given an image where a region indicates the existence of a tumor. The goal of a computer-aided diagnosis system is to predict whether this tumor corresponds to the *benign* or the *malignant* class. Optical character recognition (OCR) systems are also built around a classification system, in which the image corresponding to each letter of the alphabet has to be recognized and assigned to one of the twenty-four (for the Latin alphabet) classes; see [Section 18.11](#), for a related case study. Another example is the prediction of the authorship of a given text. Given a text written by an unknown author, the goal of a classification system is to predict the author among a number of authors (classes); this application is treated in [Section 11.15](#).

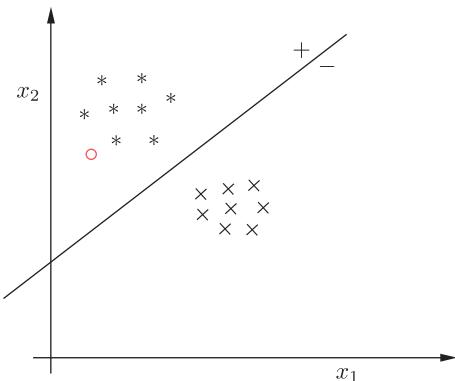
The first step in designing any machine learning task is to decide how to represent each pattern in the computer. This is achieved during the preprocessing stage; one has to “encode” related information that resides in the raw data (image pixels or strings of letters in the previous examples) in an efficient and information-rich way. This is usually done by transforming the raw data in a new space with each pattern represented by a vector, $\mathbf{x} \in \mathbb{R}^l$. This is known as the *feature vector*, and its l elements are known as the *features*. In this way, each pattern becomes a single point in an l -dimensional space, known as the *feature space* or the *input space*. We refer to this as the *feature generation* stage. Usually, one starts with some large value K of features and eventually selects the l most informative ones via an optimizing procedure known as the *feature selection* stage.

Having decided upon the input space, in which the data are represented, one has to train a classifier. This is achieved by first selecting a set of data whose class is known, which comprises the *training set*. This is a set of pairs, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, where y_n is the (output) variable denoting the class in which \mathbf{x}_n belongs, and it is known as the corresponding *class label*; the class labels, y , take values over a *discrete* set, $\{1, 2, \dots, M\}$, for an M -class classification task. For example, for a two-class classification task, $y_n \in \{-1, +1\}$. To keep our discussion simple, let us focus on the two-class case. Based on the training data, one then designs a function, f , which predicts the output label given the input; that is, given the measured values of the features. This function is known as the *classifier*. In general, we need to design a set of such functions.

Once the classifier has been designed, the system is ready for predictions. Given an unknown pattern, we form the corresponding feature vector, \mathbf{x} , from the raw data, and we plug this value into the classifier; depending on the value of $f(\mathbf{x})$ (usually on the respective sign, $\hat{y} = \text{sgn}f(\mathbf{x})$) the pattern is classified in one of the two classes. [Figure 1.1](#) illustrates the classification task. Initially, we are given the set of points, each representing a pattern in the two-dimensional space (two features used, x_1, x_2). Stars belong to one class, say ω_1 and the crosses to the other, ω_2 , in a two-class classification task. These are the training points. Based on these points, a classifier was learned; for our very simple case, this turned out to be a linear function,

$$f(\mathbf{x}) = \theta_1 x_1 + \theta_2 x_2 + \theta_0, \quad (1.1)$$

whose graph for all the points such as: $f(\mathbf{x}) = 0$, is the straight line shown in the figure. Then, we are given the point denoted by the red circle; this corresponds to the measured values from a pattern whose

**FIGURE 1.1**

The classifier (linear in this simple case) has been designed in order to separate the training data into the two classes, having on its positive side the points coming from one class and on its negative side those of the other. The “red” point, whose class is unknown, is classified to the same class as the “star” points, since it lies on the positive side of the classifier.

class is unknown to us. According to the classification system, which we have designed, this belongs to the same class as the points denoted by stars. Indeed, every point on one side of the straight line will give a positive value, $f(\mathbf{x}) > 0$, and all the points on its other side will give a negative value, $f(\mathbf{x}) < 0$. The point denoted with the red circle will then result in $f(\mathbf{x}) > 0$, as all the star points, and it is classified in the same class, ω_1 .

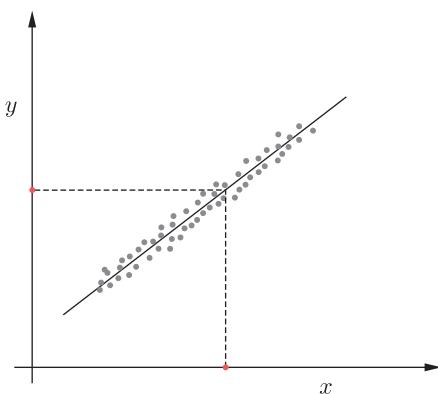
This type of learning is known as *supervised learning*, since a set of training data with known labels is available. Note that the training data can be seen as the available previous experience, and based on this, one builds a model to make predictions for the future. *Unsupervised/clustering* and *semisupervised* learning are not treated in this book, with the exception of the *k*-means algorithm, which is treated in [Chapter 12](#). Clustering and semisupervised learning are treated in detail in the companion books [1, 2].

Note that the receiver in a digital communications system can also be viewed as a classification system. Upon receiving the transmitted data, which have been contaminated by noise and also by other transformations imposed by the transmission channel ([Chapter 4](#)), one has to reach a decision on the value of the originally transmitted symbol. However, in digital communications, the transmitted symbols come from a finite alphabet, and each symbol defines a different class, ± 1 , for a binary transmitted sequence.

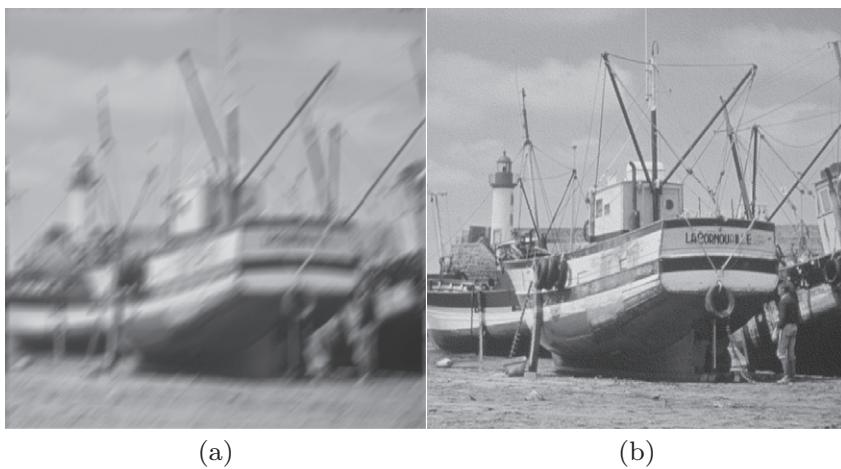
1.1.2 REGRESSION

The regression shares to a large extent the feature generation/selection stage, as described before; however, now the output variable, y , is *not* discrete but it takes values in an interval in the real axis or in a region in the complex numbers plane. The regression task is basically a curve fitting problem.

We are given a set of training points, (y_n, \mathbf{x}_n) , $y_n \in \mathbb{R}$, $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, and the task is to estimate a function, f , whose graph fits the data. Once we have found such a function, when an unknown point arrives, we can predict its output value. This is shown in [Figure 1.2](#).

**FIGURE 1.2**

Once a function (linear in this case), f , has been designed, for its graph to fit the available training data set in a regression task, given a new (red) point, x , the prediction of the associated output (red) value is given by $y = f(x)$.



(a)

(b)

FIGURE 1.3

(a) The blurred image, taken by a moving camera, and (b) its de-blurred estimate.

The training data in this case are the gray points. Once the curve fitting task has been completed, given a new point x (red), we are ready to predict its output value as $\hat{y} = f(x)$.

The regression task is a generic task that embraces a number of problems. For example, in financial applications one can predict tomorrow's stock market price given current market conditions and all other related information. Each piece of information is a measured value of a corresponding feature. Signal and image restoration come under this common umbrella of tasks. Signal and image de-noising can also be seen as a special type of a regression task. [Figure 1.3a](#) shows the case of a blurred image,

taken by a moving camera, and [Figure 1.3b](#) the de-blurred one (see [Chapter 4](#)). De-blurring is a typical image restoration task, where the de-blurred image is obtained as the output by feeding the blurred one as input to an appropriately designed function.

1.2 STRUCTURE AND A ROAD MAP OF THE BOOK

In the discussion above, we saw that seemingly different applications, e.g., authorship identification and channel equalization as well as financial prediction and image de-blurring can be treated in a unified framework. Many of the techniques that have been developed for machine learning are no different than techniques used in statistical signal processing or adaptive signal processing. Filtering comes under the general framework of regression ([Chapter 4](#)), and “adaptive filtering” is exactly the same as “online learning” in machine learning. As a matter of fact, as will be explained in more detail, this book can serve the needs of more than one advanced graduate or postgraduate course.

Over the years, a large number of techniques and “schools” have been developed, in the context of different applications. The two main paths are the Bayesian approach and the deterministic one. The former school considers the unknown parameters that define an unknown function, for example, $\theta_1, \theta_2, \theta_0$ in Eq. (1.1), as random variables, and the latter as having fixed, yet unknown, values. I respect both schools of thought, as I believe that there is more than one road that leads to the “truth.” Each can solve some problems more efficiently than the other, and vice versa. Maybe in a few years, the scene will be more clear and more definite conclusions can be drawn. Or it may turn out, as in life, that the “truth” is in the middle. It is interesting to note that one of the most powerful learning techniques of high interest currently is the deep learning approach (covered in [Chapter 18](#)), which is an interplay of probabilistic and deterministic arguments.

In any case, every newcomer to the field has to learn the basics and the classics. That’s why in this book, all major directions and methods will be discussed, in an equally balanced manner, to the greatest extent possible. Of course, the author, being human, could not avoid emphasizing the techniques with which he is most familiar. This is healthy, since writing a book is a means of sharing the author’s expertise and point of view with readers. This is why I strongly believe that a new book does not come to replace previous ones, but to complement previously published points of view.

[Chapter 2](#) is an introduction to probability and statistics. Random processes are also discussed. Readers who are familiar with such concepts can bypass this chapter. On the other hand, one can focus on different parts of this chapter. Readers who would like to focus on statistical signal processing/adaptive processing can focus more on the random processes part. Those who would like to follow a probabilistic machine learning point of view would find the part presenting the various distributions more important. In any case, the Gaussian distribution is a must for those who are not yet familiar with it.

[Chapter 3](#) is an overview of the parameter estimation task. This is a chapter that presents an overview of the book and defines the main concepts that run across its pages. This chapter has also been written to stand alone as an introduction to machine learning. Although it is my feeling that all of it should be read and taught, depending on the focus of the course and taking into account the omnipresent time limitations, one can focus more on the parts of her or his interest. Both the deterministic as well as the probabilistic approaches are defined and discussed. In any case, the parts dealing with the definition of the inverse problems, the bias-variance trade-off, and the concepts of generalization and regularization are a must.

[Chapter 4](#) is dedicated to the Mean-Square Error (MSE) linear estimation. For those following a statistical signal processing (SP) course, all of the chapter is important. The rest of the readers can bypass the parts related to complex-valued processing and also the part dealing with computational complexity issues, since this is only of importance if the input data are random processes. Bypassing this part will not affect reading later parts of the chapter that deal with the MSE of linear models, the Gauss-Markov theorem, and the Kalman filtering.

[Chapter 5](#) introduces the stochastic gradient descent family of algorithms. The first part, dealing with the stochastic approximation method, is a must for every reader. The rest of the chapter, which deals with the Least-Mean-Squares (LMS) algorithm and its offsprings, is more appropriate for readers who are interested in a statistical SP course, since these families are suited for tracking time varying environments. This may not be the first priority for readers who are interested in classification and machine learning tasks with data whose statistical properties are not time varying.

[Chapter 6](#) is dedicated to the Least-Squares (LS) cost function, which is of interest to all readers in machine learning and signal processing. The latter part dealing with the total least-squares method can be bypassed in a first reading. Emphasis is also put on ridge regression and its geometric interpretation. Ridge regression is important to the newcomer, since he/she becomes familiar with the concept of regularization; this is an important aspect in any machine learning task, tied directly with the generalization performance of the designed predictor.

I have decided to compress the part dealing with fast LS algorithms, which are appropriate when the input is a random process/signal that imposes a special structure on the involved covariance matrices, into a discussion section. It is the author's feeling that this is of no greater interest than it was a decade or two ago. Also, the main idea, that of a highly structured covariance matrix that lies behind the fast algorithms, is discussed in some detail in [Chapter 4](#), in the context of Levinson's algorithm and its lattice and lattice-ladder by-products.

[Chapter 7](#) is a must for any machine learning course. Courses on statistical SP can also accommodate the first part of the chapter dealing with the classical Bayesian classification—the classical Bayesian decision theory. This chapter introduces the first case study of the book and it concerns the protein folding prediction task.

The aforementioned six chapters comprise the part of the book that deals with more or less classical topics. The rest of the chapters deal with more advanced techniques and can fit with any course dealing with machine learning or statistical/adaptive signal processing, depending on the focus, the time constraints, and the background of the audience.

[Chapter 8](#) deals with convexity, a topic that is receiving more and more attention recently. The chapter presents the basic definitions concerning convex sets and functions and the notion of projection. These are important tools used in a number of recently developed algorithms. Also, the classical projections over convex sets (POCS) algorithm and the set theoretic approach to online learning are discussed as an alternative to gradient-descent based schemes. Then, the task of optimization of non-smooth convex loss functions is discussed, and the family of proximal mapping, alternating direction method of multipliers (ADMM), and forward backward-splitting methods are presented. This is a chapter that can be used when the emphasis of the course is optimization. Employing non-smooth loss functions and/or non-smooth regularization terms, in place of the LS and its ridge regression relative, is a trend of high research and practical interest.

[Chapters 9](#) and [10](#) deal with sparse modeling. The first of the two chapters introduces the main concepts and ideas and the second deals with algorithms for batch as well as online learning scenarios. Also, in the second chapter, a case study in the context of time-frequency analysis is discussed. Depending on time constraints, the main concepts behind sparse modeling and compressed sensing can be taught in a related course. These two chapters can also be used as a specialized course on sparsity on a postgraduate level.

[Chapter 11](#) deals with learning in reproducing kernel Hilbert spaces and nonlinear techniques. The first part of the chapter is a must for any course with an emphasis on classification. The support vector regression and support vector machines are treated in detail. Moreover, a course on statistical SP with an emphasis on nonlinear modeling can also include material and concepts from this chapter. Kernelized versions of the stochastic gradient descent rationale are treated in some detail as nonlinear versions of classical online algorithms. A case study dealing with authorship identification is discussed at the end of this chapter.

[Chapters 12](#) and [13](#) deal with Bayesian learning. Thus, both chapters can become the backbone of a course on machine learning and statistical SP that intends to emphasize Bayesian methods. The former of the chapters deals with the basic principles and it is an introduction to the expectation-maximization (EM) algorithm. The use of this celebrated algorithm is demonstrated in the context of two classical applications, that of the linear regression and the Gaussian mixture modeling for probability density function estimation. The second chapter deals with approximate inference techniques, and one can use parts of it, depending on the time constraints and the background of the audience. Sparse Bayesian learning and the relevant vector machine (RVM) framework is introduced. At the end of this chapter, Gaussian processes and nonparametric Bayesian techniques are discussed, and a case study concerning hyper-spectral image unmixing is presented. Both chapters, in their full length, can be used as a specialized course on Bayesian learning.

[Chapters 14](#) and [17](#) deal with Monte Carlo sampling methods. The latter chapter deals with particle filtering. Both chapters, together with the two previous ones that deal with Bayesian learning, can be combined in a course whose emphasis is on statistical methods of machine learning/statistical signal processing.

[Chapters 15](#) and [16](#) deal with probabilistic graphical models. The former chapter introduces the main concepts and definitions, and at the end introduces the message passage algorithm for chains and trees. This chapter is a must for any course whose emphasis is on probabilistic graphical models. The latter of the two chapters deals with message passage algorithms on junction trees and then with approximate inference techniques. Dynamic graphical models and hidden Markov models (HMM) are introduced. The Baum-Welch and the Viterbi schemes are derived as special cases of message passage algorithms by treating HMM as a special instance of a junction tree.

[Chapter 18](#) deals with neural networks and deep learning. This chapter is also a must in any course with an emphasis on classification. The perceptron algorithm and the backpropagation algorithms are discussed in detail, and then the discussion moves into deep architectures and their training. A case study in the context of optical character recognition is discussed.

[Chapter 19](#) is on dimensionality reduction techniques and latent variable modeling. The methods of principle component analysis (PCA), canonical correlations analysis (CCA), and independent component analysis (ICA) are introduced. The probabilistic approach to latent variable modeling is

discussed, and the probabilistic PCA (PPCA) is presented. Then, the focus turns to dictionary learning and robust PCA. Nonlinear dimensionality reduction techniques such as kernel PCA are discussed, along with the methods of local linear embedding (LLE) and isometric mapping (ISOMAP). Finally, a case study in the context of functional magnetic resonance imaging (fMRI) data analysis, based on ICA, is presented.

Each chapter starts with the basics and moves on to cover more recent advances in the related topic. This is true mainly for the text beginning with [Chapter 7](#), since the first six chapters cover more classical material.

In summary, we provide the following suggestions for different courses, depending on the emphasis that the instructor wants to place on various topics.

- Machine Learning with emphasis on classification:
 - Main chapters: [3](#), [7](#), [11](#), and [18](#)
 - Secondary chapters: [12](#) and [13](#), and possibly the first part of [6](#)
- Statistical Signal Processing:
 - Main chapters: [3](#), [4](#), [6](#), and [12](#)
 - Secondary chapters: [5](#) (first part) and [13–17](#)
- Machine Learning with emphasis on Bayesian techniques:
 - Main chapters: [3](#) and [12–14](#)
 - Secondary chapters: [7](#), [15](#), and [16](#), and possibly the first part of [6](#)
- Adaptive Signal Processing:
 - Main chapters: [3–6](#)
 - Secondary chapters: [8](#), [9](#), [11](#), [14](#), and [17](#)

I believe that the above suggestions of following various combinations of chapters is possible, since the book has been written in such a way as to make individual chapters as self-contained as possible.

At the end of most of the chapters there are Matlab exercises, mainly based on the various examples given in the text. The required Matlab code is available on the book's website, together with the solutions manual. Also, all figures of the book are available on the book's website.

REFERENCES

- [1] [S. Theodoridis, K. Koutroumbas, Pattern Recognition](#), fourth ed., Academic Press, Amsterdam, 2009.
- [2] S. Theodoridis, A. Pikrakis, K. Koutroumbas, D. Cavouras, [Introduction to Pattern Recognition: A MATLAB Approach](#), Academic Press, Amsterdam, 2010.

PROBABILITY AND STOCHASTIC PROCESSES

2

CHAPTER OUTLINE

2.1 Introduction	9
2.2 Probability and Random Variables	9
2.2.1 Probability	9
<i>Relative Frequency Definition</i>	10
<i>Axiomatic Definition</i>	10
2.2.2 Discrete Random Variables	11
<i>Joint and Conditional Probabilities.....</i>	11
<i>Bayes Theorem</i>	12
2.2.3 Continuous Random Variables	13
2.2.4 Mean and Variance	14
<i>Complex Random Variables</i>	15
2.2.5 Transformation of Random Variables	16
2.3 Examples of Distributions	17
2.3.1 Discrete Variables	17
<i>The Bernoulli Distribution</i>	17
<i>The Binomial Distribution</i>	17
<i>The Multinomial Distribution</i>	18
2.3.2 Continuous Variables	19
<i>The Uniform Distribution</i>	19
<i>The Gaussian Distribution</i>	19
<i>The Central Limit Theorem</i>	23
<i>The Exponential Distribution</i>	24
<i>The Beta Distribution.....</i>	24
<i>The Gamma Distribution</i>	25
<i>The Dirichlet Distribution</i>	26
2.4 Stochastic Processes	28
2.4.1 First and Second Order Statistics	29
2.4.2 Stationarity and Ergodicity	29
2.4.3 Power Spectral Density	32
<i>Properties of the Autocorrelation Sequence.....</i>	32
<i>Power Spectral Density.....</i>	34
<i>Transmission Through a Linear System</i>	34

<i>Physical Interpretation of the PSD</i>	36
2.4.4 Autoregressive Models	37
2.5 Information Theory	40
2.5.1 Discrete Random Variables	41
<i>Information</i>	41
<i>Mutual and Conditional Information</i>	42
<i>Entropy and Average Mutual Information</i>	43
2.5.2 Continuous Random Variables	44
<i>Average Mutual Information and Conditional Information</i>	46
<i>Relative Entropy or Kullback-Leibler Divergence</i>	46
2.6 Stochastic Convergence	47
<i>Convergence Everywhere</i>	47
<i>Convergence Almost Everywhere</i>	48
<i>Convergence in the Mean-Square Sense</i>	48
<i>Convergence in Probability</i>	48
<i>Convergence in Distribution</i>	48
Problems	48
References	50

2.1 INTRODUCTION

The goal of this chapter is to provide the basic definitions and properties related to probability theory and stochastic processes. It is assumed that the reader has attended a basic course on probability and statistics prior to reading this book. So, the aim is to help the reader refresh her/his memory and to establish a common language and a commonly understood notation.

Besides probability and random variables, random processes will be briefly reviewed and some basic theorems will be stated. Finally, at the end of the chapter, basic definitions and properties related to information theory will be summarized.

The reader who is familiar with all these notions can bypass this chapter.

2.2 PROBABILITY AND RANDOM VARIABLES

A random variable, x , is a variable whose variations are due to chance/randomness. A random variable can be considered as a function, which assigns a value to the outcome of an experiment. For example, in a coin tossing experiment, the corresponding random variable, x , can assume the values $x_1 = 0$ if the result of the experiment is “heads” and $x_2 = 1$ if the result is “tails.”

We will denote a random variable with a lower case roman, such as x , and the values it takes once an experiment has been performed, with mathmode italics, such as x .

A random variable is described in terms of a set of *probabilities* if its values are of a discrete nature, or in terms of a *probability density function* (pdf) if its values lie anywhere within an interval of the real axis (non-countably infinite set). For a more formal treatment and discussion, see [4, 6].

2.2.1 PROBABILITY

Although the words “probability” and “probable” are quite common in our everyday vocabulary, the mathematical definition of probability is not a straightforward one, and there are a number of different definitions that have been proposed over the years. Needless to say, whatever definition is adopted, the end result is that the properties and rules, which are derived, remain the same. Two of the most commonly used definitions are:

Relative frequency definition

The probability, $P(A)$, of an event, A , is the limit

$$P(A) = \lim_{n \rightarrow \infty} \frac{n_A}{n}, \quad (2.1)$$

where n is the number of total trials and n_A the number of times event A occurred. The problem with this definition is that in practice in any physical experiment, the numbers n_A and n can be large, yet they are always finite. Thus, the limit can only be used as a *hypothesis* and not as something that can be attained experimentally. In practice, often, we use

$$P(A) \approx \frac{n_A}{n} \quad (2.2)$$

for large values of n . However, this has to be used with caution, especially when the probability of an event is very small.

Axiomatic definition

This definition of probability is traced back to 1933 to the work of Andrey Kolmogorov, who found a close connection between probability theory and the mathematical theory of sets and functions of a real variable, in the context of measure theory, as noted in [5].

The probability, $P(A)$, of an event is a nonnegative number assigned to this event, or

$$P(A) \geq 0. \quad (2.3)$$

The probability of an event, C , which is certain to occur, equals to one

$$P(C) = 1. \quad (2.4)$$

If two events, A and B , are mutually exclusive (they cannot occur simultaneously), then the probability of occurrence of either A or B (denoted as $A \cup B$) is given by

$$P(A \cup B) = P(A) + P(B). \quad (2.5)$$

It turns out that these three defining properties, which can be considered as the respective *axioms*, suffice to develop the rest of the theory. For example, it can be shown that the probability of an impossible event is equal to zero, as noted in [6].

The previous two approaches for defining probability are not the only ones. Another interpretation, which is in line with the way we are going to use the notion of probability in a number of places in this book in the context of *Bayesian learning*, has been given by Cox [2]. There, probability was seen as a measure of *uncertainty* concerning an event. Take, for example, the uncertainty whether the Minoan civilization was destroyed as a consequence of the earthquake that happened close to the island of Santorini. This is obviously not an event whose probability can be tested with repeated trials. However,

putting together historical as well as scientific evidence, we can quantify our expression of uncertainty concerning such a conjecture. Also, we can modify the degree of our uncertainty once more historical evidence comes to light due to new archeological findings. Assigning numerical values to represent degrees of belief, Cox developed a set of axioms encoding common sense properties of such beliefs, and he came to a set of rules equivalent to the ones we are going to review soon; see also [4].

The origins of probability theory are traced back to the middle 17th century in the works of Pierre Fermat (1601-1665), Blaise Pascal (1623-1662), and Christian Huygens (1629-1695). The concepts of probability and the mean value of a random variable can be found there. The motivation for developing the theory is not related to any purpose for “serving society”; the purpose was to serve the needs of gambling and games of chance!

2.2.2 DISCRETE RANDOM VARIABLES

A discrete random variable, x , can take any value from a finite or *countably* infinite set \mathcal{X} . The probability of the event, “ $x = x \in \mathcal{X}$,” is denoted as

$$P(x = x) \quad \text{or simply } P(x). \quad (2.6)$$

The function $P(\cdot)$ is known as the *probability mass function* (pmf). Being a probability, it has to satisfy the first axiom, so $P(x) \geq 0$. Assuming that no two values in \mathcal{X} can occur simultaneously and that after any experiment a single value will always occur, the second and third axioms combined give

$$\sum_{x \in \mathcal{X}} P(x) = 1. \quad (2.7)$$

The set \mathcal{X} is also known as the *sample* or *state space*.

Joint and conditional probabilities

The *joint probability* of two events, A, B , is the probability that both events occur simultaneously, and it is denoted as $P(A, B)$. Let us now consider two random variables, x, y , with sample spaces $\mathcal{X} = \{x_1, \dots, x_{n_x}\}$ and $\mathcal{Y} = \{y_1, \dots, y_{n_y}\}$, respectively. Let us adopt the relative frequency definition and assume that we carry out n experiments and that each one of the values in \mathcal{X} occurred $n_1^x, \dots, n_{n_x}^x$ times and each one of the values in \mathcal{Y} occurred $n_1^y, \dots, n_{n_y}^y$ times, respectively. Then,

$$P(x_i) \approx \frac{n_i^x}{n}, \quad i = 1, 2, \dots, n_x, \quad \text{and} \quad P(y_j) \approx \frac{n_j^y}{n}, \quad j = 1, 2, \dots, n_y.$$

Let us denote by n_{ij} the number of times the values x_i and y_j occurred simultaneously. Then, $P(x_i, y_j) \approx \frac{n_{ij}}{n}$. Simple reasoning dictates that the total number, n_i^x , that value x_i occurred, is equal to

$$n_i^x = \sum_{j=1}^{n_y} n_{ij}. \quad (2.8)$$

Dividing both sides in the above by n , the following *sum rule* readily results.

$$P(x) = \sum_{y \in \mathcal{Y}} P(x, y) : \quad \text{Sum Rule.} \quad (2.9)$$

The *conditional probability* of an event, A , given another event, B , is denoted as $P(A|B)$ and it is defined as

$$P(A|B) := \frac{P(A, B)}{P(B)} : \text{ Conditional Probability,} \quad (2.10)$$

provided $P(B) \neq 0$. It can be shown that this is indeed a probability, in the sense that it respects all three axioms [6]. We can better grasp its physical meaning if the relative frequency definition is adopted. Let n_{AB} be the number of times that both events occurred simultaneously, and n_B the times event B occurred, out of n experiments. Then, we have

$$P(A|B) = \frac{n_{AB}}{n} \frac{n}{n_B} = \frac{n_{AB}}{n_B}. \quad (2.11)$$

In other words, the conditional probability of an event, A , given another one, B , is the relative frequency that A occurred, not with respect to the total number of experiments performed, but relative to the times event B occurred.

Viewed differently and adopting similar notation in terms of random variables, in conformity with Eq. (2.9), the definition of the conditional probability is also known as the *product rule* of probability, written as

$$P(x, y) = P(x|y)P(y) : \text{ Product Rule.} \quad (2.12)$$

To differentiate from the joint and conditional probabilities, probabilities, $P(x)$ and $P(y)$ are known as *marginal probabilities*.

Statistical Independence: Two random variables are said to be statistically independent *if and only if* their joint probability is written as the product of the respective marginals,

$$P(x, y) = P(x)P(y). \quad (2.13)$$

Bayes theorem

Bayes theorem is a direct consequence of the product rule and of the symmetry property of the joint probability, $P(x, y) = P(y, x)$, and it is stated as

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} : \text{ Bayes Theorem,} \quad (2.14)$$

where the marginal, $P(x)$, can be written as

$$P(x) = \sum_{y \in \mathcal{Y}} P(x, y) = \sum_{y \in \mathcal{Y}} P(x|y)P(y),$$

and it can be considered as the normalizing constant of the numerator on the right-hand side in Eq. (2.14), which guarantees that summing up $P(y|x)$ with respect to all possible values of $y \in \mathcal{Y}$ results in one.

Bayes theorem plays a central role in machine learning, and it will be the basis for developing Bayesian techniques for estimating the values of unknown parameters.

2.2.3 CONTINUOUS RANDOM VARIABLES

So far, we have focused on discrete random variables. Our interest now turns to the extension of the notion of probability to random variables, which take values on the real axis, \mathbb{R} .

The starting point is to compute the probability of a random variable, x , to lie in an interval, $x_1 < x \leq x_2$. Note that the two events, $x \leq x_1$ and $x_1 < x \leq x_2$, are mutually exclusive. Thus, we can write that

$$P(x \leq x_1) + P(x_1 < x \leq x_2) = P(x \leq x_2). \quad (2.15)$$

Define the *cumulative distribution function* (cdf) of x , as

$$F_x(x) := P(x \leq x) : \text{Cumulative Distribution Function.} \quad (2.16)$$

Then, Eq. (2.15) can be written as

$$P(x_1 < x \leq x_2) = F_x(x_2) - F_x(x_1). \quad (2.17)$$

Note that F_x is a monotonically increasing function. Furthermore, if it is continuous, the random variable x is said to be of a *continuous* type. Assuming that it is also differentiable, we can define the *pdf* (pdf) of x as

$$p_x(x) := \frac{dF_x(x)}{dx} : \text{Probability Density Function,} \quad (2.18)$$

which then leads to

$$P(x_1 < x \leq x_2) = \int_{x_1}^{x_2} p_x(x) dx. \quad (2.19)$$

Also,

$$F_x(x) = \int_{-\infty}^x p_x(z) dz. \quad (2.20)$$

Using familiar logic from calculus arguments, the pdf can be interpreted as

$$\Delta P(x < x \leq x + \Delta x) \approx p_x(x) \Delta x, \quad (2.21)$$

which justifies its name as a “density” function, being the probability (ΔP) of x lying in a small interval Δx , divided by the length of this interval. Note that as $\Delta x \rightarrow 0$ this probability tends to zero. Thus, the probability of a continuous random variable taking any single value is zero. Moreover, since $P(-\infty < x < +\infty) = 1$, we have

$$\int_{-\infty}^{+\infty} p_x(x) dx = 1. \quad (2.22)$$

Usually, in order to simplify notation, the subscript x is dropped and we write $p(x)$, unless it is necessary for avoiding possible confusion. Note, also, that we have adopted the lower case “ p ” to denote a pdf and the capital “ P ” to denote a probability.

All previously stated rules for the probability are readily carried out for the case of pdfs, in the following way

$$p(x|y) = \frac{p(x,y)}{p(y)}, \quad p(x) = \int_{-\infty}^{+\infty} p(x,y) dy. \quad (2.23)$$

2.2.4 MEAN AND VARIANCE

Two of the most common and useful quantities associated with any random variable are the respective mean value and variance. The mean value (or sometimes called expected value) is denoted as

$$\mathbb{E}[x] := \int_{-\infty}^{+\infty} xp(x) dx : \text{ Mean Value,} \quad (2.24)$$

where for discrete random variables the integration is replaced by summation ($\mathbb{E}[x] = \sum_{x \in \mathcal{X}} xP(x)$).

The variance is denoted as σ_x^2 and it is defined as

$$\sigma_x^2 := \int_{-\infty}^{+\infty} (x - \mathbb{E}[x])^2 p(x) dx : \text{ Variance,} \quad (2.25)$$

where integration is replaced by summation for discrete variables. The variance is a measure of the spread of the values of the random variable around its mean value.

The definition of the mean value is generalized for any function, $f(x)$, i.e.,

$$\mathbb{E}[f(x)] := \int_{-\infty}^{+\infty} f(x)p(x)dx. \quad (2.26)$$

It is readily shown that the mean value with respect to two random variables, y, x , can be written as the product

$$\mathbb{E}_{x,y}[f(x, y)] = \mathbb{E}_x [\mathbb{E}_{y|x}[f(x, y)]]. \quad (2.27)$$

This is a direct consequence of the definition of the mean value and the product rule of probability. Given two random variables x, y , their *covariance* is defined as

$$\text{cov}(x, y) := \mathbb{E}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])], \quad (2.28)$$

and their *correlation* as

$$r_{xy} := \mathbb{E}[xy] = \text{cov}(x, y) + \mathbb{E}[x]\mathbb{E}[y]. \quad (2.29)$$

A *random vector* is a collection of random variables, $\mathbf{x} = [x_1, \dots, x_l]^T$, and $p(\mathbf{x})$ is the joint pdf (probability for discrete variables),

$$p(\mathbf{x}) = p(x_1, \dots, x_l). \quad (2.30)$$

The *covariance matrix* of a random vector, \mathbf{x} , is defined as

$$\text{Cov}(\mathbf{x}) := \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] : \text{ Covariance Matrix,} \quad (2.31)$$

or

$$\text{Cov}(\mathbf{x}) = \begin{bmatrix} \text{cov}(x_1, x_1) & \dots & \text{cov}(x_1, x_l) \\ \vdots & \ddots & \vdots \\ \text{cov}(x_l, x_1) & \dots & \text{cov}(x_l, x_l) \end{bmatrix}. \quad (2.32)$$

Similarly, the *correlation matrix* of a random vector, \mathbf{x} , is defined as

$$R_x := \mathbb{E} [\mathbf{x}\mathbf{x}^T] : \quad \text{Correlation Matrix,} \quad (2.33)$$

or

$$\begin{aligned} R_x &= \begin{bmatrix} \mathbb{E}[x_1, x_1] & \dots & \mathbb{E}[x_1, x_l] \\ \vdots & \ddots & \vdots \\ \mathbb{E}[x_l, x_1] & \dots & \mathbb{E}[x_l, x_l] \end{bmatrix} \\ &= \text{Cov}(\mathbf{x}) + \mathbb{E}[\mathbf{x}] \mathbb{E}[\mathbf{x}^T]. \end{aligned} \quad (2.34)$$

Both the covariance and correlation matrices have a very rich structure, which will be exploited in various parts of this book to lead to computational savings whenever they are present in calculations. For the time being, observe that both are symmetric and positive semidefinite. The symmetry, $\Sigma = \Sigma^T$, is readily deduced from the definition. An $l \times l$ symmetric matrix, A , is called *positive semidefinite* if

$$\mathbf{y}^T A \mathbf{y} \geq 0, \quad \forall \mathbf{y} \in \mathbb{R}^l. \quad (2.35)$$

If the inequality is a strict one, the matrix is said to be *positive definite*. For the covariance matrix, we have

$$\mathbf{y}^T \mathbb{E} [(\mathbf{x} - \mathbb{E}[\mathbf{x}]) (\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] \mathbf{y} = \mathbb{E} \left[\left(\mathbf{y}^T (\mathbf{x} - \mathbb{E}[\mathbf{x}]) \right)^2 \right] \geq 0,$$

and the claim has been proved.

Complex random variables

A complex random variable, $z \in \mathbb{C}$, is a sum

$$z = x + jy, \quad (2.36)$$

where x, y are real random variables and $j := \sqrt{-1}$. Note that for complex random variables, the pdf *cannot* be defined since inequalities of the form, $x + jy \leq x + jy$, have no meaning. When we write $p(z)$, we mean the joint pdf of the real and imaginary parts, expressed as

$$p(z) := p(x, y). \quad (2.37)$$

For complex random variables, the notions of mean and covariance are defined as

$$\mathbb{E}[z] := \mathbb{E}[x] + j\mathbb{E}[y], \quad (2.38)$$

and

$$\text{cov}(z_1, z_2) := \mathbb{E} [(z_1 - \mathbb{E}[z_1])(z_2 - \mathbb{E}[z_2])^*], \quad (2.39)$$

where “ $*$ ” denotes complex conjugation. The latter definition leads to the variance of a complex variable,

$$\sigma_z^2 = \mathbb{E} [|z - \mathbb{E}[z]|^2] = \mathbb{E} [|z|^2] - |\mathbb{E}[z]|^2. \quad (2.40)$$

Similarly, for complex random vectors, $\mathbf{z} = \mathbf{x} + j\mathbf{y} \in \mathbb{C}^l$, we have

$$p(\mathbf{z}) := p(x_1, \dots, x_l, y_1, \dots, y_l), \quad (2.41)$$

where $x_i, y_i, i = 1, 2, \dots, l$, are the components of the involved real vectors, respectively. The covariance and correlation matrices are similarly defined as

$$\text{Cov}(\mathbf{z}) := \mathbb{E} \left[(\mathbf{z} - \mathbb{E}[\mathbf{z}]) (\mathbf{z} - \mathbb{E}[\mathbf{z}])^H \right], \quad (2.42)$$

where “ H ” denotes the Hermitian (transposition and conjugation) operation.

For the rest of the chapter, we are going to deal mainly with real random variables. Whenever needed, differences with the case of complex variables will be stated.

2.2.5 TRANSFORMATION OF RANDOM VARIABLES

Let \mathbf{x} and \mathbf{y} be two random vectors, which are related via the vector transform,

$$\mathbf{y} = f(\mathbf{x}), \quad (2.43)$$

where $f : \mathbb{R}^l \mapsto \mathbb{R}^l$ is an *invertible* transform. That is, given \mathbf{y} , then $\mathbf{x} = f^{-1}(\mathbf{y})$ can be uniquely obtained. We are given the joint pdf, $p_{\mathbf{x}}(\mathbf{x})$, of \mathbf{x} and the task is to obtain the joint pdf, $p_{\mathbf{y}}(\mathbf{y})$, of \mathbf{y} .

The Jacobian matrix of the transformation is defined as

$$J(\mathbf{y}; \mathbf{x}) := \frac{\partial(y_1, y_2, \dots, y_l)}{\partial(x_1, x_2, \dots, x_l)} := \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_1}{\partial x_l} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_l}{\partial x_1} & \dots & \frac{\partial y_l}{\partial x_l} \end{bmatrix}. \quad (2.44)$$

Then, it can be shown (e.g., [6]) that

$$p_{\mathbf{y}}(\mathbf{y}) = \frac{p_{\mathbf{x}}(\mathbf{x})}{|\det(J(\mathbf{y}; \mathbf{x}))|} \Big|_{\mathbf{x}=f^{-1}(\mathbf{y})}, \quad (2.45)$$

where $|\det(\cdot)|$ denotes the absolute value of the determinant of a matrix. For real random variables, as in $\mathbf{y} = f(\mathbf{x})$, Eq. (2.45) simplifies to

$$p_{\mathbf{y}}(\mathbf{y}) = \frac{p_{\mathbf{x}}(\mathbf{x})}{\left| \frac{dy}{dx} \right|} \Big|_{\mathbf{x}=f^{-1}(\mathbf{y})}. \quad (2.46)$$

The latter can be graphically understood from Figure 2.1. The following two events have equal probabilities,

$$P(x < \mathbf{x} \leq \mathbf{x} + \Delta\mathbf{x}) = P(y + \Delta y < \mathbf{y} \leq \mathbf{y}), \quad \Delta\mathbf{x} > 0, \quad \Delta\mathbf{y} < 0.$$

Hence, by the definition of a pdf we have

$$p_{\mathbf{y}}(\mathbf{y}) |\Delta\mathbf{y}| = p_{\mathbf{x}}(\mathbf{x}) |\Delta\mathbf{x}|, \quad (2.47)$$

which leads to Eq. (2.46).

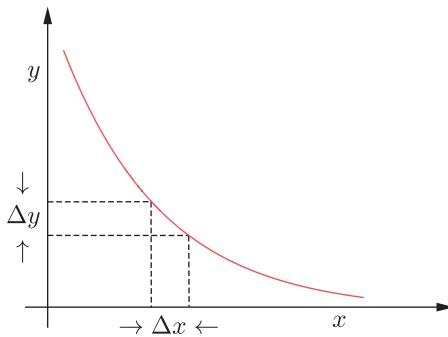
Example 2.1. Let us consider random vectors that are related via the linear transform,

$$\mathbf{y} = A\mathbf{x}, \quad (2.48)$$

where A is invertible. Compute the joint pdf of \mathbf{y} in terms of $p_{\mathbf{x}}(\mathbf{x})$.

The Jacobian of the transformation is easily computed and given by

$$J(\mathbf{y}; \mathbf{x}) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = A.$$

**FIGURE 2.1**

Note that by the definition of a pdf, $p_y(y)|\Delta y| = p_x(x)|\Delta x|$.

Hence,

$$p_y(y) = \frac{p_x(A^{-1}y)}{|\det(A)|}. \quad (2.49)$$

2.3 EXAMPLES OF DISTRIBUTIONS

In this section, some notable examples of distributions are provided. These are popular for modeling the random nature of variables met in a wide range of applications, and they will be used later in this book.

2.3.1 DISCRETE VARIABLES

The Bernoulli distribution

A random variable is said to be distributed according to a Bernoulli distribution if it is binary, $\mathcal{X} = \{0, 1\}$, with

$$P(x = 1) = p, \quad P(x = 0) = 1 - p.$$

In a more compact way, we write $x \sim \text{Bern}(x|p)$ where

$$P(x) = \text{Bern}(x|p) := p^x(1 - p)^{1-x}. \quad (2.50)$$

Its mean value is equal to

$$\mathbb{E}[x] = 1p + 0(1 - p) = p \quad (2.51)$$

and its variance is equal to

$$\sigma_x^2 = (1 - p)^2 p + p^2(1 - p) = p(1 - p). \quad (2.52)$$

The Binomial distribution

A random variable, x , is said to follow a binomial distribution with parameters n, p , and we write $x \sim \text{Bin}(x|n, p)$ if $\mathcal{X} = \{0, 1, \dots, n\}$ and

$$P(x = k) := \text{Bin}(k|n, p) = \binom{n}{k} p^k (1-p)^{n-k}, \quad k = 0, 1, \dots, n, \quad (2.53)$$

where by definition

$$\binom{n}{k} := \frac{n!}{(n-k)!k!}. \quad (2.54)$$

For example, this distribution models the times that heads occurs in n successive trials, where $P(\text{Heads}) = p$. The binomial is a generalization of the Bernoulli distribution, which results if in Eq. (2.53) we set $n = 1$. The mean and variance of the binomial distribution are (Problem 2.1)

$$\mathbb{E}[x] = np, \quad (2.55)$$

and

$$\sigma_x^2 = np(1-p). \quad (2.56)$$

Figure 2.2a shows the probability $P(k)$ as a function of k for $p = 0.4$ and $n = 9$. Figure 2.2b shows the respective cumulative distribution. Observe that the latter has a staircase form, as is always the case for discrete variables.

The Multinomial distribution

This is a generalization of the binomial distribution if the outcome of each experiment is not binary but can take one out of K possible values. For example, instead of tossing a coin, a die with K sides is thrown. Each one of the possible K outcomes has probability P_1, P_2, \dots, P_K , respectively, to occur, and we denote

$$\mathbf{P} = [P_1, P_2, \dots, P_K]^T.$$

After n experiments, assume that x_1, x_2, \dots, x_K times sides $x = 1, x = 2, \dots, x = K$ occurred, respectively. We say that the random (discrete) vector,

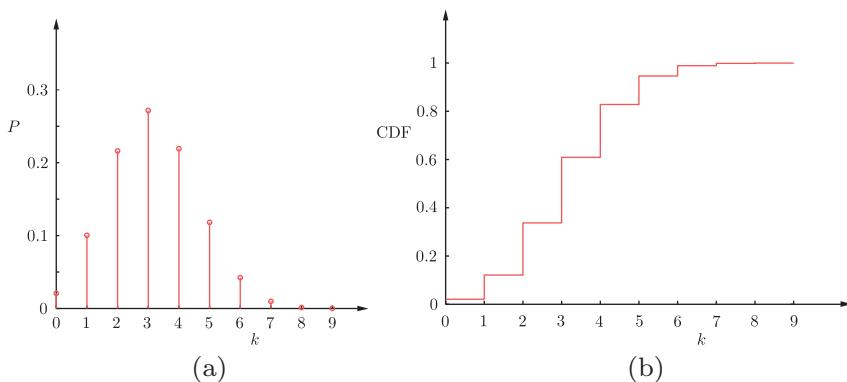


FIGURE 2.2

(a) The probability mass function (pmf) for the binomial distribution for $p = 0.4$ and $n = 9$. (b) The respective cumulative probability distribution (cdf). Since the random variable is discrete, the cdf has a staircase-like graph.

$$\mathbf{x} = [x_1, x_2, \dots, x_K]^T, \quad (2.57)$$

follows a multinomial distribution, $\mathbf{x} \sim \text{Mult}(\mathbf{x}|n, \mathbf{P})$, if

$$P(\mathbf{x}) = \text{Mult}(\mathbf{x}|n, \mathbf{P}) := \binom{n}{x_1, x_2, \dots, x_K} \prod_{k=1}^K P_k^{x_k}, \quad (2.58)$$

where

$$\binom{n}{x_1, x_2, \dots, x_K} := \frac{n!}{x_1! x_2! \dots x_K!}.$$

Note that the variables, x_1, \dots, x_K , are subject to the constraint

$$\sum_{k=1}^K x_k = n,$$

and also

$$\sum_{k=1}^K P_k = 1.$$

The mean value, the variances, and the covariances are given by

$$\mathbb{E}[\mathbf{x}] = n\mathbf{P}, \quad \sigma_k^2 = nP_k(1 - P_k), \quad k = 1, 2, \dots, K, \quad \text{cov}(x_i, x_j) = -nP_iP_j, \quad i \neq j. \quad (2.59)$$

2.3.2 CONTINUOUS VARIABLES

The uniform distribution

A random variable x is said to follow a *uniform* distribution in an interval $[a, b]$, and we write $x \sim \mathcal{U}(a, b)$, with $a > -\infty$ and $b < +\infty$, if

$$p(x) = \begin{cases} \frac{1}{b-a}, & \text{if } a \leq x \leq b, \\ 0, & \text{otherwise.} \end{cases} \quad (2.60)$$

Figure 2.3 shows the respective graph. The mean value is equal to

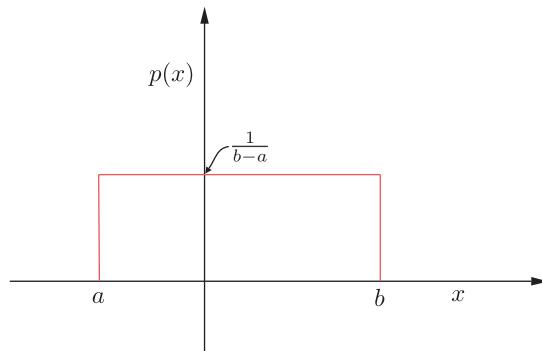
$$\mathbb{E}[x] = \frac{a+b}{2}, \quad (2.61)$$

and the variance is given by (Problem 2.2).

$$\sigma_x^2 = \frac{1}{12}(b-a)^2. \quad (2.62)$$

The Gaussian distribution

The Gaussian or normal distribution is one among the most widely used distributions in all scientific disciplines. We say that a random variable, x , is *Gaussian* or *normal* with parameters μ and σ^2 , and we write $x \sim \mathcal{N}(\mu, \sigma^2)$ or $\mathcal{N}(x|\mu, \sigma^2)$, if

**FIGURE 2.3**

The pdf of a uniform distribution $\mathcal{U}(a, b)$.

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right). \quad (2.63)$$

It can be shown that the corresponding mean and variance are

$$\mathbb{E}[x] = \mu \quad \text{and} \quad \sigma_x^2 = \sigma^2. \quad (2.64)$$

Indeed, by the definition of the mean value, we have that

$$\begin{aligned} \mathbb{E}[x] &= \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} x \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx \\ &= \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} (y + \mu) \exp\left(-\frac{y^2}{2\sigma^2}\right) dy. \end{aligned} \quad (2.65)$$

Due to the symmetry of the exponential function, performing the integration involving y gives zero and the only surviving term is due to μ . Taking into account that a pdf integrates to one, we obtain the result.

To derive the variance, from the definition of the Gaussian pdf, we have that

$$\int_{-\infty}^{+\infty} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \sqrt{2\pi}\sigma. \quad (2.66)$$

Taking the derivative of both sides with respect to σ , we obtain

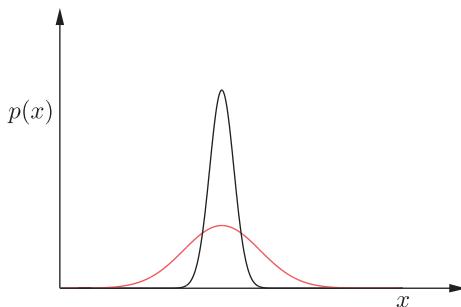
$$\int_{-\infty}^{+\infty} \frac{(x-\mu)^2}{\sigma^3} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \sqrt{2\pi} \quad (2.67)$$

or

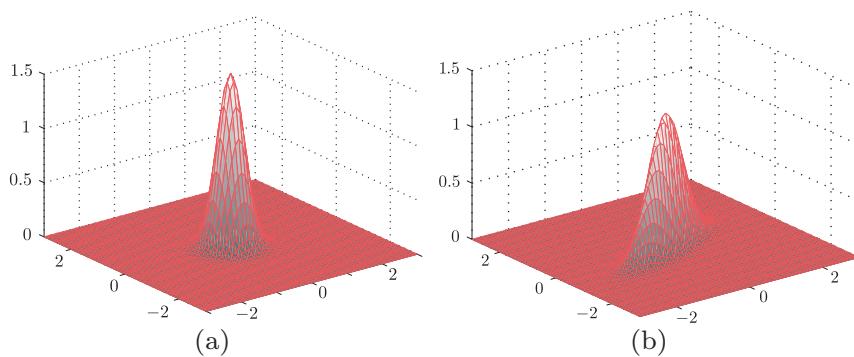
$$\frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{+\infty} (x-\mu)^2 \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) dx = \sigma^2, \quad (2.68)$$

which proves the claim.

Figure 2.4 shows the graph for two cases, $\mathcal{N}(x|1, 0.1)$ and $\mathcal{N}(x|1, 0.01)$. Both curves are symmetrically placed around the mean value $\mu = 1$. Observe that the smaller the variance is, the sharper around

**FIGURE 2.4**

The graphs of two Gaussian pdfs for $\mu = 1$ and $\sigma^2 = 0.1$ (red) and $\sigma^2 = 0.01$ (gray).

**FIGURE 2.5**

The graph of two two-dimensional Gaussian pdfs for $\mu = \mathbf{0}$ and different covariance matrices. (a) The covariance matrix is diagonal with equal elements along the diagonal. (b) The corresponding covariance matrix is nondiagonal.

the mean value the pdf becomes. The generalization of the Gaussian to vector variables, $\mathbf{x} \in \mathbb{R}^l$, results in the so-called *multivariate Gaussian* or *normal* distribution, $\mathbf{x} \sim \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, which is defined as

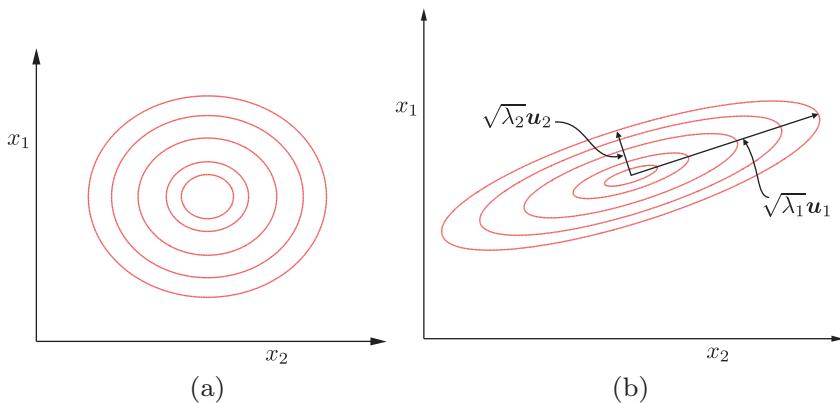
$$p(\mathbf{x}) = \frac{1}{(2\pi)^{l/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right); \quad \text{Gaussian pdf,} \quad (2.69)$$

where $|\cdot|$ denotes the determinant of a matrix. It can be shown ([Problem 2.3](#)) that

$$\mathbb{E}[\mathbf{x}] = \boldsymbol{\mu} \quad \text{and} \quad \text{Cov}(\mathbf{x}) = \boldsymbol{\Sigma}. \quad (2.70)$$

[Figure 2.5](#) shows the two-dimensional normal pdf for two cases. Both share the same mean value, $\boldsymbol{\mu} = \mathbf{0}$, but they have different covariance matrices,

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.1 \end{bmatrix}, \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.1 & 0.01 \\ 0.01 & 0.2 \end{bmatrix}. \quad (2.71)$$

**FIGURE 2.6**

The isovalue contours for the two Gaussians of Figure 2.5. The contours for the Gaussian in Figure 2.5a are circles, while those corresponding to Figure 2.5b are ellipses. The major and minor axes of the ellipse are determined by the eigenvectors/eigenvalues of the respective covariance matrix, and they are proportional to $\sqrt{\lambda_1}c$ and $\sqrt{\lambda_2}c$, respectively. In the figure, they are shown for the case of $c = 1$. For the case of the diagonal matrix, with equal elements along the diagonal, all eigenvalues are equal, and the ellipse becomes a circle.

Figure 2.6 shows the corresponding isovalue contours for equal density values. In Figure 2.6a, the contours are circles, corresponding to the symmetric pdf in Figure 2.5a with covariance matrix Σ_1 . The one shown in Figure 2.6b corresponds to the pdf in Figure 2.5b associated with Σ_2 . Observe that, in general, the isovalue curves are ellipses/hyperellipsoids. They are centered at the mean value, and the orientation of the major axis as well their exact shape is controlled by the eigenstructure of the associated covariance matrix. Indeed, all points $x \in \mathbb{R}^l$, which score the same density value, obey

$$(x - \mu)^T \Sigma^{-1} (x - \mu) = \text{constant} = c. \quad (2.72)$$

We know that the covariance matrix is *symmetric*, $\Sigma = \Sigma^T$. Thus, its eigenvalues are real and the corresponding eigenvectors can be chosen to form an orthonormal basis (Appendix A.2), which leads to its diagonalization,

$$\Sigma = U^T \Lambda U, \quad (2.73)$$

with

$$U := [u_1, \dots, u_l], \quad (2.74)$$

where $u_i, i = 1, 2, \dots, l$, are the orthonormal eigenvectors, and

$$\Lambda := \text{diag}\{\lambda_1, \dots, \lambda_l\} \quad (2.75)$$

are the respective eigenvalues. We assume that Σ is invertible, hence all eigenvalues are positive (being a positive definite it has positive eigenvalues, Appendix A.2). Due to the orthonormality of the eigenvectors, matrix U is orthogonal as expressed in $UU^T = U^T U = I$. Thus, Eq. (2.72) can now be written as

$$\mathbf{y}^T \Lambda^{-1} \mathbf{y} = c, \quad (2.76)$$

where we have used the linear transformation

$$\mathbf{y} := U(\mathbf{x} - \boldsymbol{\mu}), \quad (2.77)$$

which corresponds to a rotation of the axes by U and a translation of the origin to $\boldsymbol{\mu}$. Equation (2.76) can be written as

$$\frac{y_1^2}{\lambda_1} + \cdots + \frac{y_l^2}{\lambda_l} = c, \quad (2.78)$$

where it can be readily observed that it is an equation describing a (hyper)ellipsoid in the \mathbb{R}^l . From Eq. (2.77), it is easily seen that it is centered at $\boldsymbol{\mu}$ and that the major axes of the ellipsoid are parallel to $\mathbf{u}_1, \dots, \mathbf{u}_l$ (plug in place of \mathbf{x} the standard basis vectors, $[1, 0, \dots, 0]^T$, etc.). The size of the respective axes are controlled by the corresponding eigenvalues. This is shown in Figure 2.6b. For the special case of a diagonal covariance with equal elements across the diagonal, all eigenvalues are equal to the value of the common diagonal element and the ellipsoid becomes a (hyper)sphere (circle).

The Gaussian pdf has a number of nice properties, which we are going to discover as we move on in this book. For the time being, note that if the covariance matrix is diagonal,

$$\Sigma = \text{diag}\{\sigma_1^2, \dots, \sigma_l^2\},$$

that is, when the covariance of all the elements $\text{cov}(x_i, x_j) = 0, i, j = 1, 2, \dots, l$, then the random variables comprising \mathbf{x} are statistically *independent*. In general, this is not true. Uncorrelated variables are not necessarily independent; independence is a much stronger condition. This is true, however, if they follow a multivariate Gaussian. Indeed, if the covariance matrix is diagonal, then the multivariate Gaussian is written as

$$p(\mathbf{x}) = \prod_{i=1}^l \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - \mu_i)^2}{2\sigma_i^2}\right). \quad (2.79)$$

In other words,

$$p(\mathbf{x}) = \prod_{i=1}^l p(x_i), \quad (2.80)$$

which is the condition for statistical independence.

The central limit theorem

This is one of the most fundamental theorems in probability theory and statistics and it partly explains the popularity of the Gaussian distribution. Consider N mutually *independent* random variables, each following its own distribution with mean values μ_i and variances $\sigma_i^2, i = 1, 2, \dots, N$. Define a new random variable as their sum,

$$\mathbf{x} = \sum_{i=1}^N \mathbf{x}_i. \quad (2.81)$$

Then the mean and variance of the new variable are given by

$$\mu = \sum_{i=1}^N \mu_i, \quad \text{and} \quad \sigma^2 = \sum_{i=1}^N \sigma_i^2. \quad (2.82)$$

It can be shown (e.g., [4, 6]) that as $N \rightarrow \infty$ the distribution of the normalized variable

$$z = \frac{x - \mu}{\sigma} \quad (2.83)$$

tends to the *standard normal distribution*, and for the corresponding pdf we have

$$p(z) \xrightarrow[N \rightarrow \infty]{} \mathcal{N}(z|0, 1). \quad (2.84)$$

In practice, even summing up a relatively small number, N , of random variables, one can obtain a good approximation to a Gaussian. For example, if the individual pdfs are smooth enough and each random variable is *independent and identically distributed* (i.i.d.), a number N between 5 and 10 can be sufficient.

The exponential distribution

We say that a random variable follows an exponential distribution with parameter $\lambda > 0$, if

$$p(x) = \begin{cases} \lambda \exp(-\lambda x), & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \quad (2.85)$$

The distribution has been used, for example, to model the time between arrivals of telephone calls or of a bus at a bus stop. The mean and variance can be easily computed by following simple integration rules, and they are

$$\mathbb{E}[x] = \frac{1}{\lambda}, \quad \sigma_x^2 = \frac{1}{\lambda^2}. \quad (2.86)$$

The beta distribution

We say that a random variable, $x \in [0, 1]$, follows a beta distribution with positive parameters, a, b , and we write, $x \sim \text{Beta}(x|a, b)$, if

$$p(x) = \begin{cases} \frac{1}{B(a, b)} x^{a-1} (1-x)^{b-1}, & \text{if } 0 \leq x \leq 1, \\ 0, & \text{otherwise,} \end{cases} \quad (2.87)$$

where $B(a, b)$ is the beta function, defined as

$$B(a, b) := \int_0^1 x^{a-1} (1-x)^{b-1} dx. \quad (2.88)$$

The mean and variance of the beta distribution are given by (Problem 2.4)

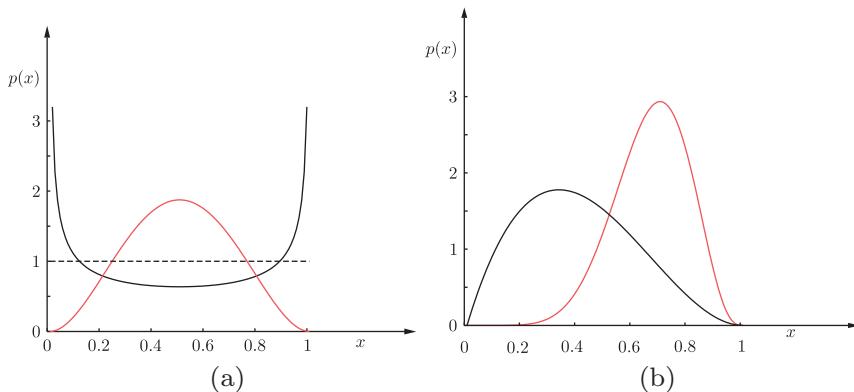
$$\mathbb{E}[x] = \frac{a}{a+b}, \quad \sigma_x^2 = \frac{ab}{(a+b)^2(a+b+1)}. \quad (2.89)$$

Moreover, it can be shown (Problem 2.5) that

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}, \quad (2.90)$$

where $\Gamma(\cdot)$ is the gamma functions defined as

$$\Gamma(a) = \int_0^\infty x^{a-1} e^{-x} dx. \quad (2.91)$$

**FIGURE 2.7**

The graphs of the pdfs of the Beta distribution for different values of the parameters. (a) The dotted line corresponds to $a = 1, b = 1$, the gray line to $a = 0.5, b = 0.5$, and the red one to $a = 3, b = 3$. (b) The gray line corresponds to $a = 2, b = 3$, and the red one to $a = 8, b = 4$. For values $a = b$, the shape is symmetric around $1/2$. For $a < 1, b < 1$, it is convex. For $a > 1, b > 1$, it is zero at $x = 0$ and $x = 1$. For $a = 1 = b$, it becomes the uniform distribution. If $a < 1$, $p(x) \rightarrow \infty$, $x \rightarrow 0$ and if $b < 1$, $p(x) \rightarrow \infty$, $x \rightarrow 1$.

The beta distribution is very flexible and one can achieve various shapes by changing the parameters a, b . For example, if $a = b = 1$, the uniform distribution results. If $a = b$, the pdf has a symmetric graph around $1/2$. If $a > 1, b > 1$ then $p(x) \rightarrow 0$ both at $x = 0$ and $x = 1$. If $a < 1$ and $b < 1$, it is convex with a unique minimum. If $a < 1$, it tends to ∞ as $x \rightarrow 0$, and if $b < 1$, it tends to ∞ for $x \rightarrow 1$. Figures 2.7a and b show the graph of the beta distribution for different values of the parameters.

The gamma distribution

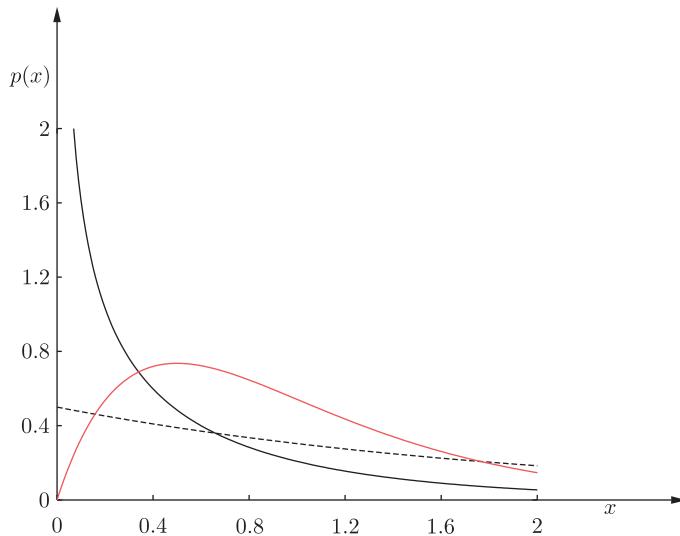
A random variable follows the gamma distribution with positive parameters a, b , and we write $x \sim \text{Gamma}(x|a, b)$ if

$$p(x) = \begin{cases} \frac{b^a}{\Gamma(a)} x^{a-1} e^{-bx}, & x > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (2.92)$$

The mean and variance are given by

$$\mathbb{E}[x] = \frac{a}{b}, \quad \sigma_x^2 = \frac{a}{b^2}. \quad (2.93)$$

The gamma distribution also takes various shapes by varying the parameters. For $a < 1$, it is strictly decreasing and $p(x) \rightarrow \infty$ as $x \rightarrow 0$ and $p(x) \rightarrow 0$ as $x \rightarrow \infty$. Figure 2.8 shows the resulting graphs for various values of the parameters.

**FIGURE 2.8**

The pdf of the gamma distribution takes different shapes for the various values of the parameters: $a = 0.5, b = 1$ (full line gray), $a = 2, b = 0.5$ (red), $a = 1, b = 2$ (dotted).

Remarks 2.1.

- Setting in the gamma distribution a to be an integer (usually $a = 2$), the *Erlang* distribution results. This distribution is being used to model waiting times in queueing systems.
- The *chi-squared* is also a special case of the gamma distribution, and it is obtained if we set $b = 1/2$ and $a = \nu/2$. The chi-squared distribution results if we sum up ν squared normal variables.

The Dirichlet distribution

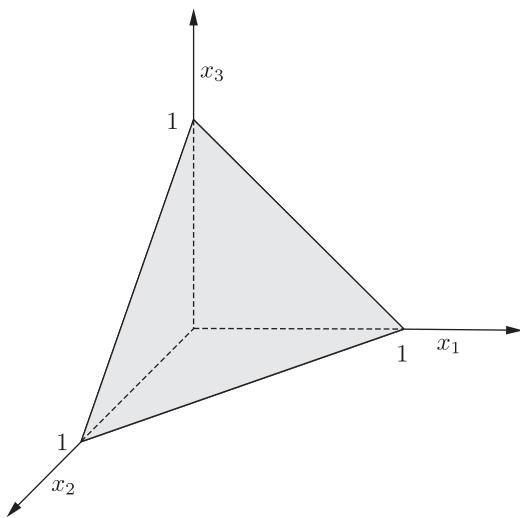
The Dirichlet distribution can be considered as the multivariate generalization of the beta distribution. Let $\mathbf{x} = [x_1, \dots, x_K]^T$ be a random vector, with components such as

$$0 \leq x_k \leq 1, \quad k = 1, 2, \dots, K, \quad \text{and} \quad \sum_{k=1}^K x_k = 1. \quad (2.94)$$

In other words, the random variables lie on $(K - 1)$ -dimensional *simplex*, Figure 2.9. We say that the random vector \mathbf{x} follows a Dirichlet distribution with parameters $\boldsymbol{a} = [a_1, \dots, a_K]^T$, and we write $\mathbf{x} \sim \text{Dir}(\mathbf{x}|\boldsymbol{a})$, if

$$p(\mathbf{x}) = \text{Dir}(\mathbf{x}|\boldsymbol{a}) := \frac{\Gamma(\bar{a})}{\Gamma(a_1) \dots \Gamma(a_K)} \prod_{k=1}^K x_k^{a_k-1},$$

(2.95)

**FIGURE 2.9**

The 2-dimensional simplex in \mathbb{R}^3 .

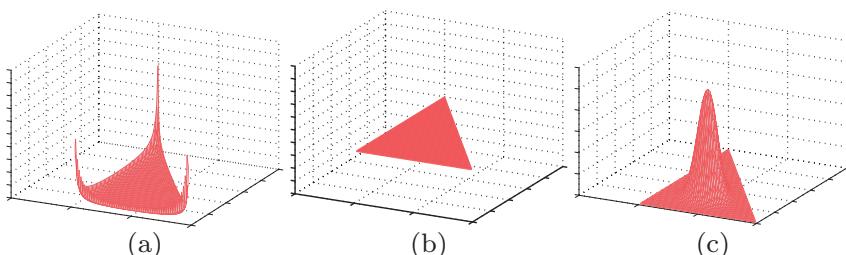
where

$$\bar{a} = \sum_{k=1}^K a_k. \quad (2.96)$$

The mean, variance, and covariances of the involved random variables are given by (Problem 2.7),

$$\mathbb{E}[\mathbf{x}] = \frac{1}{\bar{a}} \mathbf{a}, \quad \sigma_k^2 = \frac{a_k(\bar{a} - a_k)}{\bar{a}^2(\bar{a} + 1)}, \quad \text{cov}(\mathbf{x}_i, \mathbf{x}_j) = -\frac{a_i a_j}{\bar{a}^2(\bar{a} + 1)}. \quad (2.97)$$

Figure 2.10 shows the graph of the Dirichlet distribution for different values of the parameters, over the respective 2D-simplex.

**FIGURE 2.10**

The Dirichlet distribution over the 2D-simplex for (a) (0.1,0.1,0.1), (b) (1,1,1), and (c) (10,10,10).

2.4 STOCHASTIC PROCESSES

The notion of a random variable has been introduced to describe the result of a random experiment whose outcome is a single value, as occurs, heads or tails in a coin-tossing experiment, or a value between one and six when throwing the die in a backgammon game.

In this section, the notion of a *stochastic process* is introduced to describe random experiments where the outcome of each experiment is a function or a sequence; in other words, the outcome of each experiment is an infinite number of values. In this book, we are only going to be concerned with stochastic processes associated with sequences. Thus, the result of a random experiment is a *sequence*, u_n (or sometimes denoted as $u(n)$), $n \in \mathbb{Z}$, where \mathbb{Z} is the set of integers. Usually, n is interpreted as a time index, and u_n is called a *time series*, or in signal processing jargon, a *discrete-time signal*. In contrast, if the outcome is a function, $u(t)$, it is called a *continuous-time signal*. We are going to adopt the time interpretation of the free variable, n , for the rest of the chapter, without harming generality.

When discussing random variables, we used the notation x to denote the random variable, which assumes a value, x , from the sample space once an experiment is performed. Similarly, we are going to use u_n to denote the specific sequence resulting from a single experiment and the roman font, u_n , to denote the corresponding *discrete-time* random process, that is, the rule that assigns a specific sequence as the outcome of an experiment. A stochastic process can be considered as a family or *ensemble* of sequences. The individual sequences are known as *sample sequences* or simply as *realizations*.

For our notational convention, in general, we are going to reserve different symbols for processes and random variables. We have already used the symbol u and not x ; this is only for pedagogical reasons, just to make sure that the reader readily recognizes when the focus is on random variables and when it is on random processes. In signal processing jargon, a stochastic process is also known as a *random signal*. [Figure 2.11](#) illustrates the fact that the outcome of an experiment involving a stochastic process is a sequence of values.

Note that fixing the time to a specific value, $n = n_0$, makes u_{n_0} a random variable. Indeed, for each random experiment we perform, a single value results at time instant n_0 . From this perspective, a random process can be considered the collection of infinite random variables, $\{u_n, n \in \mathbb{Z}\}$. So, is there a need to study a stochastic process separate from random variables/vectors? The answer is yes, and the reason is that we are going to allow certain time dependencies among the random variables, corresponding to different time instants, and study the respective effect on the time evolution of the random process. Stochastic processes will be considered in [Chapter 5](#), where the underlying time dependencies will be exploited for computational simplifications, and in [Chapter 13](#) in the context of Gaussian processes.

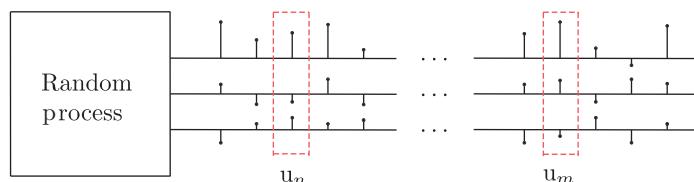


FIGURE 2.11

The outcome of each experiment, associated with a *discrete-time* stochastic process, is a *sequence* of values. For each one of the realizations, the corresponding values obtained at any instant (e.g., n or m) comprise the outcomes of a corresponding random variable, u_n or u_m , respectively.

2.4.1 FIRST AND SECOND ORDER STATISTICS

For a stochastic process to be fully described, one must know the joint pdfs (pmfs for discrete-valued random variables)

$$p(u_n, u_m, \dots, u_r; n, m, \dots, r), \quad (2.98)$$

for all possible combinations of random variables, u_n, u_m, \dots, u_r . Note that, in order to emphasize it, we have explicitly denoted the dependence of the joint pdfs on the involved time instants. However, from now on, this will be suppressed for notational convenience. Most often, in practice, and certainly in this book, the emphasis is on computing first and second order statistics only, based on $p(u_n)$ and $p(u_n, u_m)$. To this end, the following quantities are of particular interest:

Mean at Time n:

$$\mu_n := \mathbb{E}[u_n] = \int_{-\infty}^{+\infty} u_n p(u_n) du_n. \quad (2.99)$$

Autocovariance at Time Instants, n, m:

$$\text{cov}(n, m) := \mathbb{E}[(u_n - \mathbb{E}[u_n])(u_m - \mathbb{E}[u_m])]. \quad (2.100)$$

Autocorrelation at Time Instants, n, m:

$$r(n, m) := \mathbb{E}[u_n u_m]. \quad (2.101)$$

We refer to these mean values as *ensemble* averages to stress that they convey statistical information over the ensemble of sequences that comprise the process.

The respective definitions for complex stochastic processes are

$$\text{cov}(n, m) = \mathbb{E}[(u_n - \mathbb{E}[u_n])(u_m - \mathbb{E}[u_m])^*], \quad (2.102)$$

and

$$r(n, m) = \mathbb{E}[u_n u_m^*]. \quad (2.103)$$

2.4.2 STATIONARITY AND ERGODICITY

Definition 2.1 (*Strict-Sense Stationarity*). A stochastic process, u_n , is said to be *strict-sense stationary* (SSS) if its statistical properties are invariant to a shift of the origin, or if $\forall k \in \mathbb{Z}$

$$p(u_n, u_m, \dots, u_r) = p(u_{n-k}, u_{m-k}, \dots, u_{r-k}), \quad (2.104)$$

and for any possible combination of time instants, $n, m, \dots, r \in \mathbb{Z}$.

In other words, the stochastic processes u_n and u_{n-k} are described by the same joint pdfs of all orders. A weaker version of stationarity is that of the *mth order stationarity*, where joint pdfs involving up to m variables are invariant to the choice of the origin. For example, for a second order ($m = 2$) stationary process, we have that $p(u_n) = p(u_{n-k})$ and $p(u_n, u_r) = p(u_{n-k}, u_{r-k})$, $\forall n, r, k \in \mathbb{Z}$.

Definition 2.2 (*Wide-Sense Stationarity*). A stochastic process, u_n , is said to be *wide-sense stationary* (WSS) if the mean value is constant over all time instants and the autocorrelation/autocovariance sequences depend on the difference of the involved time indices, or

$$\mu_n = \mu, \quad \text{and} \quad r(n, n - k) = r(k). \quad (2.105)$$

Note that WSS is a weaker version of the second order stationarity; in the latter case, all possible second order statistics are independent of the time origin. In the former, we only require the autocorrelation (autocovariance) and the mean value to be independent of the time origin. The reason we focus on these two quantities (statistics) is that they are of major importance in the study of linear systems and in the mean-square estimation, as we will see in [Chapter 4](#).

Obviously, a strict-sense stationary process is also wide-sense stationary but, in general, not the other way around. For wide-sense stationary processes, the autocorrelation becomes a *sequence* with a *single* time index as the free parameter; thus its value, which measures a relation of the variables at two time instants, depends *solely on how much these time instants differ*, and not on their specific values.

From our basic statistics course, we know that given a random variable, x , its mean value can be approximated by the sample mean. Carrying out N successive independent experiments, let $x_n, n = 1, 2, \dots, N$, be the obtained values, known as *observations*. The *sample mean* is defined as

$$\hat{\mu}_N := \frac{1}{N} \sum_{n=1}^N x_n. \quad (2.106)$$

For large enough values of N , we expect the sample mean to be close to the true mean value, $\mathbb{E}[x]$. In a more formal way, this is guaranteed by the fact that $\hat{\mu}_N$ is associated with an *unbiased* and *consistent* estimator. We will discuss such issues in [Chapter 3](#); however, we can refresh our memory at this point. Every time we repeat the N random experiments, different samples result and hence a different estimate $\hat{\mu}_N$ is computed. Thus, the values of the estimates define a new random variable, $\hat{\mu}_n$, known as the estimator. This is unbiased, because it can easily be shown that

$$\mathbb{E}[\hat{\mu}_N] = \mathbb{E}[x], \quad (2.107)$$

and it is consistent because its variance tends to zero as $N \rightarrow +\infty$ ([Problem 2.8](#)). These two properties guarantee that, with high probability, for large values of N , $\hat{\mu}_N$ will be close to the true mean value.

To apply the concept of sample mean approximation to random processes, one must have at her/his disposal a number of N realizations, and compute the sample mean at different time instants “across the process,” using *different* realizations, representing the ensemble of sequences. Similarly, sample mean arguments can be used to approximate the autocovariance/autocorrelation sequences. However, this is a costly operation, since now each experiment results in an infinite number of values (a sequence of values). Moreover, it is common in practical applications that only one realization is available to the user.

To this end, we will now define a special type of stochastic processes, where the sample mean operation can be significantly simplified.

Definition 2.3 (Ergodicity). A stochastic process is said to be *ergodic* if the complete statistics can be determined by any one of the realizations.

In other words, if a process is ergodic, every single realization carries identical statistical information and it can describe the entire random process. Since from a single sequence only one set of pdfs can be obtained, we conclude that *every ergodic process is necessarily stationary*. A nonstationary process has infinite sets of pdfs, depending upon the choice of the origin. For example, there is only one mean value that can result from a single realization and be obtained as a (time) average, over the values of the sequence. Hence, the mean value of a stochastic process that is ergodic must be constant for all time instants, or independent of the time origin. The same is true for all higher order statistics.

A special type of ergodicity is that of the *second order* ergodicity. This means that only statistics up to a second order can be obtained from a single realization. Second order ergodic processes are

necessarily wide-sense stationary. For second order ergodic processes, the following are true:

$$\mathbb{E}[u_n] = \mu = \lim_{N \rightarrow \infty} \hat{\mu}_N, \quad (2.108)$$

where

$$\hat{\mu}_N := \frac{1}{2N+1} \sum_{n=-N}^N u_n.$$

Also,

$$\text{cov}(k) = \lim_{N \rightarrow \infty} \frac{1}{2N+1} \sum_{n=-N}^N (u_n - \mu)(u_{n+k} - \mu), \quad (2.109)$$

where both limits are in the mean-square sense; that is,

$$\lim_{N \rightarrow \infty} \mathbb{E}[(\hat{\mu}_N - \mu)^2] = 0,$$

and similarly for the autocovariance. Note that often, ergodicity is only required to be assumed for the computation of the mean and covariance and not for all possible second order statistics. In this case, we talk about *mean-ergodic* and *covariance-ergodic* processes.

In summary, when ergodic processes are involved, ensemble averages “across the process” can be obtained as time averages “along the process”; see Figure 2.12.

In practice, when only a finite number of samples from a realization is available, then the mean and covariance are approximated as the respective sample means.

An issue is to establish conditions under which a process is mean-ergodic or covariance-ergodic. Such conditions do exist, and the interested reader can find such information in more specialized books [6]. It turns out that the condition for mean-ergodicity relies on second order statistics and the condition for covariance-ergodicity on fourth order statistics.

It is very common in statistics as well as in machine learning and signal processing to subtract the mean value from the data during the *preprocessing stage*. In such a case, we say that the data are *centered*. The resulting new process has now *zero mean* value, and the covariance and autocorrelation sequences coincide. From now on, we will assume that the mean is known (or computed as a sample mean) and then subtracted. Such a treatment simplifies the analysis without harming generality.

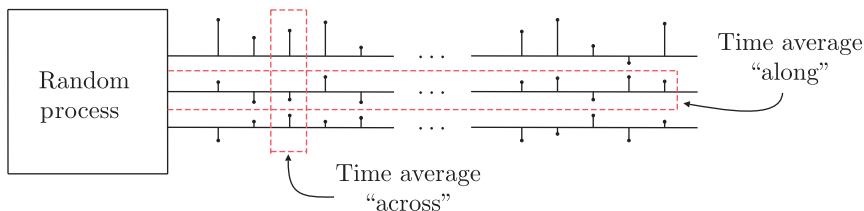


FIGURE 2.12

For ergodic processes, mean values for each time instant (time averaging “across” the process) are computed as time averages “along” the process.

Example 2.2. The goal of this example is to construct a process that is WSS yet not ergodic. Let a WSS process, u_n ,

$$\mathbb{E}[u_n] = \mu,$$

and

$$\mathbb{E}[u_n u_{n-k}] = r_u(k).$$

Define the process,

$$v_n := au_n, \quad (2.110)$$

where a is a random variable taking values in $\{0, 1\}$, with probabilities $P(0) = P(1) = 0.5$. Moreover, a and u_n are statistically independent. Then, we have that

$$\mathbb{E}[v_n] = \mathbb{E}[au_n] = \mathbb{E}[a]\mathbb{E}[u_n] = 0.5\mu, \quad (2.111)$$

and

$$\mathbb{E}[v_n v_{n-k}] = \mathbb{E}[a^2] \mathbb{E}[u_n u_{n-k}] = 0.5r_u(k). \quad (2.112)$$

Thus, v_n is WSS. However, it is not covariance-ergodic. Indeed, some of the realizations will be equal to zero (when $a = 0$), and the mean value and autocorrelation, which will result from them as time averages, will be zero, which is different from the ensemble averages.

2.4.3 POWER SPECTRAL DENSITY

The Fourier transform is an indispensable tool for representing in a compact way, in the frequency domain, the variations that a function/sequence undergoes in terms of its free variable (e.g., time). Stochastic processes are inherently related to time. The question that is now raised is whether stochastic processes can be described in terms of a Fourier transform. The answer is affirmative, and the vehicle to achieve this is via the autocorrelation sequence for processes that are at least wide-sense stationary. Prior to providing the necessary definitions, it is useful to summarize some common properties of the autocorrelation sequence.

Properties of the autocorrelation sequence

Let u_n be a wide-sense stationary process. Its autocorrelation sequence has the following properties, which are given for the more general complex-valued case:

- *Property I.*

$$r(k) = r^*(-k), \quad \forall k \in \mathbb{Z}. \quad (2.113)$$

This property is a direct consequence of the invariance with respect to the choice of the origin. Indeed,

$$r(k) = \mathbb{E}[u_n u_{n-k}^*] = \mathbb{E}[u_{n+k} u_n^*] = r^*(-k).$$

- *Property II.*

$$r(0) = \mathbb{E}[|u_n|^2]. \quad (2.114)$$

That is, the value of the autocorrelation at $k = 0$ is equal to the mean-square of the magnitude of the respective random variables. Interpreting the square of the magnitude of a variable as its energy, $r(0)$ can be interpreted as the corresponding (average) power.

- *Property III.*

$$r(0) \geq |r(k)|, \quad \forall k \neq 0. \quad (2.115)$$

The proof is provided in [Problem 2.9](#). In other words, the correlation of the variables, corresponding to two different time instants, cannot be larger (in magnitude) than $r(0)$. As we will see in [Chapter 4](#), this property is essentially the Cauchy-Schwartz inequality for the inner products (see also Appendix of [Chapter 8](#)).

- *Property IV.* The autocorrelation sequence of a stochastic process is *positive definite*. That is,

$$\sum_{n=1}^N \sum_{m=1}^N a_n a_m^* r(n, m) \geq 0, \quad \forall a_n \in \mathbb{C}, n = 1, 2, \dots, N, \forall N \in \mathbb{Z}. \quad (2.116)$$

Proof. The proof is easily obtained by the definition of the autocorrelation,

$$0 \leq \mathbb{E} \left[\left| \sum_{n=1}^N a_n u_n \right|^2 \right] = \sum_{n=1}^N \sum_{m=1}^N a_n a_m^* \mathbb{E}[u_n u_m], \quad (2.117)$$

which proves the claim. Note that strictly speaking, we should say that it is semipositive definite. However, the “positive definite” name is the one that has survived in the literature. This property will be useful when introducing *Gaussian processes* in [Chapter 13](#). \square

- *Property V.* Let u_n and v_n be two WSS processes. Define the new process

$$z_n = u_n + v_n.$$

Then,

$$r_z(k) = r_u(k) + r_v(k) + r_{uv}(k) + r_{vu}(k), \quad (2.118)$$

where the *cross-correlation* between two jointly WS stationary stochastic processes is defined as

$$r_{uv}(k) := \mathbb{E}[u_n v_{n-k}^*], \quad k \in \mathbb{Z} : \quad \text{Cross-correlation.} \quad (2.119)$$

The proof is a direct consequence of the definition. Note that if the two processes are *uncorrelated*, as when $r_{uv}(k) = r_{vu}(k) = 0$, then

$$r_z(k) = r_u(k) + r_v(k).$$

Obviously, this is also true if the processes u_n and v_n are independent and of zero mean value, since then $\mathbb{E}[u_n v_{n-k}^*] = \mathbb{E}[u_n] \mathbb{E}[v_{n-k}^*] = 0$. It should be stressed here that uncorrelatedness is a weaker condition and it does not necessarily imply independence; the opposite is true for zero mean values.

- *Property VI.*

$$r_{uv}(k) = r_{vu}^*(-k) \quad (2.120)$$

The proof is similar to that of Property I.

- *Property VII.*

$$r_u(0)r_v(0) \geq |r_{uv}(k)|, \quad \forall k \in \mathbb{Z}. \quad (2.121)$$

The proof is also given in [Problem 2.9](#).

Power spectral density

Definition 2.4. Given a WSS stochastic process, u_n , its *power spectral density* (PSD) (or simply the *power spectrum*) is defined as the Fourier transform of its autocorrelation sequence,

$$S(\omega) := \sum_{k=-\infty}^{\infty} r(k) \exp(-j\omega k) : \text{ Power Spectral Density.}$$

(2.122)

Using the Fourier transform properties, we can recover the autocorrelation sequence via the *inverse Fourier transform*, in the following manner:

$$r(k) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S(\omega) \exp(j\omega k) d\omega.$$

(2.123)

Due to the properties of the autocorrelation sequence, the PSD have some interesting and useful properties, from a practical point of view.

Properties of the PSD

- The PSD of a WSS stochastic process is a *real* and *nonnegative* function of ω . Indeed, we have that

$$\begin{aligned} S(\omega) &= \sum_{k=-\infty}^{+\infty} r(k) \exp(-j\omega k) \\ &= r(0) + \sum_{k=-\infty}^{-1} r(k) \exp(-j\omega k) + \sum_{k=1}^{\infty} r(k) \exp(-j\omega k) \\ &= r(0) + \sum_{k=1}^{+\infty} r^*(k) \exp(j\omega k) + \sum_{k=1}^{\infty} r(k) \exp(-j\omega k) \\ &= r(0) + 2 \sum_{k=1}^{+\infty} \text{Real}(r(k) \exp(-j\omega k)), \end{aligned} \quad (2.124)$$

which proves the claim that PSD is a real number. In the proof, Property I of the autocorrelation sequence has been used. We defer the proof for the nonnegative part to the end of this section.

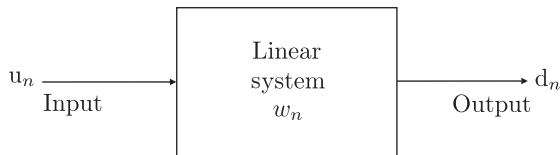
- The area under the graph of $S(\omega)$ is proportional to the power of the stochastic process, as expressed by

$$\mathbb{E}[|u_n|^2] = r(0) = \frac{1}{2\pi} \int_{-\pi}^{+\pi} S(\omega) d\omega, \quad (2.125)$$

which is obtained from Eq. (2.123) if we set $k = 0$. We will come to the physical meaning of this property very soon.

Transmission through a linear system

One of the most important tasks in signal processing and systems theory is the *linear filtering operation* on an *input* time series (*signal*) to generate another *output* sequence. The block diagram of the filtering

**FIGURE 2.13**

The linear system (filter) is excited by the input sequence (signal), u_n , and provides the output sequence (signal), d_n .

operation is shown in [Figure 2.13](#). From the linear system theory and signal processing basics, it is established that for a class of linear systems known as *linear time invariant* (LTI), the input-output relation is given via the elegant *convolution* between the input sequence and the *impulse response* of the filter,

$$d_n = w_n * u_n := \sum_{i=-\infty}^{+\infty} w_i^* u_{n-i} : \text{ Convolution Sum,} \quad (2.126)$$

where $\dots, w_0, w_1, w_2, \dots$ are the parameters comprising the impulse response describing the filter [8]. In case the impulse response is of finite duration, for example, w_0, w_1, \dots, w_{l-1} , and the rest of the values are zero, then the convolution can be written as

$$d_n = \sum_{i=0}^{l-1} w_i^* u_{n-i} = \mathbf{w}^H \mathbf{u}_n, \quad (2.127)$$

where

$$\mathbf{w} := [w_0, w_1, \dots, w_{l-1}]^T, \quad (2.128)$$

and

$$\mathbf{u}_n := [u_n, u_{n-1}, \dots, u_{n-l+1}]^T \in \mathbb{R}^l. \quad (2.129)$$

The latter is known as the *input vector* of order l and at time n . It is interesting to note that this is a random vector. However, its elements are part of the stochastic process at *successive* time instants. This gives the respective autocorrelation matrix certain properties and a rich structure, which will be studied and exploited in [Chapter 4](#). As a matter of fact, this is the reason that we used different symbols to denote processes and general random vectors; thus, the reader can readily remember that when dealing with a process, the elements of the involved random vectors have this *extra structure*. Moreover, observe from Eq. (2.126) that if the impulse response of the system is zero for negative values of the time index, n , this guarantees *causality*. That is, the output depends only on the values of the input at the current and previous time instants only, and there is no dependence on future values. As a matter of fact, this is also a necessary condition for causality; that is, if the system is causal, then its impulse response is zero for negative time instants [8].

Theorem 2.1. *The power spectral density of the output, d_n , of a linear time invariant system, when it is excited by a WSS stochastic process, u_n , is given by*

$$S_d(\omega) = |W(\omega)|^2 S_u(\omega), \quad (2.130)$$

where

$$W(\omega) := \sum_{n=-\infty}^{+\infty} w_n \exp(-j\omega n). \quad (2.131)$$

Proof. First, it is shown (Problem 2.10) that

$$r_d(k) = r_u(k) * w_k * w_{-k}^*. \quad (2.132)$$

Then, taking the Fourier transform of both sides, we obtain Eq. (2.130). To this end, we used the well-known properties of the Fourier transform,

$$r_u(k) * w_k \mapsto S_u(\omega)W(\omega), \quad \text{and} \quad w_{-k}^* \mapsto W^*(\omega).$$

□

Physical interpretation of the PSD

We are now ready to justify why the Fourier transform of the autocorrelation sequence was given the specific name of “power spectral density.” We restrict our discussion to real processes, although similar arguments hold true for the more general complex case. Figure 2.14 shows the magnitude of the Fourier transform of the impulse response of a very special linear system. The Fourier transform is unity for any frequency in the range $|\omega - \omega_0| \leq \frac{\Delta\omega}{2}$ and zero otherwise. Such a system is known as *bandpass filter*. We assume that $\Delta\omega$ is very small. Then, using Eq. (2.130) and assuming that within the intervals $|\omega - \omega_0| \leq \frac{\Delta\omega}{2}$, $S_u(\omega) \approx S_u(\omega_0)$, we have that

$$S_d(\omega) = \begin{cases} S_u(\omega_0), & \text{if } |\omega - \omega_0| \leq \frac{\Delta\omega}{2}, \\ 0, & \text{otherwise.} \end{cases} \quad (2.133)$$

Hence,

$$\Delta P := \mathbb{E}[|d_n|^2] = r_d(0) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} S_d(\omega) d\omega \approx S_u(\omega_0) \frac{\Delta\omega}{\pi}, \quad (2.134)$$

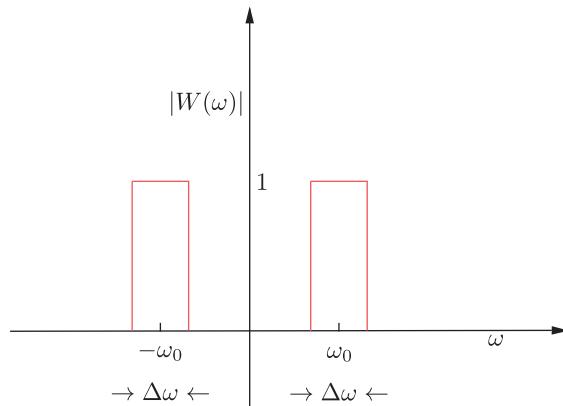


FIGURE 2.14

An ideal bandpass filter. The output contains frequencies only in the range of $|\omega - \omega_0| < \Delta\omega/2$.

due to the symmetry of the power spectral density ($S_u(\omega) = S_u(-\omega)$). Hence,

$$\frac{1}{\pi} S_u(\omega_o) = \frac{\Delta P}{\Delta \omega}. \quad (2.135)$$

In other words, the value $S_u(\omega_o)$ can be interpreted as the power density (power per frequency interval) in the frequency (spectrum) domain.

Moreover, this also establishes what was said before that the PSD is a *nonnegative* real function for any value of $\omega \in [-\pi, +\pi]$ (The PSD, being the Fourier transform of a sequence, is periodic with period 2π , e.g., [8]).

Remarks 2.2.

- Note that for any WSS stochastic process, there is only one autocorrelation sequence that describes it. However, the converse is not true. A single autocorrelation sequence can correspond to more than one WSS process. Recall that the autocorrelation is the mean value of the product of random variables. However, many random variables can have the same mean value.
- We have shown that the Fourier transform, $S(\omega)$, of an autocorrelation sequence, $r(k)$, is non-negative. Moreover, if a sequence, $r(k)$, has a nonnegative Fourier transform, then it is positive definite and we can *always* construct a WSS process that has $r(k)$ as its autocorrelation sequence (e.g., [6, pages 410,421]). Thus, the *necessary and sufficient condition for a sequence to be an autocorrelation sequence is the nonnegativity of its Fourier transform*.

Example 2.3. *White Noise Sequence.*

A stochastic process, η_n , is said to be *white noise* if the mean and its autocorrelation sequence satisfy

$$\mathbb{E}[\eta_n] = 0 \quad \text{and} \quad r(k) = \begin{cases} \sigma_\eta^2, & \text{if } k = 0, \\ 0, & \text{if } k \neq 0. \end{cases} : \quad \text{White Noise,} \quad (2.136)$$

where σ_η^2 is its variance. In other words, all variables at different time instants are uncorrelated. If, in addition, they are independent, we say that it is *strictly white noise*. It is readily seen that its PSD is given by

$$S_\eta(\omega) = \sigma_\eta^2. \quad (2.137)$$

That is, it is constant, and this is the reason it is called white noise, analogous to the white light whose spectrum is equally spread over all the wavelengths.

2.4.4 AUTOREGRESSIVE MODELS

We have just seen an example of a stochastic process, namely white noise. We now turn our attention to generating WSS processes via appropriate modeling. In this way, we will introduce controlled correlation among the variables, corresponding to the various time instants. We focus on the real data case, to simplify the discussion.

Autoregressive processes are among the most popular and widely used models. An autoregressive process of order l , denoted as AR(l), is defined via the following *difference equation*,

$$u_n + a_1 u_{n-1} + \cdots + a_l u_{n-l} = \eta_n : \text{ Autoregressive Process,} \quad (2.138)$$

where η_n is a white noise process with variance σ_η^2 .

As is always the case with any difference equation, one starts from some initial conditions and then generates samples recursively by plugging into the model the input sequence samples. The input samples here correspond to a white noise sequence and the initial conditions are set equal to zero, $u_{-1} = \dots = u_{-l} = 0$.

There is no need to mobilize mathematics to see that such a process is not stationary. Indeed, time instant $n = 0$ is distinctly different from all the rest, since it is the time in which initial conditions are applied. However, the effects of the initial conditions tend asymptotically to zero if all the roots of the corresponding characteristic polynomial,

$$z^l + a_1 z^{l-1} + \cdots + a_l = 0,$$

have magnitude *less than unity* (the solution of the corresponding homogeneous equation, without input, tends to zero) [7]. Then, it can be shown that asymptotically, the AR(l) becomes WSS. This is the assumption that is usually adopted in practice, which will be the case for the rest of this section. Note that the mean value of the process is zero (try it).

The goal now becomes to compute the corresponding autocorrelation sequence, $r(k), k \in \mathbb{Z}$. Multiplying both sides in Eq. (2.138) with $u_{n-k}, k > 0$, and taking the expectation, we obtain

$$\sum_{i=0}^l a_i \mathbb{E}[u_{n-i} u_{n-k}] = \mathbb{E}[\eta_n u_{n-k}], \quad k > 0,$$

where $a_0 := 1$, or

$$\sum_{i=0}^l a_i r(k-i) = 0. \quad (2.139)$$

We have used the fact that $\mathbb{E}[\eta_n u_{n-k}], k > 0$ is zero. Indeed, u_{n-k} depends recursively on $\eta_{n-k}, \eta_{n-k-1}, \dots$, which are all uncorrelated to η_n , since this is a white noise process. Note that Eq. (2.139) is a difference equation, which can be solved provided we have the initial conditions. To this end, multiply Eq. (2.138) by u_n and take expectations, which results in

$$\sum_{i=0}^l a_i r(i) = \sigma_\eta^2, \quad (2.140)$$

since u_n recursively depends on η_n , which contributes the σ_η^2 term, and η_{n-1}, \dots , which result to zeros. Combining Eqs. (2.140) with (2.139) the following *linear* system of equations results

$$\begin{bmatrix} r(0) & r(1) & \dots & r(l) \\ r(1) & r(0) & \dots & r(l-1) \\ \vdots & \vdots & \vdots & \vdots \\ r(l) & r(l-1) & \dots & r(0) \end{bmatrix} \begin{bmatrix} 1 \\ a_1 \\ \vdots \\ a_l \end{bmatrix} = \begin{bmatrix} \sigma_\eta^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \quad (2.141)$$

These are known as the *Yule-Walker equations*, whose solution results in the values, $r(0), \dots, r(l)$, which are then used as the initial conditions to solve the difference equation in (2.139) and obtain $r(k), \forall k \in \mathbb{Z}$.

Observe the special structure of the matrix in the linear system. This type of matrix is known as *Toeplitz*, and this is the property that will be exploited to solve efficiently such systems, which result when the autocorrelation matrix of a WSS process is involved; see Chapter 4.

Besides the autoregressive models, other types of stochastic models have been suggested and used. The *autoregressive-moving average* (ARMA) model of order (l, m) is defined by the difference equation,

$$u_n + a_1 u_{n-1} + \dots + a_l u_{n-l} = b_1 \eta_n + \dots + b_m \eta_{n-m}, \quad (2.142)$$

and the *moving average* model of order m , denoted as MA(m), is defined as

$$u_n = b_1 \eta_n + \dots + b_m \eta_{n-m}. \quad (2.143)$$

Note that the AR(l) and the MA(m) models can be considered as special cases of the ARMA(l, m). For a more theoretical treatment of the topic, see [1].

Example 2.4. Consider the AR(1) process,

$$u_n + a u_{n-1} = \eta_n.$$

Following the general methodology explained before, we have

$$r(k) + ar(k-1) = 0, \quad k = 1, 2, \dots$$

$$r(0) + ar(1) = \sigma_\eta^2.$$

Taking the first equation for $k = 1$ together with the second one readily results in

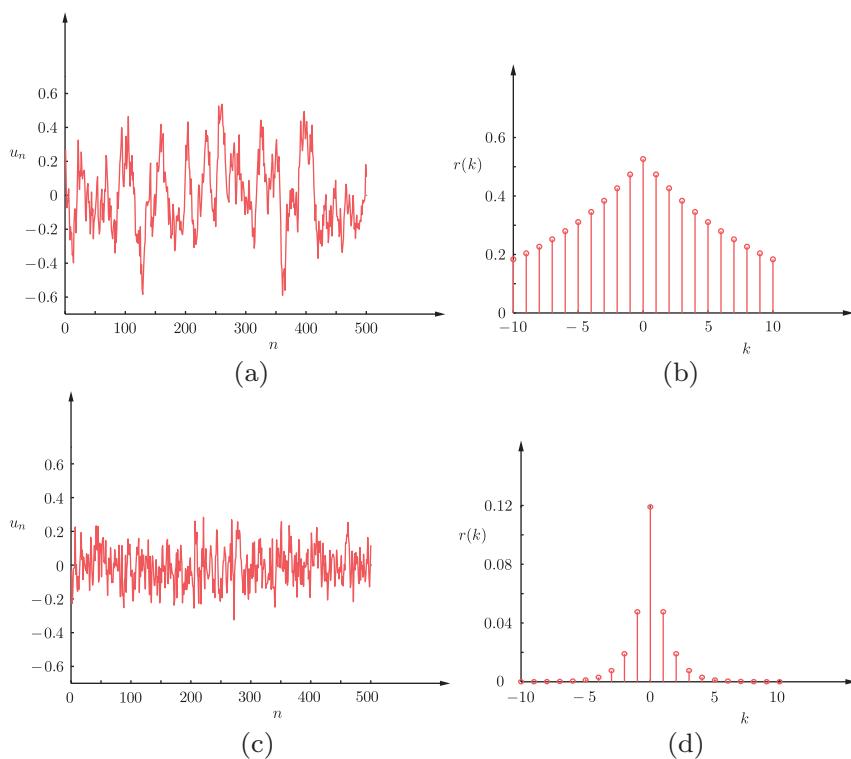
$$r(0) = \frac{\sigma_\eta^2}{1 - a^2}.$$

Plugging this value into the difference equation, we recursively obtain

$$r(k) = (-a)^{|k|} \frac{\sigma_\eta^2}{1 - a^2}, \quad k = 0, \pm 1, \pm 2, \dots, \quad (2.144)$$

where we used the property, $r(k) = r(-k)$. Observe that if $|a| > 1$, $r(0) < 0$, is meaningless. Also, $|a| < 1$ guarantees that the root of the characteristic polynomial ($z_* = -a$) is smaller than one. Moreover, $|a| < 1$ guarantees that $r(k) \rightarrow 0$ as $k \rightarrow \infty$. This is in line with common sense, since variables that are far away must be uncorrelated.

Figure 2.15 shows the time evolution of two AR(1) processes (after the processes have converged to be stationary) together with the respective autocorrelation sequences, for two cases, corresponding to $a = -0.9$ and $a = -0.4$. Observe that the larger the magnitude of a , the smoother the realization becomes and time variations are *slower*. This is natural, since nearby samples are highly correlated and so, on average, they tend to have similar values. The opposite is true for small values of a . For comparison purposes, Figure 2.16a is the case of $a = 0$, which corresponds to a white noise. Figure 2.16b shows the power spectral densities corresponding to the two cases of Figure 2.15. Observe that the faster the autocorrelation approaches zero, the more spread out the PSD is, and vice versa.

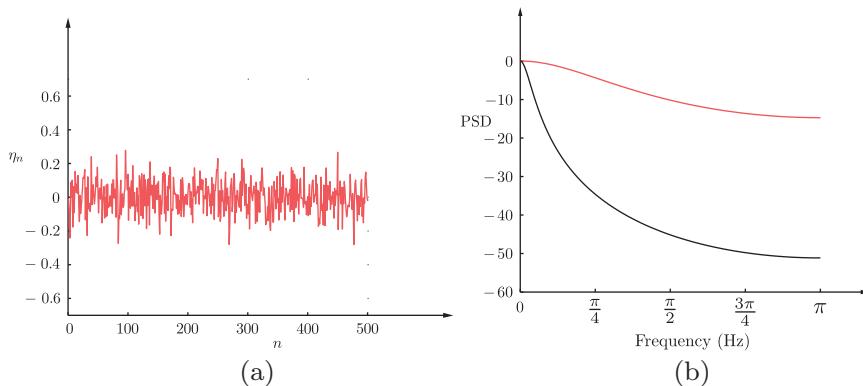
**FIGURE 2.15**

(a) The time evolution of a realization of the AR(1) with $a = -0.9$ and (b) the respective autocorrelation sequence. (c) The time evolution of a realization of the AR(1) with $a = -0.4$ and (d) the corresponding autocorrelation sequence.

2.5 INFORMATION THEORY

So far in this chapter, we have looked at some basic definitions and properties concerning probability theory and stochastic processes. In the same vein, we will now focus on the basic definitions and notions related to *information theory*. Although information theory was originally developed in the context of communications and coding disciplines, its application and use has now been adopted in a wide range of areas, including machine learning. Notions from information theory are used for establishing cost functions for optimization in parameter estimation problems, and concepts from information theory are employed to estimate unknown probability distributions in the context of constrained optimization tasks. We will discuss such methods later in this book.

The father of information theory is *Claude Elwood Shannon* (1916-2001), an American mathematician and electrical engineer. He founded information theory with the landmark paper “A mathematical theory of communication,” published in the Bell System Technical Journal in 1948. However, he is

**FIGURE 2.16**

(a) The time evolution of a realization from a white noise process. (b) The power spectral densities in dBs, for the two AR(1) sequences of Figure 2.15. The red one corresponds to $a = -0.4$ and the gray one to $a = -0.9$. The smaller the magnitude of a , the closer the process is to a white noise, and its power spectral density tends to increase the power with which high frequencies participate. Since the PSD is the Fourier transform of the autocorrelation sequence, observe that the broader a sequence is in time, the narrower its Fourier transform becomes, and vice versa.

also credited with founding digital circuit design theory in 1937, when, as a 21-year-old master's degree student at the Massachusetts Institute of Technology (MIT), he wrote his thesis demonstrating that electrical applications of Boolean algebra could construct and resolve any logical, numerical relationship. So he is also credited as a father of digital computers. Shannon, while working for the national defense during World War II, contributed to the field of cryptography, converting it from an art to a rigorous scientific field.

As is the case for probability, the notion of information is part of our everyday vocabulary. In this context, an event carries information if it is either unknown to us, or if the probability of its occurrence is very low and, in spite of that, it happens. For example, if one tells us that the sun shines bright during summer days in the Sahara desert, we could consider such a statement rather dull and useless. On the contrary, if somebody gives us news about snow in the Sahara during summer, that statement carries a lot of information and can possibly ignite a discussion concerning the climate change.

Thus, trying to formalize the notion of information from a mathematical point of view, it is reasonable to define it in terms of the negative logarithm of the probability of an event. If the event is certain to occur, it carries zero information content; however, if its probability of occurrence is low, then its information content has a large positive value.

2.5.1 DISCRETE RANDOM VARIABLES

Information

Given a discrete random variable, x , which takes values in the set \mathcal{X} , the *information* associated with any value $x \in \mathcal{X}$ is denoted as $I(x)$ and it is defined as

$$I(x) = -\log P(x) : \quad \text{Information Associated with } x = x \in \mathcal{X}. \quad (2.145)$$

Any base for the logarithm can be used. If the natural logarithm is chosen, information is measured in terms of *nats* (natural units). If the base 2 logarithm is employed, information is measured in terms of *bits* (binary digits). Employing the logarithmic function to define information is also in line with common sense reasoning that the information content of two statistically independent events should be the sum of the information conveyed by each one of them individually; $I(x, y) = -\ln P(x, y) = -\ln P(x) - \ln P(y)$.

Example 2.5. We are given a binary random variable $x \in \mathcal{X} = \{0, 1\}$, and assume that $P(1) = P(0) = 0.5$. We can consider this random variable as a source that generates and emits two possible values. The information content of each one of the two equiprobable events is

$$I(0) = I(1) = -\log_2 0.5 = 1 \text{ bit.}$$

Let us now consider another source of random events, which generates *code words* comprising k binary variables together. The output of this source can be seen as a random vector with binary-valued elements, $\mathbf{x} = [x_1, \dots, x_k]^T$. The corresponding probability space, \mathcal{X} , comprises $K = 2^k$ elements. If all possible values have the same probability, $1/K$, then the information content of each possible event is equal to

$$I(x_i) = -\log_2 \frac{1}{K} = k \text{ bits.}$$

We observe that in the case where the number of possible events is larger, the information content of each individual one (assuming equiprobable events) becomes larger. This is also in line with common sense reasoning, since if the source can emit a large number of (equiprobable) events, the occurrence of any one of them carries more information than a source that can only emit a few possible events.

Mutual and conditional information

Besides marginal probabilities, we have already been introduced to the concept of conditional probability. This leads to the definition of mutual information.

Given two discrete random variables, $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, the information content provided by the occurrence of the event $y = y$ about the event $x = x$ is measured by the *mutual information*, denoted as $I(x; y)$ and defined by

$$I(x, y) := \log \frac{P(x|y)}{P(x)} : \quad \text{Mutual Information.} \quad (2.146)$$

Note that if the two variables are statistically independent, then their mutual information is zero; this is most reasonable, since observing y says nothing about x . On the contrary, if by observing y it is certain that x will occur, as when $P(x|y) = 1$, then the mutual information becomes $I(x, y) = I(x)$, which is again in line with common reasoning. Mobilizing our now familiar product rule, we can see that

$$I(x, y) = I(y, x).$$

The *conditional information* of x given y is defined as

$$I(x|y) = -\log P(x|y) : \quad \text{Conditional Information.} \quad (2.147)$$

It is straightforward to show that

$$I(x, y) = I(x) - I(x|y). \quad (2.148)$$

Example 2.6. In a communications channel, the source transmits binary symbols, x , with probability $P(0) = P(1) = 1/2$. The channel is noisy, so the received symbols, y , may have changed polarity, due to noise, with the following probabilities:

$$P(y = 0|x = 0) = 1 - p,$$

$$P(y = 1|x = 0) = p,$$

$$P(y = 1|x = 1) = 1 - q,$$

$$P(y = 0|x = 1) = q.$$

This example illustrates in its simplest form the effect of a *communications channel*. Transmitted bits are hit by noise and what the receiver receives is the noisy (possibly wrong) information. The task of the receiver is to decide, upon reception of a sequence of symbols, which was the originally transmitted one.

The goal of our example is to determine the mutual information about the occurrence of $x = 0$ and $x = 1$ once $y = 0$ has been observed. To this end, we first need to compute the marginal probabilities,

$$P(y = 0) = P(y = 0|x = 0)P(x = 0) + P(y = 0|x = 1)P(x = 1) = \frac{1}{2}(1 - p + q),$$

and similarly,

$$P(y = 1) = \frac{1}{2}(1 - q + p).$$

Thus, the mutual information is

$$\begin{aligned} I(0, 0) &= \log_2 \frac{P(x = 0|y = 0)}{P(x = 0)} = \log_2 \frac{P(y = 0|x = 0)}{P(y = 0)} \\ &= \log_2 \frac{2(1 - p)}{1 - p + q}, \end{aligned}$$

and

$$I(1, 0) = \log_2 \frac{2q}{1 - p + q}.$$

Let us now consider that $p = q = 0$. Then $I(0, 0) = 1$ bit, which is equal to $I(x = 0)$, since the output specifies the input with certainty. If on the other hand $p = q = 1/2$, then $I(0, 0) = 0$ bits, since the noise can randomly change polarity with equal probability. If now $p = q = 1/4$, then $I(0, 0) = \log_2 \frac{3}{2} = 0.587$ bits and $I(1, 0) = -1$ bit. Observe that the mutual information can take negative values, too.

Entropy and average mutual information

Given a discrete random variable, $x \in \mathcal{X}$, its *entropy* is defined as the average information over all possible outcomes,

$$H(x) := - \sum_{x \in \mathcal{X}} P(x) \log P(x) : \quad \text{Entropy of } x.$$

(2.149)

Note that if $P(x) = 0$, $P(x) \log P(x) = 0$, by taking into consideration that $\lim_{x \rightarrow 0} x \log x = 0$.

In a similar way, the *average mutual information* between two random variables, x, y , is defined as

$$\begin{aligned} I(x, y) &:= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) I(x; y) \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log \frac{P(x|y)P(y)}{P(x)P(y)} \end{aligned}$$

or

$$I(x, y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log \frac{P(x, y)}{P(x)P(y)} : \text{ Average Mutual Information.} \quad (2.150)$$

It can be shown that

$$I(x, y) \geq 0,$$

and it is zero if x and y are statistically independent ([Problem 2.12](#)).

In comparison, the *conditional entropy* of x given y is defined as

$$H(x|y) := - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} P(x, y) \log P(x|y) : \text{ Conditional Entropy.} \quad (2.151)$$

It is readily shown, by taking into account the probability product rule, that

$$I(x, y) = H(x) - H(x|y). \quad (2.152)$$

Lemma 2.1. *The entropy of a random variable, $x \in \mathcal{X}$, takes its maximum value if all possible values, $x \in \mathcal{X}$, are equiprobable.*

Proof. The proof is given in [Problem 2.14](#). □

In other words, the entropy can be considered as a measure of randomness of a source that emits symbols randomly. The maximum value is associated with the maximum uncertainty of what is going to be emitted, since the maximum value occurs if all symbols are equiprobable. The smallest value of the entropy is equal to zero, which corresponds to the case where all events have zero probability with the exception of one, whose probability to occur is equal to one.

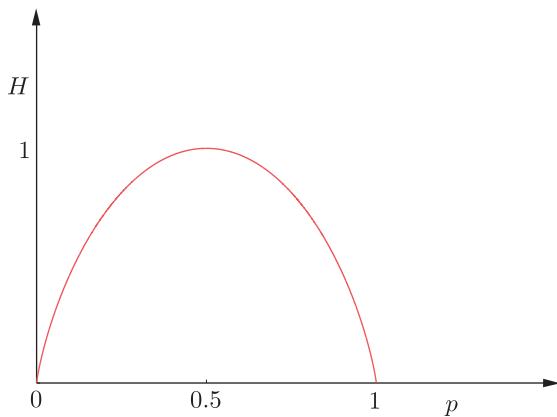
Example 2.7. Consider a binary source that transmits the values 1 or 0 with probabilities p and $1 - p$, respectively. Then the entropy of the associated random variable is

$$H(x) = -p \log_2 p - (1 - p) \log_2(1 - p).$$

[Figure 2.17](#) shows the graph for various values of $p \in [0, 1]$. Observe that the maximum value occurs for $p = 1/2$.

2.5.2 CONTINUOUS RANDOM VARIABLES

All the definitions given before can be generalized to the case of continuous random variables. However, this generalization must be made with caution. Recall that the probability of occurrence of any single value of a random variable that takes values in an interval in the real axis is zero. Hence, the corresponding information content is infinite.

**FIGURE 2.17**

The maximum value of the entropy for a binary random variable occurs if the two possible events have equal probability, $p = 1/2$.

To define the entropy of a continuous variable, x , we first *discretize* it and form the corresponding discrete variable, x_Δ ,

$$x_\Delta := n\Delta, \text{ if } (n-1)\Delta < x \leq n\Delta, \quad (2.153)$$

where $\Delta > 0$. Then,

$$P(x_\Delta = n\Delta) = P(n\Delta - \Delta < x \leq n\Delta) = \int_{(n-1)\Delta}^{n\Delta} p(x) dx = \Delta \bar{p}(n\Delta), \quad (2.154)$$

where $\bar{p}(n\Delta)$ is a number between the maximum and the minimum value of $p(x)$, $x \in (n\Delta - \Delta, n\Delta]$ (such a number exists by the mean value theorem). Then we can write,

$$H(x_\Delta) = - \sum_{n=-\infty}^{+\infty} \Delta \bar{p}(n\Delta) \log(\Delta \bar{p}(n\Delta)), \quad (2.155)$$

and since

$$\sum_{n=-\infty}^{+\infty} \Delta \bar{p}(n\Delta) = \int_{-\infty}^{+\infty} p(x) dx = 1,$$

we obtain

$$H(x_\Delta) = - \log \Delta - \sum_{n=-\infty}^{+\infty} \Delta \bar{p}(n\Delta) \log(\bar{p}(n\Delta)). \quad (2.156)$$

Note that $x_\Delta \rightarrow x$ as $\Delta \rightarrow 0$. However, if we take the limit in Eq. (2.156), then $-\log \Delta$ goes to infinity. This is the crucial difference compared to the discrete variables.

The entropy for a continuous random variable, x , is defined as the limit

$$H(x) := \lim_{\Delta \rightarrow 0} (H(x_\Delta) + \log \Delta),$$

or

$$H(x) = - \int_{-\infty}^{+\infty} p(x) \log p(x) dx : \text{ Entropy.} \quad (2.157)$$

This is the reason that the entropy of a continuous variable is also called *differential entropy*.

Note that the entropy is still a measure of randomness (uncertainty) of the distribution describing x . This is demonstrated via the following example.

Example 2.8. We are given a random variable $x \in [a, b]$. Of all the possible pdfs that can describe this variable, find the one that maximizes the entropy.

This task translates to the following constrained optimization task:

$$\begin{aligned} \text{maximize with respect to } p : H &= - \int_a^b p(x) \ln p(x) dx, \\ \text{subject to: } &\int_a^b p(x) dx = 1. \end{aligned}$$

The constraint guarantees that the function to result is indeed a pdf. Using calculus of variations to perform the optimization ([Problem 2.15](#)), it turns out that

$$p(x) = \begin{cases} \frac{1}{b-a}, & \text{if } x \in [a, b], \\ 0, & \text{otherwise.} \end{cases}$$

In other words, the result is the uniform distribution, which is indeed the most random one since it gives no preference to any particular subinterval of $[a, b]$.

We will come to this method of estimating pdfs in [Section 12.4.1](#). This elegant method for estimating pdfs comes from Jaynes [3, 4], and it is known as the *maximum entropy method*. In its more general form, more constraints are involved to fit the needs of the specific problem.

Average mutual information and conditional information

Given two continuous random variables, the average mutual information is defined as

$$I(x, y) := \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx dy \quad (2.158)$$

and the conditional entropy of x given y

$$H(x|y) := \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} p(x, y) \log p(x|y) dx dy. \quad (2.159)$$

Using standard arguments and the product rule, it is easy to show that

$$I(x; y) = H(x) - H(x|y) = H(y) - H(y|x). \quad (2.160)$$

Relative entropy or Kullback-Leibler divergence

The *relative entropy* or *Kullback-Leibler divergence* is a quantity that has been developed within the context of information theory for measuring similarity between two pdfs. It is widely used in machine

learning optimization tasks when pdfs are involved; see [Chapter 12](#). Given two pdfs, $p(\cdot)$ and $q(\cdot)$, their Kullback-Leibler divergence, denoted as $\text{KL}(p||q)$, is defined as

$$\boxed{\text{KL}(p||q) := \int_{-\infty}^{+\infty} p(x) \log \frac{p(x)}{q(x)} dx : \quad \text{Kullback-Leibler Divergence.}} \quad (2.161)$$

Note that

$$I(x, y) = \text{KL}(p(x, y)||p(x)p(y)).$$

The Kullback-Leibler divergence is *not* symmetric, i.e., $\text{KL}(p||q) \neq \text{KL}(q||p)$ and it can be shown that it is a nonnegative quantity (the proof is similar to the proof that the mutual information is nonnegative; see [Problem 12.16](#) of [Chapter 12](#)). Moreover, it is zero if and only if $p = q$.

Note that all we have said concerning entropy and mutual information is readily generalized to the case of random vectors.

2.6 STOCHASTIC CONVERGENCE

We will close this memory-refreshing tour of the theory of probability and related concepts with some definitions concerning convergence of sequences of random variables.

Let a sequence of random variables,

$$x_0, x_1, \dots, x_n \dots$$

We can consider this sequence as a discrete-time stochastic process. Due to the randomness, a realization of this process, as shown by

$$x_0, x_1, \dots, x_n \dots,$$

may converge or may not. Thus, the notion of convergence of random variables has to be treated carefully, and different interpretations have been developed.

Recall from our basic calculus that a sequence of numbers, x_n , converges to a value, x , if $\forall \epsilon > 0$ there exists a number, $n(\epsilon)$, such that

$$|x_n - x| < \epsilon, \quad \forall n \geq n(\epsilon). \quad (2.162)$$

Convergence everywhere

We say that a random sequence *converges everywhere* if every realization, x_n , of the random process converges to a value x , according to the definition given in Eq. (2.162). Note that every realization converges to a different value, which itself can be considered as the outcome of a random variable x , and we write

$$x_n \xrightarrow{n \rightarrow \infty} x. \quad (2.163)$$

It is common to denote a realization (outcome) of a random process as $x_n(\zeta)$, where ζ denotes a specific experiment.

Convergence almost everywhere

A weaker version of convergence, compared to the previous one, is the *convergence almost everywhere*. Let the set of outcomes ζ such as

$$\lim x_n(\zeta) = x(\zeta), \quad n \rightarrow \infty.$$

We say that the sequence x_n converges almost everywhere, if

$$P(x_n \rightarrow x) = 1, \quad n \rightarrow \infty. \quad (2.164)$$

Note that $\{x_n \rightarrow x\}$ denotes the event comprising *all* the outcomes such as $\lim x_n(\zeta) = x(\zeta)$. The difference with the convergence everywhere is that now it is allowed to a finite or countably infinite number of realizations (that is, to a set of zero probability) not to converge. Often, this type of convergence is referred to as *almost sure* convergence or convergence *with probability 1*.

Convergence in the mean-square sense

We say that a random sequence, x_n , converges to the random variable, x , in the *mean-square* (MS) *sense*, if

$$\mathbb{E}[(x_n - x)^2] \rightarrow 0, \quad n \rightarrow \infty. \quad (2.165)$$

Convergence in probability

Given a random sequence, x_n , a random variable, x , and a nonnegative number ϵ , then $\{|x_n - x| > \epsilon\}$ is an event. We define the new sequence of numbers, $P(\{|x_n - x| > \epsilon\})$. We say that x_n converges to x *in probability* if the constructed sequence of numbers tends to zero,

$$P(\{|x_n - x| > \epsilon\}) \rightarrow 0, \quad n \rightarrow \infty, \quad \forall \epsilon > 0. \quad (2.166)$$

Convergence in distribution

Given a random sequence, x_n , and a random variable, x , let $F_n(x)$ and $F(x)$ be the cdfs, respectively. We say that x_n converges to x *in distribution*, if

$$F_n(x) \rightarrow F(x), \quad n \rightarrow \infty, \quad (2.167)$$

for every point x of continuity of $F(x)$.

It can be shown that if a random sequence converges either almost everywhere or in the MS sense then it necessarily converges in probability, and if it converges in probability then it necessarily converges in distribution. The converse arguments are not necessarily true. In other words, the weakest version of convergence is that of convergence in distribution.

PROBLEMS

2.1 Derive the mean and variance for the binomial distribution.

2.2 Derive the mean and variance for the uniform distribution.

2.3 Derive the mean and covariance matrix of the multivariate Gaussian.

2.4 Show that the mean and variance of the beta distribution with parameters a and b are given by

$$\mathbb{E}[x] = \frac{a}{a+b}$$

and

$$\sigma_x^2 = \frac{ab}{(a+b)^2(a+b+1)}.$$

Hint. Use the property $\Gamma(a+1) = a\Gamma(a)$.

2.5 Show that the normalizing constant in the beta distribution with parameters a, b is given by

$$\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}.$$

2.6 Show that the mean and variance of the gamma pdf

$$\text{Gamma}(x|a,b) = \frac{b^a}{\Gamma(a)} x^{a-1} e^{-bx}, \quad a, b, x > 0$$

are given by

$$\mathbb{E}[x] = \frac{a}{b},$$

$$\sigma_x^2 = \frac{a}{b^2}.$$

2.7 Show that the mean and variance of a Dirichlet pdf with K variables $x_k, k = 1, 2, \dots, K$ and parameters $a_k, k = 1, 2, \dots, K$, are given by

$$\begin{aligned}\mathbb{E}[x_k] &= \frac{a_k}{\bar{a}}, \quad k = 1, 2, \dots, K \\ \sigma_k^2 &= \frac{a_k(\bar{a} - a_k)}{\bar{a}^2(1 + \bar{a})}, \quad k = 1, 2, \dots, K, \\ \text{cov}[x_i x_j] &= -\frac{a_i a_j}{\bar{a}^2(1 + \bar{a})}, \quad i \neq j,\end{aligned}$$

where $\bar{a} = \sum_{k=1}^K a_k$.

2.8 Show that the sample mean, using N i.i.d. drawn samples, is an unbiased estimator with variance that tends to zero asymptotically, as $N \rightarrow \infty$.

2.9 Show that for WSS processes

$$r(0) \geq |r(k)|, \quad \forall k \in \mathbb{Z},$$

and that for jointly WSS processes

$$r_u(0)r_v(0) \geq |r_{uv}(k)|, \quad \forall k \in \mathbb{Z}.$$

2.10 Show that the autocorrelation of the output of a linear system, with impulse response $w_n, n \in \mathbb{Z}$, is related to the autocorrelation of the input WSS process, via

$$r_d(k) = r_u(k) * w_k * w_{-k}^*.$$

2.11 Show that

$$\ln x \leq x - 1.$$

2.12 Show that

$$I(x, y) \geq 0.$$

Hint. Use the inequality of Problem 2.11.

2.13 Show that if $a_i, b_i, i = 1, 2, \dots, M$, are positive numbers, such as

$$\sum_{i=1}^M a_i = 1 \quad \text{and} \quad \sum_{i=1}^M b_i \leq 1,$$

then

$$-\sum_{i=1}^M a_i \ln a_i \leq -\sum_{i=1}^M a_i \ln b_i.$$

2.14 Show that the maximum value of the entropy of a random variable occurs if all possible outcomes are equiprobable.

2.15 Show that from all the pdfs that describe a random variable in an interval $[a, b]$, the uniform one maximizes the entropy.

REFERENCES

- [1] P.J. Brockwell, R.A. Davis, Time Series: Theory and Methods, second ed., Springer, New York, 1991.
- [2] R.T. Cox, Probability, frequency and reasonable expectation. Am. J. Phys. 14 (1) (1946) 1-13.
- [3] E.T. Jaynes, Information theory and statistical mechanics. Phys. Rev. 106 (4) (1957) 620-630.
- [4] E.T. Jaynes, Probability Theory: The Logic of Science, Cambridge University Press, Cambridge, 2003.
- [5] A.N. Kolmogorov, Foundations of the Theory of Probability, second ed., Chelsea Publishing Company, New York, 1956.
- [6] A. Papoulis, S.U. Pillai, Probability, Random Variables and Stochastic Processes, fourth ed., McGraw Hill, New York, 2002.
- [7] M.B. Priestly, Spectral Analysis and Time Series, Academic Press, New York, 1981.
- [8] J. Proakis, D. Manolakis, Digital Signal Processing, second ed., MacMillan, New York, 1992.

This page intentionally left blank

LEARNING IN PARAMETRIC MODELING: BASIC CONCEPTS AND DIRECTIONS

3

CHAPTER OUTLINE

3.1	Introduction	53
3.2	Parameter Estimation: The Deterministic Point of View	54
3.3	Linear Regression	57
3.4	Classification	60
	<i>Generative Versus Discriminative Learning</i>	63
	<i>Supervised, Semisupervised, and Unsupervised Learning</i>	64
3.5	Biased Versus Unbiased Estimation	64
3.5.1	Biased or Unbiased Estimation?	65
3.6	The Cramér-Rao Lower Bound	67
3.7	Sufficient Statistic	70
3.8	Regularization	72
	<i>Inverse Problems: Ill-Conditioning and Overfitting</i>	74
3.9	The Bias-Variance Dilemma	77
3.9.1	Mean-Square Error Estimation	77
3.9.2	Bias-Variance Tradeoff	78
3.10	Maximum Likelihood Method	82
3.10.1	Linear Regression: The Nonwhite Gaussian Noise Case	84
3.11	Bayesian Inference	84
3.11.1	The Maximum A Posteriori Probability Estimation Method	88
3.12	Curse of Dimensionality	89
3.13	Validation	91
	<i>Cross-Validation</i>	92
3.14	Expected and Empirical Loss Functions	93
3.15	Nonparametric Modeling and Estimation	95
Problems	97	
References	102	

3.1 INTRODUCTION

Parametric modeling is a theme that runs across the spine of this book. A number of chapters focus on different aspects of this important problem. This chapter provides basic definitions and concepts related to the task of learning when parametric models are mobilized to describe the available data.

As it has already been pointed out in the introductory chapter, a large class of machine learning problems ends up in being equivalent to a function estimation/approximation task. The function is “learned” during the learning/training phase by digging in the information that resides in the available training data set. This function relates the so-called input variables to the output variable(s). Once this functional relationship is established, one can in turn exploit it to predict the value(s) of the output(s), based on measurements obtained from the respective input variables; these predictions can then be used to proceed to the decision-making phase.

In parametric modeling, the aforementioned functional dependence is defined via a set of unknown parameters, whose number is *fixed*. In contrast, in the so-called *nonparametric* methods, unknown parameters may still be involved, yet their number depends on the size of the data set. Nonparametric methods will also be treated in this book. However, the emphasis in this chapter lies in the former ones.

In parametric modeling, there are two possible paths to deal with the uncertainty imposed by the unknown values of the parameters. According to the first one, specific values are obtained and assigned to the unknown parameters. In the other approach, which has a stronger statistical flavor, parametric models are adopted in order to describe the underlying probability distributions, which describe the input and output variables, without it being necessary to obtain specific values for the unknown parameters.

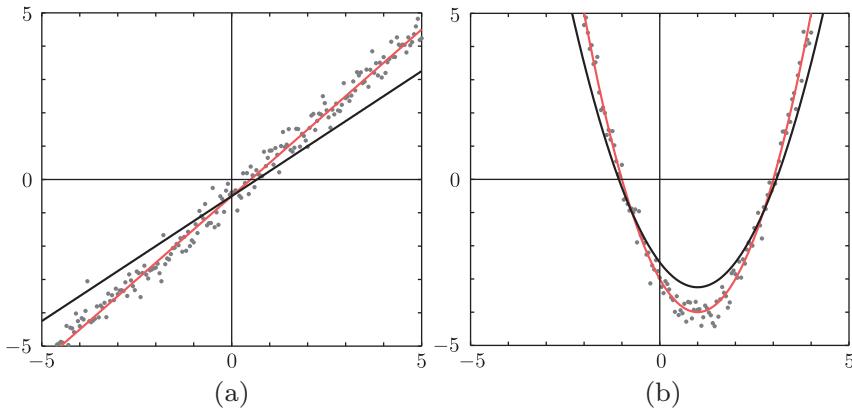
Two of the major machine learning tasks, namely the regression and the classification, are presented and the main directions in dealing with these problems are exposed. Various issues that are related to the parameter estimation task, such as estimator efficiency, bias-variance dilemma, overfitting, curse of dimensionality are introduced and discussed. The chapter can also be considered as a road map to the rest of the book. However, instead of just presenting the main ideas and directions in a rather “dry” way, we chose to deal and work with the involved tasks by adopting simple models and techniques, so that the reader gets a better feeling of the topic. An effort was made to pay more attention to the scientific notions than to algebraic manipulations and mathematical details, which will, unavoidably, be used to a larger extent while “embroidering” the chapters to follow.

The Least-Squares (LS), the Maximum Likelihood (ML), the Regularization as well as the Bayesian Inference techniques are presented and discussed. An effort has been made to assist the reader to grasp an informative view of the big picture conveyed by the book. Thus, this chapter could also be used as an overview introduction to the parametric modeling task in the realm of machine learning.

3.2 PARAMETER ESTIMATION: THE DETERMINISTIC POINT OF VIEW

The task of estimating the value of an unknown parameter vector, θ , has been at the center of interest in a number of application areas. For example, in the early years in the University, one of the very first tasks any student has to study is the so-called curve fitting problem. Given a set of data points, one must find a curve or a surface that “fits” the data. The usual path to follow is to *adopt* a functional form, such as a linear function or a quadratic one, and try to estimate the associated unknown coefficients so that the graph of the function “passes through” the data and follows their deployment in space as close as possible. [Figures 3.1a](#) and b are two such examples. The data lie in the \mathbb{R}^2 space and are given to us as a set of points (y_n, x_n) , $n = 1, 2, \dots, N$. The adopted functional form for the curve corresponding to [Figure 3.1a](#) is

$$y = f_{\theta}(x) = \theta_0 + \theta_1 x, \quad (3.1)$$

**FIGURE 3.1**

Fitting (a) a linear function and (b) a quadratic one. The red lines are the optimized ones.

and for the case of Figure 3.1b

$$y = f_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2. \quad (3.2)$$

The unknown parameter vectors are $\theta = [\theta_0, \theta_1]^T$ and $\theta = [\theta_0, \theta_1, \theta_2]^T$, respectively. In both cases, the parameter values, which define the curves drawn by the red lines, provide a much better fit compared to the values associated with the black ones. In both cases, the task comprises two steps: (a) first adopt a specific *parametric functional* form, which we reckon to be more appropriate for the data at hand and (b) estimate the values of the unknown parameters in order to obtain a “good” fit.

In the more general and formal setting, the task can be defined as follows. Given a set of data points, (y_n, \mathbf{x}_n) , $y_n \in \mathbb{R}$, $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, and a parametric set of functions,

$$\mathcal{F} := \left\{ f_{\theta}(\cdot) : \theta \in \mathcal{A} \subseteq \mathbb{R}^K \right\}, \quad (3.3)$$

find a function in \mathcal{F} , which will be denoted as $f(\cdot) := f_{\theta_*}(\cdot)$, such that given a value of $\mathbf{x} \in \mathbb{R}^l$, $f(\mathbf{x})$ best approximates the corresponding value $y \in \mathbb{R}$. We start our discussion by considering y to be a real variable, $y \in \mathbb{R}$, and as we move on and understand better the various “secrets,” we will allow it to move to higher dimensional Euclidean spaces. The value θ_* is the value that results from the estimation procedure. The values of θ_* that define the red line curves in Figures 3.1a and b are

$$\theta_* = [-0.5, 1]^T, \quad \theta_* = [-3, -2, 1]^T, \quad (3.4)$$

respectively.

To reach a decision with respect to the choice of \mathcal{F} is not an easy task. For the case of the data in Figure 3.1, we were a bit “lucky.” First, the data live in the two-dimensional space, where we have the luxury of visualization. Second, the data were scattered along curves whose shape is pretty familiar to us; hence, a simple inspection suggested the proper family of functions, for each one of the two cases. Obviously, real life is hardly as generous as that and in the majority of practical applications,

the data reside in high-dimensional spaces and/or the shape of the surface (hypersurface, for spaces of dimensionality higher than three) can be quite complex. Hence, the choice of \mathcal{F} , which dictates the functional form (e.g., linear, quadratic, etc.) is not easy. In practice, one has to use as much a priori information as possible concerning the physical mechanism that underlies the generation of the data, and most often use different families of functions and finally keep the one that results in the best performance, according to a chosen criterion.

Having adopted a parametric family of functions, \mathcal{F} , one has to get an estimate for the unknown set of parameters. To this end, a measure of fitness has to be adopted. The more classical approach is to adopt a *loss* function, which quantifies the deviation/error between the measured value of y and the predicted one using the corresponding measurements x , as in $f_\theta(x)$. In a more formal way, we adopt a *nonnegative* (loss) function,

$$\mathcal{L}(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \mapsto [0, \infty),$$

and compute θ_* so as to minimize the total loss, or as we say the *cost*, over all the data points, or

$$f(\cdot) := f_{\theta_*}(\cdot) : \theta_* = \arg \min_{\theta \in \mathcal{A}} J(\theta), \quad (3.5)$$

where

$$J(\theta) := \sum_{n=1}^N \mathcal{L}(y_n, f_\theta(x_n)), \quad (3.6)$$

assuming that a minimum exists. Note that, in general, there may be more than one optimal values θ_* , depending on the shape of $J(\theta)$.

As the book evolves, we are going to see different loss functions and different parametric families of functions. For the sake of simplicity, for the rest of this chapter we will adhere to the LS loss function,

$$\mathcal{L}(y, f_\theta(x)) = (y - f_\theta(x))^2,$$

and to the linear class of functions.

The LS loss function is credited to the great mathematician Carl Frederich Gauss, who proposed the fundamentals of the LS method in 1795 at the age of eighteen. However, it was Adrien-Marie Legendre who first published the method in 1805, working independently. Gauss published it in 1809. The strength of the method was demonstrated when it was used to predict the location of the asteroid Ceres. Since then, the LS loss function has “haunted” all scientific fields, and even if it is not used directly, it is, most often, used as the standard against which the performance of more modern alternatives are compared. This success is due to some nice properties that this loss criterion has, which will be explored as we move on in this book.

The combined choice of linearity with the LS loss function turns out to simplify the algebra and hence becomes very pedagogic for introducing the newcomer to the various “secrets” that underlie the area of parameter estimation. Moreover, understanding linearity is very important. Treating nonlinear tasks, most often, turns out to finally resort to a linear problem. Take, for example, the nonlinear model in Eq. (3.2) and consider the transformation

$$\mathbb{R} \ni x \mapsto \phi(x) := \begin{bmatrix} x \\ x^2 \end{bmatrix} \in \mathbb{R}^2. \quad (3.7)$$

Then, Eq. (3.2) becomes

$$y = \theta_0 + \theta_1 \phi_1(x) + \theta_2 \phi_2(x). \quad (3.8)$$

That is, the model is now linear with respect to the components $\phi_k(x)$, $k = 1, 2$, of the two-dimension image, $\phi(x)$, of x . As a matter of fact, this simple trick is at the heart of a number of nonlinear methods that will be treated later on in the book. No doubt, the procedure can be generalized to any number, K , of functions, $\phi_k(x)$, $k = 1, 2, \dots, K$, and besides monomials, other types of nonlinear functions can be used such as exponentials, splines, wavelets, to name a few. In spite of the nonlinear nature of the input-output dependence modeling, we still consider this model to be linear, because it retains its linearity with respect to the involved unknown parameters, θ_k , $k = 1, 2, \dots, K$. Although for the rest of the chapter we will adhere to linear functions, in order to keep our discussion simpler, everything that will be said applies to nonlinear ones. All that is needed is to replace x with $\phi(x) := [\phi_1(x), \dots, \phi_K(x)]^T \in \mathbb{R}^K$.

In the sequel, we will present two examples in order to demonstrate the use of parametric modeling. These examples are generic and can represent a wide class of problems.

3.3 LINEAR REGRESSION

In statistics, the term *regression* was coined to define the task of modeling the relationship of a *dependent* random variable, y , which is considered to be the response of a system, when this is activated by a set of random variables, x_1, x_2, \dots, x_l , which will be represented as the components of an equivalent random vector \mathbf{x} . The relationship is modeled via an additive disturbance or noise term, η . The block diagram of the process, which relates the involved variables, is given in Figure 3.2. The noise variable, η , is an *unobserved* random variable. The goal of the regression task is to estimate the parameter vector, θ , given a set of measurements, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, that we have at our disposal. This is also known as the *training data set*, or the *observations*. The dependent variable is usually known as the *output* variable and the vector \mathbf{x} as the *input* vector or the *regressor*. If we model the system as a linear combiner, the dependence relationship is written as

$$y = \theta_0 + \theta_1 x_1 + \dots + \theta_l x_l + \eta = \theta_0 + \theta^T \mathbf{x} + \eta. \quad (3.9)$$

The parameter θ_0 is known as the *bias* or the *intercept*. Usually, this term is absorbed by the parameter vector θ with a simultaneous increase of the dimension of \mathbf{x} by adding the constant 1 as its last element. Indeed, we can write

$$\theta_0 + \theta^T \mathbf{x} + \eta = [\theta^T, \theta_0] \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} + \eta.$$

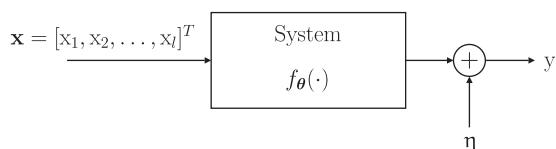


FIGURE 3.2

Block diagram showing the input-output relation in a regression model.

From now on, the regression model will be written as

$$\mathbf{y} = \boldsymbol{\theta}^T \mathbf{x} + \eta, \quad (3.10)$$

and, unless otherwise stated, this notation means that the bias term has been absorbed by $\boldsymbol{\theta}$ and \mathbf{x} has been extended by adding 1 as an extra component. Because the noise variable is unobserved, we need a model to be able to *predict* the output value of y , given the value \mathbf{x} .

In linear regression, we adopt the following prediction model:

$$\hat{y} = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \cdots + \hat{\theta}_l x_l := \hat{\boldsymbol{\theta}}^T \mathbf{x}. \quad (3.11)$$

Using the LS loss function, the estimate $\hat{\boldsymbol{\theta}}$ is set equal to $\boldsymbol{\theta}_*$, which minimizes the square difference between \hat{y}_n and y_n , over the set of the available observations; that is, by minimizing, with respect to $\boldsymbol{\theta}$, the cost function

$$J(\boldsymbol{\theta}) = \sum_{n=1}^N (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2. \quad (3.12)$$

Taking the derivative (gradient) with respect to $\boldsymbol{\theta}$ and equating to the zero vector, $\mathbf{0}$, we obtain

$$\left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T \right) \hat{\boldsymbol{\theta}} = \sum_{n=1}^N \mathbf{x}_n y_n. \quad (3.13)$$

Another more popular way to write the previously obtained relation is via the so-called input matrix, X , defined as the $N \times (l+1)$ matrix, which has as rows the (extended) regressor vectors, \mathbf{x}_n^T , $n = 1, 2, \dots, N$, expressed as

$$X := \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1l} & 1 \\ x_{21} & \dots & x_{2l} & 1 \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{Nl} & 1 \end{bmatrix}. \quad (3.14)$$

Then, it is straightforward to see that Eq. (3.13) can be written as

$$X^T X \hat{\boldsymbol{\theta}} = X^T \mathbf{y}, \quad (3.15)$$

where

$$\mathbf{y} := [y_1, y_2, \dots, y_N]^T, \quad (3.16)$$

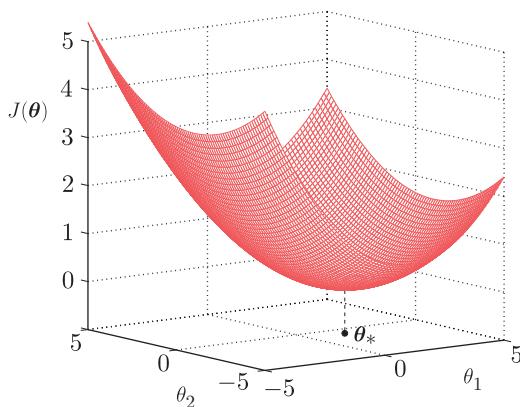
and the LS estimate is given by

$\hat{\boldsymbol{\theta}} = (X^T X)^{-1} X^T \mathbf{y} :$ The LS Estimate,

(3.17)

assuming, of course, that $(X^T X)^{-1}$ exists.

In other words, the obtained estimate of the parameter vector is given by a *linear set of equations*. This is a major advantage of the LS loss function, when applied to a linear model. Moreover, this solution is *unique*, provided that the $(l+1) \times (l+1)$ matrix $X^T X$ is invertible. The uniqueness is due to the parabolic shape of the graph of the LS cost function. This is illustrated in [Figure 3.3](#) for the

**FIGURE 3.3**

The least-squares loss function has a unique minimum at the point θ_* .

two-dimensional space. It is readily observed that the graph has a unique minimum. This is a consequence of the fact the LS cost function is a strictly convex one. Issues related to convexity of loss functions will be treated in more detail in [Chapter 8](#).

Example 3.1. Consider the system that is described by the following model:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \eta := [0.25, -0.25, 0.25] \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} + \eta, \quad (3.18)$$

where η is a Gaussian random variable of zero mean and variance $\sigma^2 = 1$. The random variables x_1 and x_2 are assumed to be mutually independent, and uniformly distributed over the interval $[0, 10]$. Generate $N = 50$ points for each one of the three random variables. For each triplet, use Eq. (3.18) to generate the corresponding value, y , of y . In this way, the points (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, 50$, are generated, where each observation, \mathbf{x}_n of \mathbf{x} , lies in \mathbb{R}^3 , after extending it by adding one as its last element. These are used as the training points to obtain the LS estimates of the coefficients of the linear model

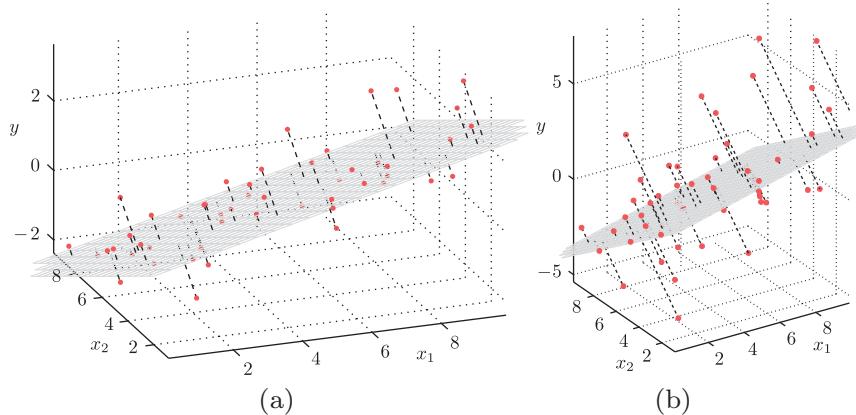
$$\hat{y} = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2.$$

Repeat the experiments with $\sigma^2 = 10$.

The values of the LS optimal estimates are obtained by solving a 3×3 linear system of equations and they are

- (a) $\hat{\theta}_0 = 0.6642$, $\hat{\theta}_1 = 0.2471$, $\hat{\theta}_2 = -0.3413$,
- (b) $\hat{\theta}_0 = 1.5598$, $\hat{\theta}_1 = 0.2408$, $\hat{\theta}_2 = -0.5386$,

for the two cases, respectively. [Figures 3.4a](#) and b show the recovered planes. Observe that in the case of [Figure 3.4a](#), corresponding to a noise variable of small variance, the obtained plane follows the data points much closer, compared to that of [Figure 3.4b](#).

**FIGURE 3.4**

Fitting a plane using the LS method for (a) a low variance and (b) a high variance noise case. Note that when the noise variance in the regression model is low, a much better fit to the data set is obtained.

Remarks 3.1.

- The set of points $(\hat{y}_n, x_{n1}, \dots, x_{nl}), n = 1, 2, \dots, N$, lie on a *hyperplane* in the \mathbb{R}^{l+1} space. Equivalently, they lie on a hyperplane that crosses the origin and, thus, it is a linear subspace in the extended space \mathbb{R}^{l+2} , when one absorbs θ_0 in θ , as explained previously.
- Notice that the prediction model in Eq. (3.11) could still be used, even if the true systems' structure does not obey the linear model in Eq. (3.9). For example, the true dependence between y and x may be a nonlinear one. Well, in such a case, the predictions of the y 's, based on the model in Eq. (3.11), may not be satisfactory. It all depends on the deviation of our adopted model from the true structure of the system that generates the data.
- The prediction performance of the model depends, also, on the statistical properties of the noise variable. This is an important issue. We will see later on that, depending on the statistical properties of the noise variable, some loss functions and methods may be more suitable than others.
- The two previous remarks suggest that in order to quantify the performance of an estimator some related criteria are necessary. In [Section 3.9](#), we will present some theoretical touches that shed light on certain aspects related to the performance of an estimator.

3.4 CLASSIFICATION

Classification is the task of predicting the class to which an object, known as *pattern*, belongs. The pattern is assumed to belong to one and only one among a number of a priori known classes. Each pattern is *uniquely* represented by a set of measurements, known as *features*. One of the early stages in designing a classification system is to select an appropriate set of feature variables. These should “encode” as much class-discriminatory information, so that, by measuring their value for a given pattern, to be able to predict, with high enough probability, the class of the pattern. Selecting the appropriate set of features,

for each problem is not an easy task and it comprises one of the most important areas within the field of *Pattern Recognition* (e.g., [11, 35]). Having selected, say, l feature (random) variables, x_1, x_2, \dots, x_l , we stack them as the components of the so-called *feature vector*, $\mathbf{x} \in \mathbb{R}^l$. The goal is to design a *classifier*, such as a function¹ $f(\mathbf{x})$, or equivalently a *decision surface*, $f(\mathbf{x}) = 0$, in \mathbb{R}^l , so that given the values in a feature vector, \mathbf{x} , which corresponds to a pattern, we will be able to *predict* the class to which the pattern belongs.

To formulate the task in mathematical terms, each class is represented by the *class label* variable, y . For the simple two-class classification task, this can take either of two values, depending on the class, e.g., 1, -1, or 1, 0, etc. Then, given the value of \mathbf{x} , corresponding to a specific pattern, its class label is predicted according to the rule,

$$\hat{y} = \phi(f(\mathbf{x})),$$

where $\phi(\cdot)$ is a nonlinear function that indicates on which side of the decision surface, $f(\mathbf{x}) = 0$, \mathbf{x} lies. For example, if the class labels are ± 1 , the nonlinear function is chosen to be the sign function, or $\phi(\cdot) = \text{sgn}(\cdot)$. It is now clear that what we have said so far in the previous section can be transferred here and the task becomes that of estimating a function $f(\cdot)$, based on a set of training points $(y_n, \mathbf{x}_n) \in D \times \mathbb{R}^l$, $n = 1, 2, \dots, N$, where D denotes the discrete set in which y lies. Function $f(\cdot)$ is selected so as to belong in a specific parametric class of functions, \mathcal{F} , and the goal is, once more, to estimate the parameters so that the deviation between the true class labels, y_n , and the predicted ones, \hat{y}_n , is minimum according to a preselected criterion. So, is the classification any different from the regression task?

The answer to the previous question is that they are similar, yet different. Note that in a classification task, the dependent variables are of a *discrete* nature, in contrast to the regression, where they lie in an interval. This suggests that, in general, different techniques have to be adopted to optimize the parameters. For example, the most obvious choice for a criterion in a classification task is the probability of error. However, in a number of cases, one can attack both tasks using the same type of loss functions, as we will do in this section; even if such an approach is adopted, in spite of the similarities in their mathematical formalism, the goals of the two tasks remain different.

In the regression task, the function $f(\cdot)$ has to “explain” the data generation mechanism. The corresponding surface in the (y, \mathbf{x}) space \mathbb{R}^{l+1} should develop so as to follow the spread of the data in the space, as close as possible. In contrast, in classification, the goal is to place the corresponding surface $f(\mathbf{x}) = 0$, in \mathbb{R}^l , so as to separate the data that belong to different classes as much as possible. The goal of a classifier is to *partition* the space where the features vectors lie into regions and associate each region with a class. [Figure 3.5](#) illustrates two cases of classification tasks. The first one is an example of two linearly separable classes, where a straight line can separate the two classes, and the second one of two nonlinearly separable classes, where the use of a linear classifier would have failed to separate the two classes.

Let us now make what we have said, so far, more concrete. We are given a set of training patterns, $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, that belong to either of two classes, say ω_1 and ω_2 . The goal is to design a hyperplane

$$\begin{aligned} f(\mathbf{x}) &= \theta_0 + \theta_1 x_1 + \dots + \theta_l x_l \\ &= \boldsymbol{\theta}^T \mathbf{x} = 0, \end{aligned}$$

¹ In the more general case, a set of functions.

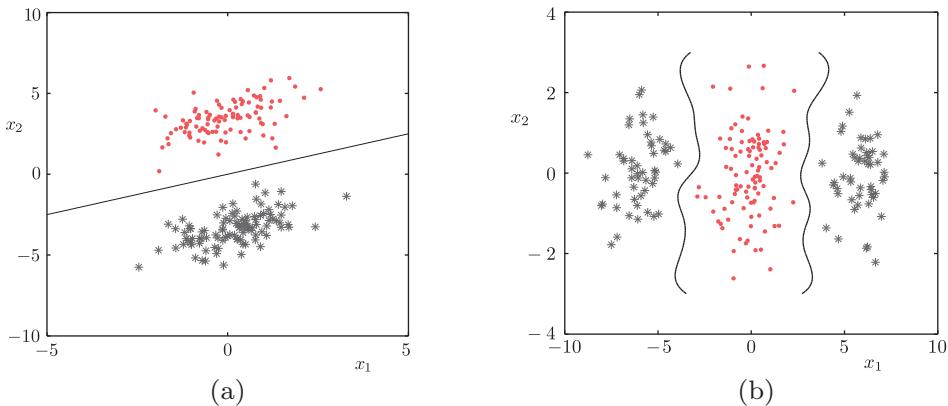


FIGURE 3.5

Examples of two-class classification tasks. (a) A linearly separable and (b) a nonlinearly separable one. The goal of a classifier is to divide the space into regions and associate each region with a class.

where we have absorbed the bias θ_0 in $\boldsymbol{\theta}$ and extend the dimension of \mathbf{x} , as it has already been explained before. Our aim is to place this hyperplane in between the two classes. Obviously, any point lying on this hyperplane scores a zero, $f(\mathbf{x}) = 0$, and the points lying on either side of the hyperplane score either a positive ($f(\mathbf{x}) > 0$) or a negative value ($f(\mathbf{x}) < 0$), depending on which side of the hyperplane they lie. We, therefore, should train our classifier so that the points from one class score a positive value and the points of the other a negative one. This can be done, for example, by labeling all the points from class, say ω_1 , with $y_n = 1$, $\forall n : \mathbf{x}_n \in \omega_1$, and all the points from class ω_2 with $y_n = -1$, $\forall n : \mathbf{x}_n \in \omega_2$. Then the LS loss is mobilized to compute $\boldsymbol{\theta}$ so as to minimize the cost

$$J(\boldsymbol{\theta}) = \sum_{n=1}^N \left(y_n - \boldsymbol{\theta}^T \mathbf{x}_n \right)^2.$$

The solution is exactly the same as Eq. (3.13). Figure 3.6 shows the resulting LS classifiers for two cases of data. Observe that in the case of Figure 3.6b, the resulting classifier cannot classify correctly all the data points. Our desire to place all the data, which originate from one class, on one side and the rest on the other cannot be satisfied. All that our LS classifier can do is to place the hyperplane so that the sum of squared errors, between the desired (true) values of the labels, y_n , and the predicted outputs, $\theta^T x_n$, are a minimum. It is mainly for cases such as overlapping classes, which are usually encountered in practice, where one has to look for an alternative to the LS criteria and methods, in order to serve better the needs and the goals of the classification task. For example, a reasonable optimality criterion would be to minimize the probability of error; that is, the percentage of points for which the true labels, y_n , and the predicted by the classifier ones, \hat{y}_n , are different. Chapter 7 presents methods and loss functions appropriate for the classification task. In Chapter 11, support vector machines are discussed and in Chapter 18, neural networks and deep learning methods are presented, which are currently among the most powerful techniques for classification problems.

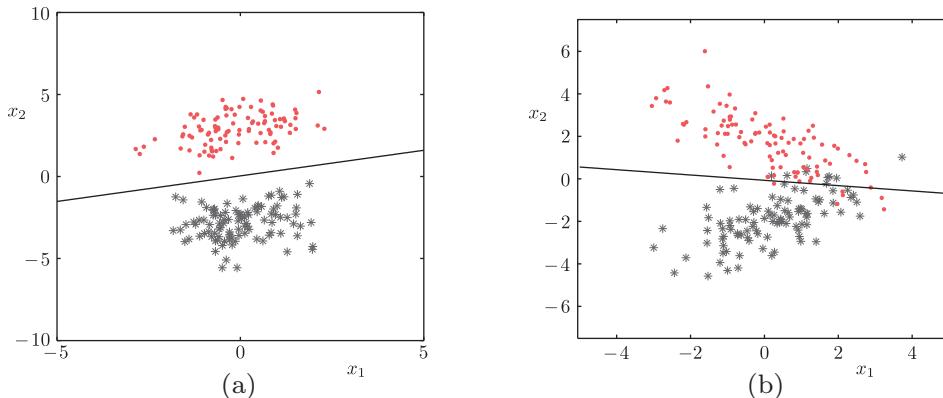


FIGURE 3.6

Design of a linear classifier, $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$, based on the LS loss function. (a) The case of two linearly separable classes and (b) the case of nonseparable classes. In the latter case, the classifier cannot separate fully the two classes. All it can do is to place the separating (decision) line so as to minimize the deviation between the true labels and the predicted output values in the LS sense.

Generative versus discriminative learning

The path that we have taken in order to introduce the classification task was to consider a functional dependence between the output variable (label), y , and the input variables (features), \mathbf{x} . The involved parameters were optimized with respect to a cost function. This path of modeling is also known as *discriminative learning*. We were not concerned with the statistical nature of the dependence that ties these two sets of variables together. In a more general setting, the term discriminative learning is also used to cover methods that model directly the posterior probability of a class, represented by its label y , given the feature vector \mathbf{x} , as in $P(y|\mathbf{x})$. The common characteristic of all these methods is that they bypass the need of modeling the input data distribution explicitly. From a statistical point of view, discriminative learning is justified as follows.

Using the product rule for probabilities, the joint distribution between the input data and their respective labels can be written as

$$p(y, \mathbf{x}) = P(y|\mathbf{x})p(\mathbf{x}).$$

In the discriminative learning, only the first of the two terms in the product is considered; a functional form is adopted and parameterized appropriately, as $P(y|\mathbf{x}; \boldsymbol{\theta})$. Parameters are then estimated via the use of a cost. The distribution of the input data is ignored. Such an approach has the advantage that simpler models can be used, especially if the input data are described by pdfs of a complex form. The disadvantage is that the input data distribution is ignored, although it can carry important information, which could be exploited to the benefit of the overall performance.

In contrast, the alternative path, known as *generative learning*, exploits the input data distribution. Once more, employing the product rule, we have

$$p(y, \mathbf{x}) = p(\mathbf{x}|y)P(y).$$

$P(y)$ is the probability concerning the classes and $p(\mathbf{x}|y)$ is the distribution of the input given the class label. For such an approach, we end up with one distribution per class, which has to be learned. In parametric modeling, a set of parameters is associated with each one of these conditional distributions. Once the joint distribution has been learned, the prediction of the class label of an unknown pattern, \mathbf{x} , is performed based on the a posteriori probability,

$$P(y|\mathbf{x}) = \frac{p(y, \mathbf{x})}{p(\mathbf{x})} = \frac{p(y, \mathbf{x})}{\sum_y p(y, \mathbf{x})}.$$

We will return to these issues in more detail in [Chapter 7](#).

Supervised, semisupervised, and unsupervised learning

The way both the regression as well as the classification tasks were introduced relied on a given set of training data. In other words, the values of the dependent variables, y , are known over the available training set of the regressors and feature vectors/patterns, respectively. For this reason, such tasks belong to the family of problems known as *supervised learning*. However, there are learning problems where the dependent variable is not known, or it may be known only for a small percentage of the available data. In such cases, we refer to *clustering* and *semisupervised learning*, respectively. In this book, our main concern is on the supervised learning. For the other types of learning, the interested reader can consult, for example, [\[7, 35\]](#).

3.5 BIASED VERSUS UNBIASED ESTIMATION

In supervised learning, we are given a set of training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, and we return an estimate of the unknown parameter vector, say $\hat{\theta}$. However, the training points themselves are random variables. If we are given another set of N observations of the *same* random variables, these are going to be different, and obviously the resulting estimate will also be different. In other words, by changing our training data different estimates result. Hence, we can assume that the resulting estimate, of a fixed yet unknown parameter, is itself a random variable. This, in turn, poses questions on how good an estimator is. No doubt, each time, the obtained estimate is optimal with respect to the adopted loss function *and* the specific training set used. However, who guarantees that the resulting estimates are “close” to the true value, assuming that there is one? In this section, we will try to address this task and to illuminate some related theoretical aspects. Note that we have already used the term *estimator* in place of the term *estimate*. Let us elaborate a bit on their difference, before presenting more details.

An estimate, such as $\hat{\theta}$, has a specific value, which is the result of a function acting on a set of observations, on which our chosen estimate depends (see Eq. [\(3.17\)](#)). In general, we could generalize Eq. [\(3.17\)](#) and write that

$$\hat{\theta} = f(\mathbf{y}, \mathbf{X}).$$

However, once we allow the set of observations to change randomly, and the estimate becomes itself a random variable, we write the previous equation in terms of the corresponding random variables,

$$\hat{\theta} = f(\mathbf{y}, \mathbf{X}),$$

and we refer to this functional dependence as the *estimator* of the unknown vector θ .

In order to simplify the analysis and focus on the insight behind the methods, we will assume that our parameter space is that of real numbers, \mathbb{R} . We will also assume that the model (i.e., the set of functions \mathcal{F}), which we have adopted for modeling our data, is the correct one and the (unknown to us) value of the associated true parameter is equal to² θ_o . Let $\hat{\theta}$ denote the random variable of the associated estimator. Adopting the squared error loss function to quantify deviations, a reasonable criterion to measure the performance of an estimator is the *mean-square error* (MSE),

$$\text{MSE} = \mathbb{E}[(\hat{\theta} - \theta_o)^2], \quad (3.19)$$

where the mean \mathbb{E} is taken over *all* possible training data sets of size N . If the MSE is small, then we expect that, on average, the resulting estimates to be close to the true value. However, this simple and “natural” looking criterion hides some interesting surprises for us. Let us insert the mean value $\mathbb{E}[\hat{\theta}]$ of $\hat{\theta}$ in Eq. (3.19) to get

$$\begin{aligned} \text{MSE} &= \mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right) + \left(\mathbb{E}[\hat{\theta}] - \theta_o\right)\right]^2 \\ &= \underbrace{\mathbb{E}\left[\left(\hat{\theta} - \mathbb{E}[\hat{\theta}]\right)^2\right]}_{\text{Variance}} + \underbrace{\left(\mathbb{E}[\hat{\theta}] - \theta_o\right)^2}_{\text{Bias}^2}, \end{aligned} \quad (3.20)$$

where, for the second equality, we have taken into account that the mean value of the product of the two involved terms turns out to be zero, as it is readily seen. What Eq. (3.20) suggests is that the MSE consists of two terms. The first one is the variance around the mean value and the second one is due to the bias; that is, the deviation of the mean value of the estimator from the true one.

3.5.1 BIASED OR UNBIASED ESTIMATION?

One may naively think that choosing an estimator that is *unbiased*, as is $\mathbb{E}[\hat{\theta}] = \theta_o$, such that the second term in Eq. (3.20) becomes zero, is a reasonable choice. Adopting an unbiased estimator may also be appealing from the following point of view. Assume that we have L different training sets, each comprising N points. Let us denote each data set by \mathcal{D}_i , $i = 1, 2, \dots, L$. For each one, an estimate $\hat{\theta}_i$, $i = 1, 2, \dots, L$, will result. Then, form the new estimator by taking the average value,

$$\hat{\theta}^{(L)} := \frac{1}{L} \sum_{i=1}^L \hat{\theta}_i.$$

This is also an unbiased estimator, because

$$\mathbb{E}[\hat{\theta}^{(L)}] = \frac{1}{L} \sum_{i=1}^L \mathbb{E}[\hat{\theta}_i] = \theta_o.$$

Moreover, assuming that the involved estimators are mutually uncorrelated,

$$\mathbb{E}[(\hat{\theta}_i - \theta_o)(\hat{\theta}_j - \theta_o)] = 0,$$

² Not to be confused with the intercept; the subscript here is “ o ” and not “0.”

and of the same variance, σ^2 , then the variance of the new estimator is now much smaller ([Problem 3.1](#)),

$$\sigma_{\hat{\theta}^{(L)}}^2 = \mathbb{E} \left[(\hat{\theta}^{(L)} - \theta_o)^2 \right] = \frac{\sigma^2}{L}.$$

Hence, by averaging a large number of such unbiased estimators, we expect to get an estimate close to the true value. However, in practice, data is a commodity that is not always abundant. As a matter of fact, very often the opposite is true and one has to be very careful about how to exploit it. In such cases, where one cannot afford to obtain and average a large number of estimators, an unbiased estimator may not necessarily be the best choice. Going back to Eq. [\(3.20\)](#), there is no reason to suggest that by making the second term equal to zero, the MSE (which, after all, is the quantity of interest to us) becomes minimum. Indeed, let us look at Eq. [\(3.20\)](#) from a slightly different view. Instead of computing the MSE for a given estimator, let us replace $\hat{\theta}$ with θ in Eq. [\(3.20\)](#) and compute an estimator that will minimize the MSE with respect to θ , directly. In this case, focusing on unbiased estimators, or $\mathbb{E}[\theta] = \theta_o$, introduces a constraint to the task of minimizing the MSE, and it is well-known that an unconstrained minimization problem always results in loss function values that are less than or equal to any value generated by a constrained counterpart,

$$\min_{\theta} \text{MSE}(\theta) \leq \min_{\theta: \mathbb{E}[\theta] = \theta_o} \text{MSE}(\theta), \quad (3.21)$$

where the dependence of MSE on the estimator θ in Eq. [\(3.21\)](#) is explicitly denoted.

Let us denote by $\hat{\theta}_{\text{MVU}}$ a solution of the task $\min_{\theta: \mathbb{E}[\theta] = \theta_o} \text{MSE}(\theta)$. It can be readily verified by Eq. [\(3.20\)](#) that $\hat{\theta}_{\text{MVU}}$ is an unbiased estimator of minimum variance. Such an estimator is known as the *minimum variance unbiased estimator* (MVU) and we assume that such an estimator exists. An MVU does not always exist ([\[20\], Problem 3.2](#)). Moreover, if it exists it is unique ([Problem 3.3](#)). Motivated by Eq. [\(3.21\)](#), our next goal is to search for a *biased* estimator, which results, hopefully, in a smaller MSE. Let us denote this estimator as $\hat{\theta}_b$. For the sake of illustration, and in order to limit our search for $\hat{\theta}_b$, we consider here only $\hat{\theta}_b$ s that are scalar multiples of $\hat{\theta}_{\text{MVU}}$, so that

$$\hat{\theta}_b = (1 + \alpha)\hat{\theta}_{\text{MVU}}, \quad (3.22)$$

where $\alpha \in \mathbb{R}$ is a free parameter. Notice that $\mathbb{E}[\hat{\theta}_b] = (1 + \alpha)\theta_o$. By substituting Eq. [\(3.22\)](#) into Eq. [\(3.20\)](#) and after some simple algebra we obtain

$$\text{MSE}(\hat{\theta}_b) = (1 + \alpha)^2 \text{MSE}(\hat{\theta}_{\text{MVU}}) + \alpha^2 \theta_o^2. \quad (3.23)$$

In order to get $\text{MSE}(\hat{\theta}_b) < \text{MSE}(\hat{\theta}_{\text{MVU}})$, α must be in the range ([Problem 3.4](#))

$$-\frac{2\text{MSE}(\hat{\theta}_{\text{MVU}})}{\text{MSE}(\hat{\theta}_{\text{MVU}}) + \theta_o^2} < \alpha < 0. \quad (3.24)$$

It is easy to verify that the previous range implies that $|1 + \alpha| < 1$. Hence, $|\hat{\theta}_b| = |(1 + \alpha)\hat{\theta}_{\text{MVU}}| < |\hat{\theta}_{\text{MVU}}|$. We can go a step further and try to compute the optimum value of α , which corresponds to the minimum MSE. By taking the derivative of $\text{MSE}(\hat{\theta}_b)$ in Eq. [\(3.23\)](#) with respect to α , it turns out ([Problem 3.5](#)) that this occurs for

$$\alpha_* = -\frac{\text{MSE}(\hat{\theta}_{\text{MVU}})}{\text{MSE}(\hat{\theta}_{\text{MVU}}) + \theta_o^2} = -\frac{1}{1 + \frac{\theta_o^2}{\text{MSE}(\hat{\theta}_{\text{MVU}})}}. \quad (3.25)$$

Therefore, we have found a way to obtain the optimum estimator, among those in the set $\{\hat{\theta}_b = (1 + \alpha)\hat{\theta}_{\text{MVU}} : \alpha \in \mathbb{R}\}$, which results in minimum MSE. This is true, but as many nice things in life, this is not, in general, realizable. The optimal value for α is given in terms of the unknown, θ_o ! However, Eq. (3.25) is useful in a number of other ways. First, there are cases where the MSE is proportional to θ_o^2 ; hence, this formula can be used. Also, for certain cases, it can be used to provide useful bounds [19]. Moreover, as far as we are concerned in this book, it says something very important. If we want to do better than the MVU, then, looking at the text after Eq. (3.24), a possible way is to *shrink* the norm of the MVU estimator. Shrinking the norm is a way of introducing bias into an estimator. We will discuss ways to achieve this in [Section 3.8](#) and later on in [Chapters 6 and 11](#).

Note that what we have said so far is readily generalized to parameter vectors. An unbiased parameter vector satisfies

$$\mathbb{E}[\boldsymbol{\theta}] = \boldsymbol{\theta}_o,$$

and the MSE around the true value, $\boldsymbol{\theta}_o$, is defined as

$$\text{MSE} = \mathbb{E}[(\boldsymbol{\theta} - \boldsymbol{\theta}_o)^T(\boldsymbol{\theta} - \boldsymbol{\theta}_o)].$$

Looking carefully at the previous definition reveals that the MSE for a parameter vector is the sum of the MSEs of the components, θ_i , $i = 1, 2, \dots, l$, around the corresponding true values θ_{oi} .

3.6 THE CRAMÉR-RAO LOWER BOUND

In the previous sections, we saw how one can improve upon the performance of the MVU estimator, provided that this exists and it is also known. However, how can one know that an unbiased estimator, that has been obtained, is also of minimum variance? The goal of this section is to introduce a criterion that can provide such information.

The *Cramér-Rao lower bound* [8, 31] is an elegant theorem and one of the most well-known techniques used in statistics. It provides a lower bound on the variance of *any* unbiased estimator. This is very important because (a) it offers the means to assert whether an unbiased estimator has minimum variance, which, of course, in this case coincides with the corresponding MSE in Eq. (3.20), and (b) if this is not the case, it can be used to indicate how far away the performance of an unbiased estimator is from the optimal one, and finally (c) it provides the designer with a tool to know the best possible performance that can be achieved by an unbiased estimator. Because our main purpose here is to focus on the insight and physical interpretation of the method, we will deal with the simple case where our unknown parameter is a real number. The general form of the theorem, involving vectors, is given in [Appendix B](#).

We are looking for a bound of the variance of an unbiased estimator, whose randomness is due to the randomness of the training data, as we change from one set to another. Thus, it does not come as a surprise that the bound involves the joint pdf of the data, parameterized in terms of the unknown parameter, $\boldsymbol{\theta}$. Let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ denote the set of N observations, corresponding to a random vector,³ \mathbf{x} , that depends on the unknown parameter. Also, let the respective joint pdf of the observations be denoted as $p(\mathcal{X}; \boldsymbol{\theta})$.

³ Note, here, that \mathbf{x} is treated as a random quantity in a general setting, and not necessarily in the context of the regression/classification tasks.

Theorem 3.1. *It is assumed that the joint pdf satisfies the following regularity condition:*

$$\mathbb{E} \left[\frac{\partial \ln p(\mathcal{X}; \theta)}{\partial \theta} \right] = 0, \quad \forall \theta. \quad (3.26)$$

This regularity condition is a weak one and holds for most of the cases in practice (Problem 3.6). Then, the variance of any unbiased estimator, $\hat{\theta}$, must satisfy the following inequality:

$$\sigma_{\hat{\theta}}^2 \geq \frac{1}{I(\theta)} : \quad \text{Cramér-Rao Lower Bound},$$

(3.27)

where

$$I(\theta) := -\mathbb{E} \left[\frac{\partial^2 \ln p(\mathcal{X}; \theta)}{\partial \theta^2} \right]. \quad (3.28)$$

Moreover, the necessary and sufficient condition for obtaining an unbiased estimator that achieves the bound is the existence of a function $g(\cdot)$ such that for all possible values of θ ,

$$\frac{\partial \ln p(\mathcal{X}; \theta)}{\partial \theta} = I(\theta) (g(\mathcal{X}) - \theta). \quad (3.29)$$

The MVU estimate is then given by

$$\hat{\theta} = g(\mathcal{X}) := g(x_1, x_2, \dots, x_N), \quad (3.30)$$

and the variance of the respective estimator is equal to $1/I(\theta)$.

When an MVU estimator attains the Cramér-Rao bound, we say that it is *efficient*. All the expectations before are taken with respect to $p(\mathcal{X}; \theta)$. The interested reader may find more on the topic in more specialized books on statistics [20, 27, 34].

Example 3.2. Let us consider the simplified version of the linear regression model in Eq. (3.10), where the regressor is real valued and the bias term is zero,

$$y_n = \theta x + \eta_n, \quad (3.31)$$

where we have explicitly denoted the dependence on n , which runs over the number of available observations. Note that in order to further simplify the discussion, we have assumed that our N observations are the result of different realizations of the noise variable only, and that we have kept the value of the input, x , constant, which can be considered to be equal to one, without harming generality; that is, our task degenerates to that of estimating a parameter from its noisy measurements. Thus, for this case, the observations are the scalar outputs, y_n , $n = 1, 2, \dots, N$, which we consider to be the components of a vector, $\mathbf{y} \in \mathbb{R}^N$. We further assume that η_n are samples of a Gaussian white noise with zero mean and variance equal to σ_η^2 . Then, the joint pdf of the output observations is given by

$$p(\mathbf{y}; \theta) = \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma_\eta^2}} \exp \left(-\frac{(y_n - \theta)^2}{2\sigma_\eta^2} \right), \quad (3.32)$$

or

$$\ln p(\mathbf{y}; \theta) = -\frac{N}{2} \ln(2\pi\sigma_\eta^2) - \frac{1}{2\sigma_\eta^2} \sum_{n=1}^N (y_n - \theta)^2. \quad (3.33)$$

We will derive the corresponding Cramér-Rao bound. Taking the derivative of the logarithm with respect to θ we have

$$\frac{\partial \ln p(\mathbf{y}; \theta)}{\partial \theta} = \frac{1}{\sigma_\eta^2} \sum_{n=1}^N (y_n - \theta) = \frac{N}{\sigma_\eta^2} (\bar{y} - \theta), \quad (3.34)$$

where

$$\bar{y} := \frac{1}{N} \sum_{n=1}^N y_n,$$

that is, the sample mean of the measurements. The second derivative, as required by the theorem, is given by

$$\frac{\partial^2 \ln p(\mathbf{y}; \theta)}{\partial \theta^2} = -\frac{N}{\sigma_\eta^2},$$

and hence,

$$I(\theta) = \frac{N}{\sigma_\eta^2}. \quad (3.35)$$

Equation (3.34) is in the form of Eq. (3.29), with $g(\mathbf{y}) = \bar{y}$; thus, an efficient estimator can be obtained and the lower bound of the variance of any unbiased estimator, for our data model of Eq. (3.31), is

$$\sigma_{\hat{\theta}}^2 \geq \frac{\sigma_\eta^2}{N}. \quad (3.36)$$

We can easily verify that the corresponding estimator, \bar{y} is indeed an unbiased one under the adopted model of Eq. (3.31),

$$\mathbb{E}[\bar{y}] = \frac{1}{N} \sum_{n=1}^N \mathbb{E}[y_n] = \frac{1}{N} \sum_{n=1}^N \mathbb{E}[\theta + \eta_n] = \theta.$$

Moreover, the previous formula, combined with Eq. (3.34), also establishes the regularity condition, as it is required by the Cramér-Rao theorem.

The bound in Eq. (3.36) is a very natural result. The Cramér-Rao lower bound depends on the variance of the noise source. The higher this is, and therefore the higher the uncertainty of each measurement with respect to the value of the true parameter is, the higher the minimum variance of an estimator is expected to be. On the other hand, as the number of observations increases and more “information” is disclosed to us, the uncertainty decreases and we expect the variance of our estimator to decrease.

Having obtained the lower bound for our task, let us return our attention to the LS estimator for the specific regression model of Eq. (3.31). This results from Eq. (3.13), by setting $x_n = 1$ and a

simple inspection shows that the LS estimate is nothing but the sample mean, \bar{y} , of the observations. Furthermore, the variance of the corresponding estimator is given by

$$\begin{aligned}\sigma_{\bar{y}}^2 &= \mathbb{E}[(\bar{y} - \theta)^2] = \mathbb{E}\left[\frac{1}{N^2}\left(\sum_{n=1}^N(y_n - \theta)\right)^2\right] \\ &= \frac{1}{N^2}\mathbb{E}\left[\left(\sum_{n=1}^N\eta_n\right)^2\right] = \frac{1}{N^2}\mathbb{E}\left[\sum_{i=1}^N\eta_i\sum_{j=1}^N\eta_j\right] \\ &= \frac{1}{N^2}\sum_{i=1}^N\sum_{j=1}^N\mathbb{E}[\eta_i\eta_j] = \frac{\sigma_{\eta}^2}{N},\end{aligned}$$

which coincides with our previous finding via the use of the Cramér-Rao theorem. In other words, for this particular task and having assumed that the noise is Gaussian, the LS estimator \bar{y} is an MVU estimator and it attains the Cramér-Rao bound. However, if the input is not fixed, but it also varies from experiment to experiment and the training data become (y_n, x_n) , then the LS estimator attains the Cramér-Rao bound only asymptotically, for large values of N ([Problem 3.7](#)). Moreover, it has to be pointed out that if the assumptions for the noise being Gaussian *and* white are not valid, then the LS estimator is not efficient anymore.

It turns out that this result, which has been obtained for the real axis case, is also true for the general regression model given in Eq. (3.10) ([Problem 3.8](#)). We will return to the properties of the LS estimator in more detail in [Chapter 6](#).

Remarks 3.2.

- The Cramér-Rao bound is not the only one that is available in the literature. For example, the Bhattacharya bound makes use of higher order derivatives of the pdf. It turns out that in cases where an efficient estimator does not exist, then the Bhattacharya bound is tighter compared to the Cramér-Rao one, with respect to the variance of the MVU estimator [27]. Other bounds also exist [21]; however, the Cramér-Rao bound is the easiest to determine.

3.7 SUFFICIENT STATISTIC

If an efficient estimator does not exist, this does not necessarily mean that the MVU estimator cannot be determined. It may exist, but it will not be an efficient one, in the sense that it does not satisfy the Cramér-Rao bound. In such cases, the notion of *sufficient statistic* and the Rao-Blackwell theorem come into the picture.⁴ Although these are beyond the focus of this book, they are mentioned here in order to provide a more complete picture of the topic.

The notion of sufficient statistic is due to Sir Ronald Aylmer Fisher (1890-1962). Fisher was an English statistician and biologist who made a number of fundamental contributions that laid out many of the foundations of modern statistics. Besides statistics, he made important contributions in genetics.

⁴ It must be pointed out that the use of sufficient statistic in statistics extends much beyond the search for MVUs.

In short, given a random vector, \mathbf{x} , which depends on a parameter θ , a sufficient statistic for the unknown parameter is a function

$$T(\mathcal{X}) := T(x_1, x_2, \dots, x_N),$$

of the respective observations, which contains *all* information about θ . From a mathematical point of view, a statistic $T(\mathcal{X})$ is said to be sufficient for the parameter θ if the conditional joint pdf

$$p(\mathcal{X}|T(\mathcal{X});\theta),$$

does not depend on θ . In such a case, it becomes apparent that $T(\mathcal{X})$ must provide *all* information about θ , which is contained in the set \mathcal{X} . Once $T(\mathcal{X})$ is known, \mathcal{X} is no longer needed, because no further information can be extracted from it; this justifies the name of “sufficient statistic.” The concept of sufficient statistic is also generalized to parameter vectors $\boldsymbol{\theta}$. In such a case, the sufficient statistic may be a *set* of functions, called a *jointly sufficient statistic*. Typically, there are as many functions as there are parameters; in a slight abuse of notation, we will still write $T(\mathcal{X})$ to denote this set (vector of) functions.

A very important theorem, which facilitates the search for a sufficient statistic in practice, is the following [27].

Theorem 3.2 (Factorization Theorem). *A statistic $T(\mathcal{X})$ is sufficient if and only if the respective joint pdf can be factored as*

$$p(\mathcal{X};\theta) = h(\mathcal{X})g(T(\mathcal{X}),\theta).$$

That is, the joint pdf is factored into two parts: one part that depends only on the statistic and the parameters and a second part that is independent of the parameters. The theorem is also known as the Fisher-Neyman factorization theorem.

Once a sufficient statistic has been found and under certain conditions related to the statistic, the Rao-Blackwell theorem determines the MVU estimator (MVUE) by taking the expectation conditioned on $T(\mathcal{X})$. A by-product of this theorem is that if an unbiased estimator is expressed *solely* in terms of the sufficient statistic, then it is necessarily the unique MVUE [23]. The interested reader can obtain more on these issues from [20, 21, 27].

Example 3.3. Let x be a Gaussian, $\mathcal{N}(\mu, \sigma^2)$, random variable and let the set of observations be $\mathcal{X} = \{x_1, x_2, \dots, x_N\}$. Assume μ to be the unknown parameter. Show that

$$S_\mu = \frac{1}{N} \sum_{n=1}^N x_n,$$

is a sufficient statistic for the parameter μ .

The joint pdf is given by

$$p(\mathcal{X};\mu) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}} \exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2\right).$$

Plugging the obvious identity,

$$\sum_{n=1}^N (x_n - \mu)^2 = \sum_{n=1}^N (x_n - S_\mu)^2 + N(S_\mu - \mu)^2,$$

into the joint pdf, we obtain

$$p(\mathcal{X}; \mu) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}} \exp\left(-\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - S_\mu)^2\right) \exp\left(-\frac{N}{2\sigma^2} (S_\mu - \mu)^2\right),$$

which, according to the factorization theorem, proves the claim.

In a similar way, one can prove (Problem 3.9) that if the unknown parameter is the variance σ^2 , then $\bar{S}_{\sigma^2} := \frac{1}{N} \sum_{n=1}^N (x_n - \mu)^2$ is a sufficient statistic, and if both μ and σ^2 are unknown then a sufficient statistic is the set (S_μ, S_{σ^2}) , where

$$S_{\sigma^2} = \frac{1}{N} \sum_{n=1}^N (x_n - S_\mu)^2.$$

That is, in this case, all information concerning the unknown set of parameters that can be possibly extracted from the available N observations, can be fully recovered by considering only the sum of the observations and the sum of their squares.

3.8 REGULARIZATION

We have already seen that the LS estimator is a minimum variance unbiased estimator, under the assumptions of linearity of the regression model and in the presence of a Gaussian white noise source. We also know that one can improve the performance by shrinking the norm of the MVU estimator. There are additional ways to achieve this goal and they will be discussed later on in this book. In this section, we focus on one possibility. Moreover, we will see that trying to keep the norm of the solution small serves important needs in the context of machine learning.

Regularization is a mathematical tool to impose a priori information on the structure of the solution, which comes as the outcome of an optimization task. Regularization was first suggested by the great Russian mathematician Andrey Nikolayevich Tychonoff (sometimes spelled Tikhonov) for the solution of integral equations. Sometimes, it is also referred as Tychonoff-Phillips regularization, to honor David Phillips as well, who developed the method independently [29, 37].

In the context of our task and in order to shrink the norm of the parameter vector estimate, we can reformulate the LS minimization task, given in Eq. (3.12), as

$$\text{minimize: } J(\boldsymbol{\theta}) = \sum_{n=1}^N (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2, \quad (3.37)$$

$$\text{subject to: } \|\boldsymbol{\theta}\|^2 \leq \rho, \quad (3.38)$$

where $\|\cdot\|$ stands for the Euclidean norm of a vector. In this way, we do not allow the LS criterion to be completely “free” to reach a solution, but we *limit* the space in which to search for it. Obviously, using different values of ρ , we can achieve different levels of shrinkage. As we have already discussed, the optimal value of ρ cannot be analytically obtained, and one has to experiment in order to select an estimator that results in a good performance. For the LS loss function and the constraint used before, the optimization task can equivalently be written as [5, 6]

$$\text{minimize: } L(\boldsymbol{\theta}, \lambda) = \sum_{n=1}^N \left(y_n - \boldsymbol{\theta}^T \mathbf{x}_n \right)^2 + \lambda \|\boldsymbol{\theta}\|^2 : \text{ Ridge Regression.} \quad (3.39)$$

It turns out that, for specific choices of $\lambda \geq 0$ and ρ , the two tasks are equivalent. Note that this new cost function, $L(\boldsymbol{\theta}, \lambda)$, involves one term that measures the model *misfit* and a second one that quantifies the *size* of the norm of the parameter vector. It is straightforward to see that taking the gradient of L in Eq. (3.39) with respect to $\boldsymbol{\theta}$ and equating to zero, we obtain the *regularized LS* solution for the linear regression task of Eq. (3.13)

$$\left(\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T + \lambda I \right) \hat{\boldsymbol{\theta}} = \sum_{n=1}^N y_n \mathbf{x}_n, \quad (3.40)$$

where I is the identity matrix of appropriate dimensions. The presence of λ biases the new solution away from that which would have been obtained from the unregularized LS formulation. The task is also known as *ridge regression*. Ridge regression attempts to reduce the norm of the estimated vector and *at the same time* tries to keep the sum of squared errors small; in order to achieve this *combined* goal, the vector components, θ_i , are modified in such a way so that the contribution in the misfit measuring term, from the less informative directions in the input space, is minimized. We will return to this in more detail in Chapter 6. Ridge regression was first introduced in [18].

It has to be emphasized that in practice, the bias parameter, θ_0 , is left out from the norm in the regularization term; penalization of the bias would make the procedure dependent on the origin chosen for y . Indeed, it is easily checked out that adding a constant term to each one of the output values, y_n , in the cost function, would not result in just a shift of the predictions by the same constant, if the bias term is included in the norm. Hence, usually, ridge regression is formulated as

$$\text{minimize } L(\boldsymbol{\theta}, \lambda) = \sum_{n=1}^N \left(y_n - \theta_0 - \sum_{i=1}^l \theta_i x_{ni} \right)^2 + \lambda \sum_{i=1}^l |\theta_i|^2. \quad (3.41)$$

It turns out (Problem 3.10) that minimizing Eq. (3.41) with respect to θ_i , $i = 1, 2, \dots, l$, is equivalent with minimizing Eq. (3.39) using *centered* data and neglecting the intercept. That is, one solves the task

$$\text{minimize } L(\boldsymbol{\theta}, \lambda) = \sum_{n=1}^N \left((y_n - \bar{y}) - \sum_{i=1}^l \theta_i (x_{ni} - \bar{x}_i) \right)^2 + \lambda \sum_{i=1}^l |\theta_i|^2, \quad (3.42)$$

and the estimate of θ_0 in Eq. (3.41) is given in terms of the obtained estimates, $\hat{\theta}_i$,

$$\hat{\theta}_0 = \bar{y} - \sum_{i=1}^l \hat{\theta}_i \bar{x}_i,$$

where

$$\bar{y} = \frac{1}{N} \sum_{n=1}^N y_n \quad \text{and} \quad \bar{x}_i = \frac{1}{N} \sum_{n=1}^N x_{ni}, \quad i = 1, 2, \dots, l.$$

In other words, $\hat{\theta}_0$ compensates for the differences between the sample means of the output and input variables. Note that similar arguments hold true if the Euclidean norm, used in Eq. (3.40) as a regularizer, is replaced by other norms, such as the ℓ_1 or in general ℓ_p , $p > 1$ norms, Chapter 9.

From a different viewpoint, reducing the norm can be considered as an attempt to “simplify” the structure of the estimator, because a smaller number of components of the regressor now have an important say. This viewpoint becomes more clear if one considers nonlinear models, as was discussed in Section 3.2. In this case, the existence of the norm of the respective parameter vector in Eq. (3.39) forces the model to get rid of the less important terms in the nonlinear expansion, $\sum_{k=1}^K \theta_k \phi_k(\mathbf{x})$, and effectively pushes K to lower values.

Although in the current context, the complexity issue emerges in a rather disguised form, one can make it a major player in the game by choosing to use different functions and norms for the regularization term; and there are many reasons that justify such choices.

Inverse problems: Ill-conditioning and overfitting

Most tasks in machine learning belong to the so-called *inverse problems*. The latter term encompasses all the problems where one has to infer/predict/estimate the values of a model based on a set of available output/input observations-training data. In a less mathematical terminology, in inverse problems one has to unravel unknown causes from known effects; in other words, to reverse the cause-effect relations. Inverse problems are typically *ill-posed*, as opposed to the *well-posed* ones. Well-posed problems are characterized by (a) the existence of a solution, (b) the uniqueness of the solution, and (c) the stability of the solution. The latter condition is usually violated in machine learning problems. This means that the obtained solution may be very sensitive to changes of the training set. *Ill conditioning* is another term used to describe this sensitivity. The reason for this behavior is that the model used to describe the data can be complex, in the sense that the number of the unknown free parameters is large with respect to the number of data points. The “face” with which this problem manifests itself in machine learning is known as *overfitting*. This means that during training, the estimated parameters of the unknown model learn too much about the idiosyncrasies of the specific training data set, and the model performs badly when it deals with another set of data, other than that used for the training. As a matter of fact, the MSE criterion discussed in Section 3.5 attempts to quantify exactly this way; that is, the mean deviation of the obtained estimates from the true value by changing the training sets.

When the number of training samples is small with respect to the number of the unknown parameters, the available information is not enough to “reveal” a sufficiently good model that fits the data, and it can be misleading due to the presence of the noise and possible outliers. Regularization is an elegant and efficient tool to cope with the complexity of the model; that is, to make it less complex, more smooth. There are different ways to achieve this. One way is by constraining the norm of the unknown vector, as ridge regression does. When dealing with more complex, compared to linear, models, one can use constraints on the smoothness of the involved nonlinear function; for example, by involving derivatives of the model function in the regularization term. Also, regularization can help when the adopted model and the number of training points are such that no solution is possible. For example, in our LS linear regression task of Eq. (3.13), if the number, N , of the training points is less than the dimension of the regressors \mathbf{x}_n , then the $l \times l$ matrix, $\bar{\Sigma} = \sum_n \mathbf{x}_n \mathbf{x}_n^T$, is not invertible. Indeed, each term in the summation is the outer product of a vector with itself and hence it is a matrix of rank one.

Thus, as we know from linear algebra, we need at least l linearly independent terms of such matrices to guarantee that the sum is of full rank, hence invertible. However, in ridge regression, this can be bypassed, because the presence of λI in Eq. (3.40) guarantees that the left-hand matrix is invertible. Furthermore, the presence of λI can also help when Σ is invertible but it is ill-conditioned. Usually in such cases, the resulting LS solution has a very large norm and, thus, it is meaningless. Regularization helps to replace the original ill-conditioned problem with a “nearby” one, which is well-conditioned and whose solution approximates the target one.

Another example where regularization can help to obtain a solution, and, more important, a unique solution to an otherwise unsolvable problem, is when the model’s order is large compared to the number of data, albeit we know that it is sparse. That is, only a very small percentage of the model’s parameters are nonzero. For such a task, a standard LS linear regression approach has no solution. However, regularizing the LS loss function using the ℓ_1 norm of the parameters’ vector can lead to a unique solution; the ℓ_1 norm of a vector comprises the sum of the absolute values of its components. This problem will be considered in [Chapters 9 and 10](#).

Regularization is closely related to the task of using priors in Bayesian learning, as we will discuss in [Section 3.11](#). Finally, note that regularization is not a panacea for facing the problem of overfitting. As a matter of fact, selecting the right set of functions \mathcal{F} in Eq. (3.3) is the first crucial step. The issue of the complexity of an estimator and the consequences on its “average” performance, as this is measured over all possible data sets, is discussed in [Section 3.9](#).

Example 3.4. The goal of this example is to demonstrate that the estimator obtained via the ridge regression can score a better MSE performance compared to the unconstrained LS solution. Let us consider, once again, the model exposed in [Example 3.2](#), and assume that the data are generated according to

$$y_n = \theta_o + \eta_n, \quad n = 1, 2, \dots, N,$$

where, for simplicity, we have assumed that the regressors $x_n \equiv 1$, and η_n , $n = 1, 2, \dots, N$, are i.i.d. zero-mean Gaussian noise samples of variance σ_η^2 .

We have already seen in [Example 3.2](#) that the solution to the LS parameter estimation task is the sample mean $\hat{\theta}_{\text{MVU}} = \frac{1}{N} \sum_{n=1}^N y_n$. We have shown also that this solution scores an MSE of σ_η^2/N and under the Gaussian assumption for the noise it achieves the Cramér-Rao bound. The question now is whether a biased estimator, $\hat{\theta}_b$, which corresponds to the solution of the associated ridge regression task, can achieve an MSE lower than $\text{MSE}(\hat{\theta}_{\text{MVU}})$.

It can be readily verified that Eq. (3.40), adapted to the needs of the current linear regression scenario, results in

$$\hat{\theta}_b(\lambda) = \frac{1}{N + \lambda} \sum_{n=1}^N y_n = \frac{N}{N + \lambda} \hat{\theta}_{\text{MVU}},$$

where we have explicitly expressed the dependence of the estimate $\hat{\theta}_b$ on the regularization parameter λ . Notice that for the associated estimator we have, $\mathbb{E}[\hat{\theta}_b(\lambda)] = \frac{N}{N + \lambda} \theta_o$.

A simple inspection of the previous relation takes us back to the discussion related to Eq. (3.22). Indeed, by following a sequence of similar steps to [Section 3.5.1](#), one can verify (see [Problem 3.11](#)) that the minimum value of $\text{MSE}(\hat{\theta}_b)$ is

$$\text{MSE}(\hat{\theta}_b(\lambda_*)) = \frac{\frac{\sigma_\eta^2}{N}}{1 + \frac{\sigma_\eta^2}{N\theta_o^2}} < \frac{\sigma_\eta^2}{N} = \text{MSE}(\hat{\theta}_{\text{MVU}}), \quad (3.43)$$

attained at $\lambda_* = \sigma_\eta^2/\theta_o^2$. The answer to the question whether the ridge regression estimate offers an improvement to the MSE performance is therefore positive in the current context. As a matter of fact, there *always* exists a $\lambda > 0$ such that the ridge regression estimate, which solves the general task of Eq. (3.39), achieves an MSE lower than the one corresponding to the MVU estimate [4, Section 8.4].

We will now demonstrate the previous theoretical findings via some simulations. To this end, the true value of the model was chosen to be $\theta_o = 10^{-2}$. The noise was Gaussian of zero mean value and variance $\sigma_\eta^2 = 0.1$. The number of generated samples was $N = 100$. Note that this is quite large, compared to a single parameter we have to estimate. The previous values imply that $\theta_o^2 < \sigma_\eta^2/N$. Then, it can be shown that, for any value of $\lambda > 0$, we can obtain a value for $\text{MSE}(\hat{\theta}_b(\lambda))$, which is smaller than that of $\text{MSE}(\hat{\theta}_{\text{MVU}})$ (see Problem 3.11). This is verified by the values shown in Table 3.1. To compute the MSE values in the table, the expectation operation in the definition in Eq. (3.19) was approximated by the respective sample mean. To this end, the experiment was repeated L times and the MSE was computed as

$$\text{MSE} \approx \frac{1}{L} \sum_{i=1}^L (\hat{\theta}_i - \theta_o)^2.$$

To get accurate results, we perform $L = 10^6$ trials. The corresponding MSE value for the unconstrained LS task is equal to $\text{MSE}(\hat{\theta}_{\text{MVU}}) = 1.00108 \times 10^{-3}$. Observe that substantial improvements can be attained when using regularization, in spite of the relatively large number of training data.

However, the percentage of performance improvement depends heavily on the specific values that define the model, as Eq. (3.43) suggests. For example, if $\theta_o = 0.1$, the obtained values from the experiments were $\text{MSE}(\hat{\theta}_{\text{MVU}}) = 1.00061 \times 10^{-3}$ and $\text{MSE}(\hat{\theta}_b(\lambda_*)) = 9.99578 \times 10^{-4}$. The theoretical ones, as computed from Eq. (3.43), are 1×10^{-3} and 9.99001×10^{-4} , respectively. The improvement obtained by using the ridge regression is now rather insignificant.

Table 3.1 Attained Values of MSE for Ridge Regression and Different Values of the Regularization Parameter

λ	$\text{MSE}(\hat{\theta}_b(\lambda))$
0.1	9.99082×10^{-4}
1.0	9.79790×10^{-4}
100.0	2.74811×10^{-4}
$\lambda_* = 10^3$	9.09671×10^{-5}

The attained MSE for the unconstrained LS estimate was $\text{MSE}(\hat{\theta}_{\text{MVU}}) = 1.00108 \times 10^{-3}$.

3.9 THE BIAS-VARIANCE DILEMMA

This section goes one step beyond [Section 3.5](#). There, the MSE criterion was used to quantify the performance with respect to the unknown parameter. Such a setting was useful, in order to help us understand some trends and also better digest the notions of “biased” versus “unbiased” estimation. Here, although the criterion will be the same, it will be used in a more general setting. To this end, we shift our interest from the unknown parameter to the dependent variable and our goal becomes to obtain an estimator of the value y , given a measurement of the regressor vector, $\mathbf{x} = \mathbf{x}$. Let us first consider the more general form of regression,

$$y = g(\mathbf{x}) + \eta, \quad (3.44)$$

where, once more, we have assumed that the dependent variable takes values in the real axis, $y \in \mathbb{R}$, for simplicity and without harm of the generality. The first question to be addressed is whether there exists an estimator that guarantees minimum MSE performance.

3.9.1 MEAN-SQUARE ERROR ESTIMATION

Our goal is to obtain an estimate $\hat{g}(\mathbf{x})$ of the unknown (nonlinear in general) function $g(\mathbf{x})$. This problem can be cast in the context of the more general estimation task setting.

Let the *jointly distributed* random variables, y, \mathbf{x} . Then, given a set of observations, $\mathbf{x} = \mathbf{x} \in \mathbb{R}^l$, the task is to obtain a function $\hat{y} := \hat{g}(\mathbf{x}) \in \mathbb{R}$, such that

$$\hat{g}(\mathbf{x}) = \arg \min_{f: \mathbb{R}^l \rightarrow \mathbb{R}} \mathbb{E}[(y - f(\mathbf{x}))^2], \quad (3.45)$$

where the expectation is taken with respect to the conditional probability of y given the value of \mathbf{x} ; in other words, $p(y|\mathbf{x})$.

We will show that the optimal estimate is the mean value of y , or

$$\hat{g}(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}] := \int_{-\infty}^{+\infty} y p(y|\mathbf{x}) dy : \text{Optimal MSE Estimate.}$$

(3.46)

Proof. We have that

$$\begin{aligned} \mathbb{E}[(y - f(\mathbf{x}))^2] &= \mathbb{E}[(y - \mathbb{E}[y|\mathbf{x}] + \mathbb{E}[y|\mathbf{x}] - f(\mathbf{x}))^2] \\ &= \mathbb{E}[(y - \mathbb{E}[y|\mathbf{x}])^2] + \mathbb{E}[(\mathbb{E}[y|\mathbf{x}] - f(\mathbf{x}))^2] \\ &\quad + 2 \mathbb{E}[(y - \mathbb{E}[y|\mathbf{x}])(\mathbb{E}[y|\mathbf{x}] - f(\mathbf{x}))], \end{aligned}$$

where the dependence of the expectation on \mathbf{x} has been suppressed for notational convenience. It is readily seen that the last (product) term on the right-hand side is zero, hence, we are left with the following:

$$\mathbb{E}[(y - f(\mathbf{x}))^2] = \mathbb{E}[(y - \mathbb{E}[y|\mathbf{x}])^2] + (\mathbb{E}[y|\mathbf{x}] - f(\mathbf{x}))^2, \quad (3.47)$$

where we have taken into account that, for fixed \mathbf{x} , the terms $\mathbb{E}[y|\mathbf{x}]$ and $f(\mathbf{x})$ are not random variables. From Eq. (3.47) we finally obtain our claim,

$$\mathbb{E}[(y - f(\mathbf{x}))^2] \geq \mathbb{E}[(y - \mathbb{E}[y|\mathbf{x}])^2]. \quad (3.48)$$

□

Note that this is a very elegant result. The optimal estimate, in the MSE sense, of the unknown function is given as $\hat{g}(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}]$. Sometimes, the latter is also known as the *regression of y conditioned on $\mathbf{x} = \mathbf{x}$* . This is, in general, a nonlinear function. It can be shown that if (\mathbf{y}, \mathbf{x}) take values in $\mathbb{R} \times \mathbb{R}^l$ and are jointly Gaussian, then the optimal MSE estimator $\mathbb{E}[y|\mathbf{x}]$ is a linear (affine) function of \mathbf{x} .

The previous results generalize to the case where \mathbf{y} is a random vector that takes values in \mathbb{R}^k . The optimal MSE estimate, given the values of $\mathbf{x} = \mathbf{x}$, is equal to

$$\hat{g}(\mathbf{x}) = \mathbb{E}[y|\mathbf{x}],$$

where now $\hat{g}(\mathbf{x}) \in \mathbb{R}^k$ (Problem 3.13). Moreover, if (\mathbf{y}, \mathbf{x}) are jointly Gaussian random vectors, the MSE optimal estimate is also an affine function of \mathbf{x} (Problem 3.14).

The findings of this subsection can be fully justified by physical reasoning. Assume, for simplicity, that the noise source in Eq. (3.44) is of zero mean. Then, for a fixed value $\mathbf{x} = \mathbf{x}$, we have that $\mathbb{E}[y|\mathbf{x}] = g(\mathbf{x})$ and the respective MSE is equal to

$$\text{MSE} = \mathbb{E}[(y - \mathbb{E}[y|\mathbf{x}])^2] = \sigma_\eta^2. \quad (3.49)$$

No other function of \mathbf{x} can do better, because the optimal one achieves an MSE equal to the noise variance, which is irreducible; it represents the intrinsic uncertainty of the system. As Eq. (3.47) suggests, any other function, $f(\mathbf{x})$, will result in an MSE larger by the factor $(\mathbb{E}[y|\mathbf{x}] - f(\mathbf{x}))^2$, which corresponds to the deviation from the optimal one.

3.9.2 BIAS-VARIANCE TRADEOFF

We have just seen that the optimal estimate, in the MSE sense, of the dependent variable in a regression task is given by the conditional expectation $\mathbb{E}[y|\mathbf{x}]$. In practice, any estimator is computed based on a specific training data set, say \mathcal{D} . Let us make the dependence on the training set explicit and express the estimate as a function of \mathbf{x} parameterized on \mathcal{D} , or $f(\mathbf{x}; \mathcal{D})$. A reasonable measure to quantify the performance of an estimator is its mean-square deviation from the optimal one, expressed by $\mathbb{E}_{\mathcal{D}}[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}[y|\mathbf{x}])^2]$, where the mean is taken with respect to all possible training sets, because each one results in a different estimate. Following a similar path as for Eq. (3.20), we obtain

$$\mathbb{E}_{\mathcal{D}}[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}[y|\mathbf{x}])^2] = \underbrace{\mathbb{E}_{\mathcal{D}}[(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[f(\mathbf{x}; \mathcal{D})])^2]}_{\text{Variance}} + \underbrace{(\mathbb{E}_{\mathcal{D}}[f(\mathbf{x}; \mathcal{D})] - \mathbb{E}[y|\mathbf{x}])^2}_{\text{Bias}^2}. \quad (3.50)$$

As was the case for the MSE parameter estimation task when changing from one training set to another, the mean-square deviation from the optimal estimator comprises two terms. The first one is contributed from the variance of the estimator around its own mean value and the second one from the difference of the mean from the optimal estimate; in other words, the bias. It turns out that one *cannot* make both terms small simultaneously. For a fixed number of training points, N , in the data sets \mathcal{D} , trying to minimize the

variance term results in an increase of the bias term and vice versa. This is because, in order to reduce the bias term, one has to increase the complexity (more free parameters) of the adopted estimator $f(\cdot; \mathcal{D})$. This, in turn, results in higher variance as we change the training sets. This is a manifestation of the overfitting issue that we have already discussed. The only way to reduce both terms simultaneously is to increase the number of the training data points, N , and at the same time increase the complexity of the model *carefully*, so as to achieve the aforementioned goal. If one increases the number of training points and at the same time increases the model complexity excessively, the overall MSE may increase. This is known as the *bias-variance dilemma* or *tradeoff*. This is an issue that is omnipresent in any estimation task. Usually, we refer to it as *Occam's razor* rule.

Occam was a logician and a nominalist scholastic medieval philosopher who expressed this law of parsimony: “Plurality must never be posited without necessity.” The great physicist Paul Dirac expressed the same statement from an aesthetics point of view, which underlies mathematical theories: “A theory with a mathematical beauty is more likely to be correct than an ugly one that fits the data.” In our context of model selection, it is understood that one has to select the simplest model that can “explain” the data. Although this is not a scientifically proven result, it underlies the rationale behind a number of developed model selection techniques [1, 32, 33, 38] and [35, Chapter 5], which trade off complexity with accuracy.

Next, we present a simplistic, yet pedagogic, example to demonstrate this tradeoff between bias and variance. We are given the training points plotted in Figure 3.7 in the (x, y) plane. The points have been generated according to a regression model of the form

$$y = g(x) + \eta. \quad (3.51)$$

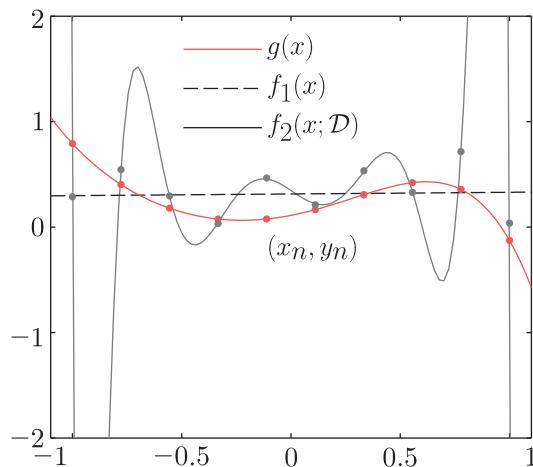


FIGURE 3.7

The observed data are the points denoted as gray dots. These are the result of adding noise to the red points, which lie on the red curve associated with the unknown $g(\cdot)$. Fitting the data by a low degree polynomial, $f_1(x)$, results in high bias; observe that most of the data points lie outside the straight line. On the other hand, the variance of the estimator will be low. In contrast, fitting a high-degree polynomial, $f_2(x; \mathcal{D})$, results in low bias, because the corresponding curve goes through all the data points; however, the respective variance will be high.

The graph of $g(x)$ is shown in [Figure 3.7](#). First, we are going to be very naive and very cautious in spending computational resources, so we have chosen a fixed linear model to fit the data,

$$\hat{y} = f_1(x) = \theta_0 + \theta_1 x,$$

where the values θ_1 and θ_0 have been chosen arbitrarily, irrespective of the training data. The graph of this straight line is shown in [Figure 3.7](#). Because no training was involved and the model parameters are fixed, there is no variation as we change the training sets and $\mathbb{E}_{\mathcal{D}}[f_1(x)] = f_1(x)$, with the variance term being equal to zero. On the other hand, the square of the bias, which is equal to $(f_1(x) - \mathbb{E}[y|x])^2$, is expected to be large because the choice of the model was arbitrary, without paying attention to the training data. In the sequel, we go to the other extreme. We choose a complex class of functions, such as a very high-order polynomial, $f_2(\cdot; \mathcal{D})$. Then, the corresponding graph of the model is expected always to go through the training points. One such curve is illustrated in [Figure 3.7](#). Generate different data sets \mathcal{D}_i as

$$\mathcal{D}_i = \{(g(x_n) + \eta_{i,n}, x_n) : n = 1, 2, \dots, N\}, \quad i = 1, 2, \dots,$$

where $\eta_{i,n}$ denotes different noise samples, drawn from a white noise process. In other words, all training points have the same x -coordinate and the change in the training sets is due to the different values of the noise. For such an experimental setup, the bias term at each point, x_n , $n = 1, 2, \dots, N$, is zero, because

$$\mathbb{E}_{\mathcal{D}}[f_2(x_n; \mathcal{D})] = \mathbb{E}_{\mathcal{D}}[g(x_n) + \eta] = g(x_n) = \mathbb{E}_{\mathcal{D}}[y|x_n].$$

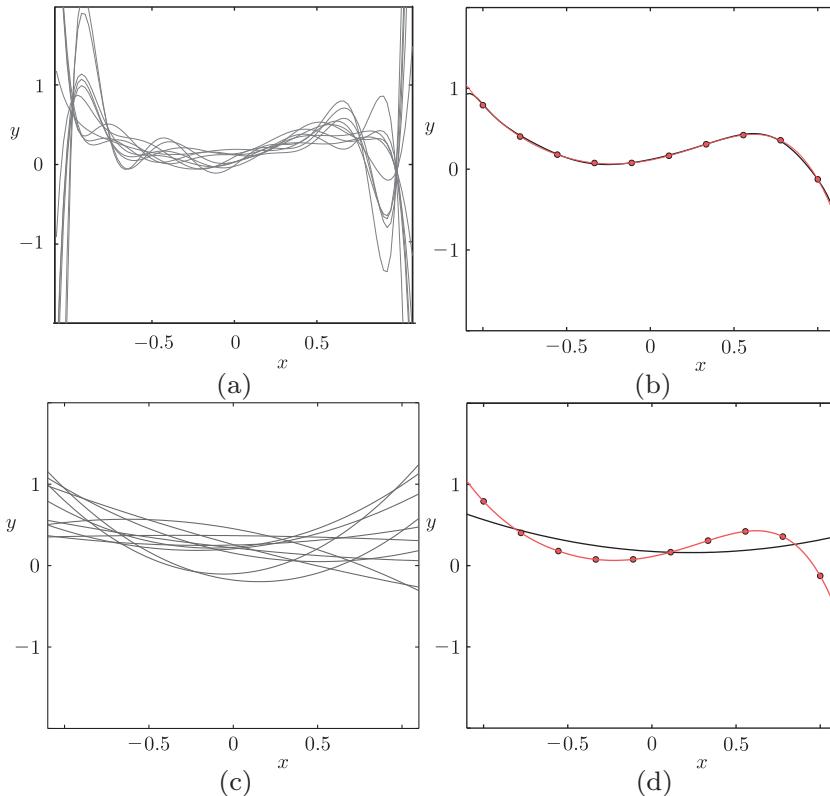
On the other hand, the variance term at the points x_n , $n = 1, 2, \dots, N$, is expected to be large, because

$$\mathbb{E}_{\mathcal{D}}[(f_2(x_n; \mathcal{D}) - g(x_n))^2] = \mathbb{E}_{\mathcal{D}}[(g(x_n) + \eta - g(x_n))^2] = \sigma_\eta^2.$$

Assuming that the functions $f_2(\cdot)$ and $g(\cdot)$ are continuous and smooth enough and the points x_n are dense enough to cover the interval of interest in the real axis, we expect similar behavior at all the points $x \neq x_n$.

A more realistic example is illustrated in [Figure 3.8](#). Consider the model in Eq. (3.51), where $g(\cdot)$ is a fifth-order polynomial. We select a number of points across the respective curve and add noise to them; these comprise the training data set. We run two sets of experiments. The first one attempts to fit in the noisy data a high-order polynomial of degree equal to ten and the second one a low second-order polynomial. For each one of the two setups, we repeat the experiment 1000 times, each time adding a different noise realization to the originally selected points. [Figures 3.8a and c](#) show ten (for visibility reasons, out of the 1000) of the resulting curves for the high- and low-order polynomials, respectively. The substantially higher variance for the case of the high-order polynomial is readily noticed. [Figures 3.8b and d](#) show the corresponding curves, which result from averaging over the 1000 performed experiments, together with the graph of our “unknown” function. The high-order polynomial results in an excellent fit of very low bias. The opposite is true for the case of the second-order polynomial. The reader may find more information on the bias-variance dilemma problem in [16].

Finally, note that the left-hand side of Eq. (3.50) is the mean, with respect to \mathcal{D} , of the second term in Eq. (3.47). It is easy to see that, by reconsidering Eq. (3.47) and taking the expectation on both y and \mathcal{D} , given the value of $\mathbf{x} = \mathbf{x}$, the resulting MSE becomes (try it, following similar arguments as for Eq. (3.50))

**FIGURE 3.8**

(a) Ten of the resulting curves from fitting a tenth-order polynomial and (b) the corresponding average over 1000 different experiments, together with the red curve of the unknown polynomial. The dots indicate the points that give birth to the training data, as described in the text. (c) and (d) illustrate the results from fitting a second-order polynomial. Observe the bias-variance tradeoff as a function of the complexity of the fitted model.

$$\begin{aligned} \text{MSE}(\mathbf{x}) &= \mathbb{E}_{\mathbf{y}|\mathbf{x}} \mathbb{E}_{\mathcal{D}} [(y - f(\mathbf{x}; \mathcal{D}))^2] \\ &= \sigma_\eta^2 + \mathbb{E}_{\mathcal{D}} [(f(\mathbf{x}; \mathcal{D}) - \mathbb{E}_{\mathcal{D}} [f(\mathbf{x}; \mathcal{D})])^2] \\ &\quad + (\mathbb{E}_{\mathcal{D}} [f(\mathbf{x}; \mathcal{D})] - \mathbb{E}[y|\mathbf{x}])^2, \end{aligned} \quad (3.52)$$

where Eq. (3.49) has been used and the product rule, as stated in Chapter 2, has been exploited. In the sequel, one can take the mean over \mathbf{x} . The resulting MSE is also known as the *test* or *generalization error* and it is a measure of the performance of the respective adopted model. Note that the generalization error in Eq. (3.52) involves averaging over (theoretically) all possible training data sets of certain size N . In contrast, the so-called *training error* is computed over a single data set, the one used for the training, and this results in an overoptimistic estimate of the error. We will come back to this important issue in Section 3.13.

3.10 MAXIMUM LIKELIHOOD METHOD

So far, we have approached the estimation problem as an optimization task around a set of training examples, without paying any attention to the underlying statistics that generates these points. We only used statistics in order to check under which conditions the estimators were efficient. However, the optimization step did not involve any statistical information. For the rest of the chapter, we are going to involve statistics more and more. In this section, the ML method is introduced. It is not an exaggeration to say that ML and LS are two of the major pillars on which parameter estimation is based and new methods are inspired from. The ML method was suggested by Sir Ronald Aylmer Fisher.

Once more, we will first formulate the method in a general setting, independent of the regression/classification tasks. We are given a set of say, N , observations, $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, drawn from a probability distribution. We assume that the joint pdf of these N observations is of a known parametric functional type, denoted as $p(\mathcal{X}; \boldsymbol{\theta})$, where the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^K$ is unknown and the task is to estimate its value. This is known as the *likelihood function* of $\boldsymbol{\theta}$ with respect to the given set of observations, \mathcal{X} . According to the ML method, the estimate is provided by

$$\hat{\boldsymbol{\theta}}_{\text{ML}} := \arg \max_{\boldsymbol{\theta} \in \mathcal{A} \subset \mathbb{R}^K} p(\mathcal{X}; \boldsymbol{\theta}) : \quad \text{Maximum Likelihood Estimate.} \quad (3.53)$$

For simplicity, we will assume that the parameter space $\mathcal{A} = \mathbb{R}^K$, and that the parameterized family $\{p(\mathcal{X}; \boldsymbol{\theta}) : \boldsymbol{\theta} \in \mathbb{R}^K\}$ enjoys a unique minimizer with respect to the parameter $\boldsymbol{\theta}$. This is illustrated in [Figure 3.9](#). In other words, given the set of observations $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, one selects the unknown parameter vector so as to make this joint event the most likely one to happen.

Because the logarithmic function, $\ln(\cdot)$, is monotone and increasing, one can instead search for the maximum of the *log-likelihood function*,

$$\frac{\partial \ln p(\mathcal{X}; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{\text{ML}}} = \mathbf{0}. \quad (3.54)$$

Assuming the observations to be i.i.d., the ML estimator has some very attractive properties, namely:

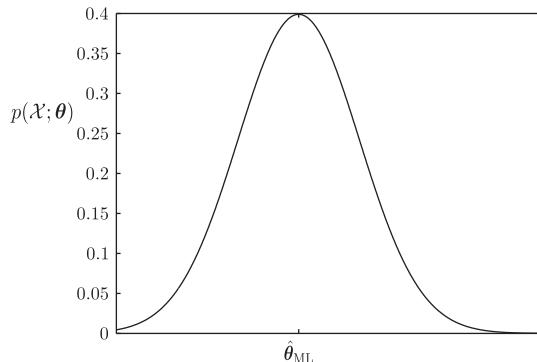


FIGURE 3.9

According to the maximum likelihood method, we assume that, given the set of observations, the estimate of the unknown parameter is the value that maximizes the corresponding likelihood function.

- The ML estimator is asymptotically unbiased; that is, assuming that the model of the pdf, which we have adopted, is correct and there exists a true parameter θ_o , then

$$\lim_{N \rightarrow \infty} \mathbb{E}[\hat{\theta}_{\text{ML}}] = \theta_o. \quad (3.55)$$

- The ML estimate is asymptotically *consistent* so that given any value of $\epsilon > 0$,

$$\lim_{N \rightarrow \infty} \text{Prob} \left\{ \left| \hat{\theta}_{\text{ML}} - \theta_o \right| > \epsilon \right\} = 0, \quad (3.56)$$

that is, for large values of N , we expect the ML estimate to be very close to the true value with high probability.

- The ML estimator is asymptotically efficient; that is, it achieves the Cramér-Rao lower bound.
- If there exists a sufficient statistic, $T(\mathcal{X})$, for an unknown parameter, then only $T(\mathcal{X})$ suffices to express the respective ML estimate ([Problem 3.18](#)).
- Moreover, assuming that an efficient estimator does exist, then this estimator is optimal in the ML sense ([Problem 3.19](#)).

Example 3.5. Let $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, be the observation vectors stemming from a normal distribution with known covariance matrix and unknown mean; that is,

$$p(\mathbf{x}_n; \boldsymbol{\mu}) = \frac{1}{(2\pi)^{l/2} |\Sigma|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{x}_n - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_n - \boldsymbol{\mu}) \right).$$

Assume that the observations are mutually independent. Obtain the ML estimate of the unknown mean vector.

For the N statistically independent observations, the joint log-likelihood function is given by

$$L(\boldsymbol{\mu}) = \ln \prod_{n=1}^N p(\mathbf{x}_n; \boldsymbol{\mu}) = -\frac{N}{2} \ln \left((2\pi)^l |\Sigma| \right) - \frac{1}{2} \sum_{n=1}^N (\mathbf{x}_n - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}_n - \boldsymbol{\mu}).$$

Taking the gradient with respect to $\boldsymbol{\mu}$, we obtain⁵

$$\frac{\partial L(\boldsymbol{\mu})}{\partial \boldsymbol{\mu}} := \begin{bmatrix} \frac{\partial L}{\partial \mu_1} \\ \frac{\partial L}{\partial \mu_2} \\ \vdots \\ \frac{\partial L}{\partial \mu_l} \end{bmatrix} = \sum_{n=1}^N \Sigma^{-1} (\mathbf{x}_n - \boldsymbol{\mu}),$$

and equating to $\mathbf{0}$ leads to

$$\hat{\boldsymbol{\mu}}_{\text{ML}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n.$$

In other words, for Gaussian distributed data, the ML estimate of the mean is the sample mean. Moreover, note that the ML estimate is expressed in terms of its sufficient statistic, (see [Section 3.7](#)).

⁵ Recall from matrix algebra that $\frac{\partial(\mathbf{x}^T \mathbf{b})}{\partial \mathbf{x}} = \mathbf{b}$ and $\frac{\partial(\mathbf{x}^T A \mathbf{x})}{\partial \mathbf{x}} = 2A\mathbf{x}$, if A is symmetric.

3.10.1 LINEAR REGRESSION: THE NONWHITE GAUSSIAN NOISE CASE

Consider the linear regression model

$$\mathbf{y} = \boldsymbol{\theta}^T \mathbf{x} + \boldsymbol{\eta}.$$

We are given N training data points (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$. The corresponding (unobserved) noise samples, $\boldsymbol{\eta}_n$, $n = 1, \dots, N$, are assumed to follow a jointly Gaussian distribution with zero mean and covariance matrix equal to $\Sigma_{\boldsymbol{\eta}}$. Our goal is to obtain the ML estimate of the parameters $\boldsymbol{\theta}$.

The joint log-likelihood function of $\boldsymbol{\theta}$, with respect to the training set, is given by

$$L(\boldsymbol{\theta}) = -\frac{N}{2} \ln(2\pi) - \frac{1}{2} \ln |\Sigma_{\boldsymbol{\eta}}| - \frac{1}{2} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T \Sigma_{\boldsymbol{\eta}}^{-1} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}), \quad (3.57)$$

where $\mathbf{y} := [y_1, y_2, \dots, y_N]^T$, and $\mathbf{X} := [\mathbf{x}_1, \dots, \mathbf{x}_N]^T$ stands for the input matrix. Taking the gradient with respect to $\boldsymbol{\theta}$, we get

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{X}^T \Sigma_{\boldsymbol{\eta}}^{-1} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}), \quad (3.58)$$

and equating to the zero vector, we obtain

$$\hat{\boldsymbol{\theta}}_{\text{ML}} = \left(\mathbf{X}^T \Sigma_{\boldsymbol{\eta}}^{-1} \mathbf{X} \right)^{-1} \mathbf{X}^T \Sigma_{\boldsymbol{\eta}}^{-1} \mathbf{y}. \quad (3.59)$$

Remarks 3.3.

- Compare Eq. (3.59) with the LS solution given in Eq. (3.17). They are different, unless the covariance matrix of the successive noise samples, $\Sigma_{\boldsymbol{\eta}}$, is diagonal and of the form $\sigma_{\boldsymbol{\eta}}^2 I$; that is, if the noise is Gaussian as well as white. In this case, the LS and the ML solutions coincide. However, if the noise sequence is nonwhite, the two estimates differ. Moreover, it can be shown ([Problem 3.8](#)) that, *in this case of colored Gaussian noise, the ML estimate is an efficient one and it attains the Cramér-Rao bound, even if N is finite.*

3.11 BAYESIAN INFERENCE

In our discussion, so far, we have assumed that the parameter associated with the functional form of the adopted model is a deterministic constant, whose value is unknown to us. In this section, we will follow a different rationale. The unknown parameter will be treated as a random variable. Hence, whenever our goal is to estimate its value, this is conceived as an effort to estimate the value of a *specific* realization that corresponds to the observed data. A more detailed discussion concerning the Bayesian inference rationale is provided in [Chapter 12](#). As the name Bayesian suggests, the heart of the method beats around the celebrated Bayes theorem. Given two jointly distributed random vectors, say, \mathbf{x} , $\boldsymbol{\theta}$, Bayes theorem states that

$$p(\mathbf{x}, \boldsymbol{\theta}) = p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathbf{x})p(\mathbf{x}). \quad (3.60)$$

David Bayes (1702-1761) was an English mathematician and a Presbyterian minister who first developed the basics of the theory. However, it was Pierre-Simon Laplace (1749-1827), the great French mathematician, who further developed and popularized it.

Assume that $\mathbf{x}, \boldsymbol{\theta}$ are two statistically dependent random vectors. Let $\mathcal{X} = \{\mathbf{x}_n \in \mathbb{R}^l, n = 1, 2, \dots, N\}$, be the set of the observations resulting from N successive experiments. Then, Bayes theorem gives

$$p(\boldsymbol{\theta}|\mathcal{X}) = \frac{p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{X})} = \frac{p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta}) d\boldsymbol{\theta}}. \quad (3.61)$$

Obviously, if the observations are i.i.d., then we can write

$$p(\mathcal{X}|\boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{x}_n|\boldsymbol{\theta}).$$

In the previous formulas, $p(\boldsymbol{\theta})$ is the a priori pdf concerning the statistical distribution of $\boldsymbol{\theta}$, and $p(\boldsymbol{\theta}|\mathcal{X})$ is the conditional or a posteriori pdf, formed after the set of N observations has been obtained. The prior probability density, $p(\boldsymbol{\theta})$, can be considered as a constraint that *encapsulates our prior knowledge* about $\boldsymbol{\theta}$. No doubt, our uncertainty about $\boldsymbol{\theta}$ is modified after the observations have been received, because more information is now disclosed to us. If the adopted assumptions about the underlying models are sensible, we expect the posterior pdf to be a more accurate one to describe the statistical nature of $\boldsymbol{\theta}$. We will refer to the process of approximating the pdf of a random quantity, based on a set of training data, as *inference*, to differentiate it from the process of estimation, that returns a single value for each parameter/variable. So, according to the inference approach, one attempts to draw conclusions about the nature of the randomness that underlies the variables of interest. This information can in turn be used to make predictions and to take decisions.

We will exploit Eq. (3.61) in two ways. The first refers to our familiar goal of obtaining an estimate of the parameter vector $\boldsymbol{\theta}$, which “controls” the model that describes the generation mechanism of our observations, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$. Because \mathbf{x} and $\boldsymbol{\theta}$ are two statistically dependent random vectors, we know from Section 3.9 that the MSE optimal estimate of the value of $\boldsymbol{\theta}$, given \mathcal{X} , is

$$\hat{\boldsymbol{\theta}} = \mathbb{E}[\boldsymbol{\theta}|\mathcal{X}] = \int \boldsymbol{\theta} p(\boldsymbol{\theta}|\mathcal{X}) d\boldsymbol{\theta}. \quad (3.62)$$

Another direction along which one can exploit the Bayes theorem, in the context of statistical inference, is to obtain an estimate of the pdf of \mathbf{x} given the observations \mathcal{X} . This can be done by *marginalizing* over a distribution, using the equation

$$p(\mathbf{x}|\mathcal{X}) = \int p(\mathbf{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{X}) d\boldsymbol{\theta}, \quad (3.63)$$

where the conditional independence of \mathbf{x} on \mathcal{X} , given the value $\boldsymbol{\theta} = \boldsymbol{\theta}$, expressed as $p(\mathbf{x}|\mathcal{X}, \boldsymbol{\theta}) = p(\mathbf{x}|\boldsymbol{\theta})$, has been used. Equation (3.63) provides an estimate of the unknown pdf, by exploiting the information that resides in the obtained observations as well as in the adopted functional dependence on the parameters $\boldsymbol{\theta}$. Note that, in contrast to what we did in the case of the ML method, where we used the observations to obtain an estimate of the parameter vector, here we assume the parameters to be random variables, we provide our prior knowledge about $\boldsymbol{\theta}$ via $p(\boldsymbol{\theta})$ and integrate the joint pdf, $p(\mathbf{x}, \boldsymbol{\theta}|\mathcal{X})$, over $\boldsymbol{\theta}$.

Once $p(\mathbf{x}|\mathcal{X})$ is available, it can be used for prediction. Assuming that we have obtained the observations $\mathbf{x}_1, \dots, \mathbf{x}_N$, our estimate about the next value, \mathbf{x}_{N+1} , to occur can be determined via $p(\mathbf{x}_{N+1}|\mathcal{X})$. Obviously, the form of $p(\mathbf{x}|\mathcal{X})$ is, in general, changing as new observations are obtained,

because each time an observation becomes available, part of our uncertainty about the underlying randomness is removed.

Example 3.6. Consider the simplified linear regression task of Eq. (3.31) and assume $x = 1$. As we have already said, this problem is that of estimating the value of a constant buried in noise. Our methodology will follow the Bayesian philosophy. Assume that the noise samples are i.i.d. drawn from a Gaussian pdf of zero mean and variance σ_η^2 . However, we impose our a priori knowledge concerning the unknown θ , via the prior distribution

$$p(\theta) = \mathcal{N}(\theta_0, \sigma_0^2). \quad (3.64)$$

That is, we assume that we know that the values of θ lie around θ_0 , and σ_0^2 quantifies our degree of uncertainty about this prior knowledge. Our goals are first to obtain the a posteriori pdf, given the set of measurements $\mathbf{y} = [y_1, \dots, y_N]^T$, and then to obtain $\mathbb{E}[\theta|\mathbf{y}]$, according to Eqs. (3.61) and (3.62) and adapting them to our current notational needs. We have that

$$\begin{aligned} p(\theta|\mathbf{y}) &= \frac{p(\mathbf{y}|\theta)p(\theta)}{p(\mathbf{y})} = \frac{1}{p(\mathbf{y})} \left(\prod_{n=1}^N p(y_n|\theta) \right) p(\theta) \\ &= \frac{1}{p(\mathbf{y})} \left(\prod_{n=1}^N \frac{1}{\sqrt{2\pi}\sigma_\eta} \exp\left(-\frac{(y_n - \theta)^2}{2\sigma_\eta^2}\right) \right) \\ &\quad \times \frac{1}{\sqrt{2\pi}\sigma_0} \exp\left(-\frac{(\theta - \theta_0)^2}{2\sigma_0^2}\right). \end{aligned} \quad (3.65)$$

After some algebraic manipulations on Eq. (3.65) (Problem 3.23), one ends up in the following:

$$p(\theta|\mathbf{y}) = \frac{1}{\sqrt{2\pi}\sigma_N} \exp\left(-\frac{(\theta - \bar{\theta}_N)^2}{2\sigma_N^2}\right), \quad (3.66)$$

where

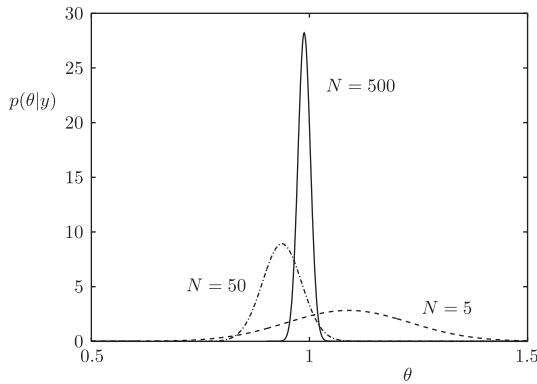
$$\bar{\theta}_N = \frac{N\sigma_0^2\bar{y}_N + \sigma_\eta^2\theta_0}{N\sigma_0^2 + \sigma_\eta^2}, \quad (3.67)$$

with $\bar{y}_N = \frac{1}{N} \sum_{n=1}^N y_n$ being the sample mean of the observations and

$$\sigma_N^2 = \frac{\sigma_\eta^2\sigma_0^2}{N\sigma_0^2 + \sigma_\eta^2}. \quad (3.68)$$

In other words, if the prior and the conditional pdfs are Gaussians, then the posterior is also Gaussian. Moreover, the mean and the variance of the posterior are given by Eqs. (3.67) and (3.68), respectively.

Observe that as the number of observations increases, $\bar{\theta}_N$ tends to the sample mean of the observations; recall that the latter is the estimate that results from the ML method. Also, note that the variance keeps decreasing as the number of observations increases, which is in line with common sense, because more observations mean less uncertainty. Figure 3.10 illustrates the previous discussion. Data samples, y_n , were generated using a Gaussian pseudorandom generator with mean equal to $\theta = 1$ and variance equal to $\sigma_\eta^2 = 0.1$. So the true value of our constant is equal to 1. We used a Gaussian

**FIGURE 3.10**

In the Bayesian inference approach, note that as the number of observations increases, our uncertainty about the true value of the unknown parameter is reduced and the mean of the posterior pdf tends to the true value and the variance tends to zero.

prior pdf with mean value equal to $\theta_0 = 2$ and variance $\sigma_0^2 = 6$. We observe that as N increases, the posterior pdf gets narrower and its mean tends to the true value of 1.

It should be pointed out that in the case of this example, both the ML and LS estimates become identical, or

$$\hat{\theta} = \frac{1}{N} \sum_{n=1}^N y_n = \bar{y}_N.$$

This will also be the case for the mean value in Eq. (3.67), if we set σ_0^2 very large, as might happen if we have no confidence in our initial estimate of θ_0 and we assign a very large value to σ_0^2 . In effect, this is equivalent to using no prior information.

Let us now investigate what happens if our prior knowledge about θ_0 is “embedded” in the LS criterion in the form of a constraint. This can be done by modifying the constraint in Eq. (3.38), such that

$$(\theta - \theta_0)^2 \leq \rho, \quad (3.69)$$

which leads to the minimization of the following Lagrangian

$$\text{minimize } L(\theta, \lambda) = \sum_{n=1}^N (y_n - \theta)^2 + \lambda ((\theta - \theta_0)^2 - \rho). \quad (3.70)$$

Taking the derivative with respect to θ and equating to zero, we obtain

$$\hat{\theta} = \frac{N\bar{y}_N + \lambda\theta_0}{N + \lambda},$$

which, for $\lambda = \sigma_\eta^2/\sigma_0^2$, becomes identical to Eq. (3.67). The world is small after all! This has happened only because we used Gaussians both for the conditional as well as the prior pdfs. For different forms

of pdfs, this would not be the case. However, this example shows that a close relationship ties priors and constraints. They both attempt to impose prior information. Each method, in its own unique way, is associated with the respective pros and cons. In [Chapters 12 and 13](#), where a more extended treatment of the Bayesian inference task is provided, we will see that the very essence of regularization, which is a means against overfitting, lies at the heart of the Bayesian approach.

Finally, one may wonder if the Bayesian inference has offered us any more information, compared to the deterministic parameter estimation path. After all, when the aim is to obtain a specific value for the unknown parameter, then taking the mean of the Gaussian posterior comes to the same solution, which results from the regularized LS approach. Well, even for this simple case, the Bayesian inference readily provides a piece of extra information; this is an estimate of the variance around the mean, which is very valuable in order to assess our trust of the recovered estimate. Of course, all these are valid provided that the adopted pdfs offer a good description of the statistical nature of the process at hand [\[24\]](#).

Finally, it can be shown, [Problem 3.24](#), that the previously obtained results are generalized for the more general linear regression model, of nonwhite Gaussian noise, which was considered in [Section 3.10](#), as shown by

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta}.$$

It turns out that the posterior pdf is also Gaussian with mean value equal to

$$\mathbb{E}[\boldsymbol{\theta}|\mathbf{y}] = \boldsymbol{\theta}_0 + \left(\Sigma_0^{-1} + \mathbf{X}^T \Sigma_{\eta}^{-1} \mathbf{X} \right)^{-1} \mathbf{X}^T \Sigma_{\eta}^{-1} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}_0), \quad (3.71)$$

and

$$\Sigma_{\boldsymbol{\theta}|\mathbf{y}} = \left(\Sigma_0^{-1} + \mathbf{X}^T \Sigma_{\eta}^{-1} \mathbf{X} \right)^{-1}. \quad (3.72)$$

3.11.1 THE MAXIMUM A POSTERIORI PROBABILITY ESTIMATION METHOD

The Maximum A Posteriori Probability Estimation technique, usually denoted as MAP, is based on the Bayesian theorem, but it does not go as far as the Bayesian philosophy allows. The goal becomes that of obtaining an estimate which maximizes Eq. [\(3.61\)](#); in other words,

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg \max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\mathcal{X}) : \quad \text{MAP Estimate}, \quad (3.73)$$

and because $p(\mathcal{X})$ is independent of $\boldsymbol{\theta}$, this leads to

$$\begin{aligned} \hat{\boldsymbol{\theta}}_{\text{MAP}} &= \arg \max_{\boldsymbol{\theta}} p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \\ &= \arg \max_{\boldsymbol{\theta}} \{ \ln p(\mathcal{X}|\boldsymbol{\theta}) + \ln p(\boldsymbol{\theta}) \}. \end{aligned} \quad (3.74)$$

If we consider [Example 3.6](#), it is a matter of simple exercise to obtain the MAP estimate and show that

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \frac{N\bar{y}_N + \frac{\sigma_{\eta}^2}{\sigma_0^2}\boldsymbol{\theta}_0}{N + \frac{\sigma_{\eta}^2}{\sigma_0^2}} = \bar{\boldsymbol{\theta}}_N. \quad (3.75)$$

Note that for this case, the MAP estimate coincides with the regularized LS solution, for $\lambda = \sigma_\eta^2/\sigma_0^2$. Once more, we verify that adopting a prior pdf for the unknown parameter acts as a regularizer, which embeds into the problem the available prior information.

Remarks 3.4.

- Observe that for the case of the [Example 3.6](#), all three estimators, namely ML, MAP and the Bayesian (taking the mean), result *asymptotically*, as N increases, in the same estimate. This is a more general result and it is true for other pdfs as well as for the case of parameter vectors. As the number of observations increases, our uncertainty is reduced and $p(\mathcal{X}|\boldsymbol{\theta}), p(\boldsymbol{\theta}|\mathcal{X})$ peak sharply around a value of $\boldsymbol{\theta}$. This forces all the methods to result in similar estimates. However, the obtained estimates are different for finite values of N . Recently, as we will see in [Chapters 12](#) and [13](#), Bayesian methods have become very popular, and seem to be the choice, among the three methods, for a number of practical problems.
- The choice of the prior pdf in the Bayesian methods is not an innocent task. In [Example 3.6](#), we chose the conditional pdf (likelihood function) as well as the prior pdf to be Gaussians. We saw that the posterior pdf was also Gaussian. The advantage of such a choice was that we could come to closed form solutions. This is not always the case, and then the computation of the posterior pdf needs sampling methods or other approximate techniques. We will come to that in [Chapters 12](#) and [14](#). However, the family of Gaussians is not the only one with this nice property of leading to closed form solutions. In probability theory, if the posterior is of the same form as the prior, we say that $p(\boldsymbol{\theta})$ is a *conjugate prior* of the likelihood function $p(\mathcal{X}|\boldsymbol{\theta})$ and then the involved integrations can be carried out in closed form, see, e.g., [15, 30] and [Chapter 12](#). Hence, the Gaussian pdf is a conjugate of itself.
- Just for the sake of pedagogical purposes, it is interesting to recapitulate some of the nice properties that the Gaussian pdf possesses. We have met these properties in various sections and problems in the book, so far: (a) it is a conjugate of itself; (b) if two random variables (vectors) are jointly Gaussian, then their marginal pdfs are also Gaussian and the posterior pdf of one w.r.t. the other is also Gaussian; (c) moreover, the linear combination of jointly Gaussian variables turns out to be Gaussian; (d) as a by-product, it turns out that the sum of statistically independent Gaussian random variables is also a Gaussian one; and finally (e) the central limit theorem states that the sum of a large number of independent random variables tends to be Gaussian, as the number of the summands increases.

3.12 CURSE OF DIMENSIONALITY

In a number of places in this chapter, we mentioned the need of having a large number of training points. In [Section 3.9.2](#), while discussing the bias-variance tradeoff, it was stated that in order to end up with a low overall MSE, the complexity (number of parameters) of the model should be small enough with respect to the number of training points. In [Section 3.8](#), overfitting was discussed and it was pointed out that, if the number of training points is small with respect to the number of parameters, overfitting occurs.

The question that is now raised is how big a data set should be, in order to be more relaxed concerning the performance of the designed predictor. The answer to the previous question depends largely on the

dimensionality of the input space. It turns out that, the larger the dimension of the input space the more data points are needed. This is related to the so-called *curse of dimensionality*, a term coined for the first time in [3].

Let us assume that we are given the same number of points, N , thrown randomly in a unit cube (hypercube) in two different spaces, one being of low and the other of very high dimension. Then, the average distance of the points in the latter case will be much larger than that in the low-dimensional space case. As a matter of fact, the average distance shows a dependence that is analogous to the exponential term ($N^{-1/l}$), where l is the dimensionality of the space [14, 35]. For example, the average distance between two out of 10^{10} points in the 2-dimensional space is 10^{-5} and in the 40-dimensional space is equal to 1.83. Figure 3.11 shows two cases, each one consisting of 100 points. The red points lie on a (one-dimensional) line segment of length equal to one and were generated according to the uniform distribution. Gray points cover a (two-dimensional) square region of unit area, which were also generated by a two-dimensional uniform distribution. Observe that, the square area is more sparsely populated compared to the line segment. This is the general trend and high-dimensional spaces are sparsely populated; thus, many more data points are needed in order to fill in the space with enough data. Fitting a model in a parameter space, one must have enough data covering sufficiently well all regions in the space, in order to be able to learn well enough the input-output functional dependence, (Problem 3.26).

There are various ways to cope with the curse of dimensionality and try to exploit the available data set in the best possible way. A popular direction is to resort to suboptimal solutions by projecting the input/feature vectors in a lower dimensional subspace or manifold. Very often, such an approach leads to small performance losses, because the original training data, although they are generated in a high-dimensional space, in fact they may “live” in a lower-dimensional subspace or manifold, due to physical

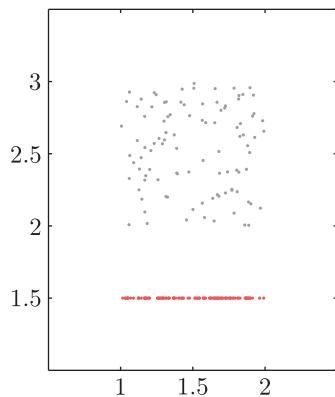


FIGURE 3.11

A simple experiment, which demonstrates the curse of dimensionality. A number of 100 points are generated randomly, drawn from a uniform distribution, in order to fill the 1-d segment of length equal to one ($[1, 2] \times \{1.5\}$) (red points), and the two-dimensional rectangular region of unit area, $[1, 2] \times [2, 3]$ (gray points). Observe that, although the number of points in both cases is the same, the rectangular region is sparsely populated compared to the densely populated line segment.

dependencies that restrict the number of free parameters. Take as an example a case where the data are 3-dimensional vectors, but they lie around a straight line, which is a one-dimensional linear manifold (affine set or subspace if it crosses the origin) or around a circle (one-dimensional nonlinear manifold) embedded in the 3-dimensional space. That is, the true number of free parameters, in this case, is equal to one; this is because one free parameter suffices to describe the location of a point on a circle or on a straight line. The true number of free parameters is also known as the *intrinsic dimensionality* of the problem. The challenge, now, becomes that of learning the subspace/manifold onto which to project. These issues will be considered in more detail in [Chapter 19](#).

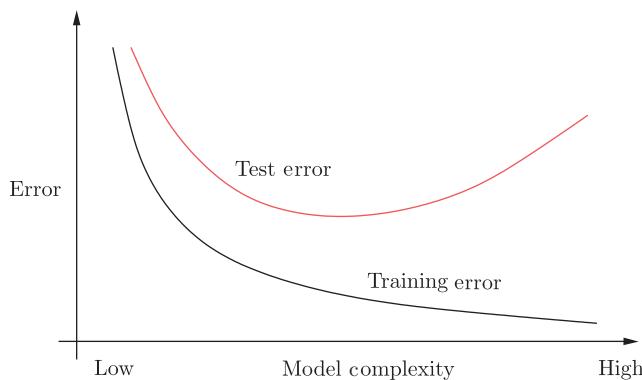
Finally, it has to be noted that the dimensionality of the input space is not always the crucial issue. In pattern recognition, it has been shown that the critical factor is the so-called *VC-dimension* of a classifier. In a number of classifiers, such as (generalized) linear classifiers or neural networks (to be considered in [Chapter 18](#)), the VC-dimension is directly related to the dimensionality of the input space. However, one can design classifiers, such as the support vector machines ([Chapter 11](#)), whose performance is not directly related to the input space and they can be efficiently designed in spaces of very high (of even infinite) dimensionality [35, 38].

3.13 VALIDATION

From previous sections, we already know that what is a “good” estimate according to one set of training points, it is not necessarily a good one for other data sets. This is an important aspect in any machine learning task; the performance of a method may vary with the random choice of the training set. A major phase, in any machine learning task, is to quantify/predict the performance that the designed (prediction) model is expected to exhibit in practice. It will not come as a surprise to state that “measuring” the performance against the training data set would lead to an “optimistic” value of the performance index, because this is computed on the same set on which the estimate was optimized; this trend has been known since the early 1930s [22]. For example, if the model is complex enough, with a large number of free parameters, the training error may even become zero, since a perfect fit to the data can be achieved. What is more meaningful and fair is to look for the so-called *generalization* performance of an estimator; that is, its average performance computed over *different* data sets, which *did not* participate in the training (see, [Section 3.9.2](#)).

[Figure 3.12](#) shows a typical performance that is expected to result in practice. The error measured on the (single) training data set is shown together with the (average) test/generalization error as the model complexity varies. If one tries to fit a complex model, with respect to the size of the available training set, then the error measured on the training set will be overoptimistic. On the contrary, the true error, as this is represented by the test error, takes large values; in the case where the performance index is the MSE, this is mainly contributed by the variance term ([Section 3.9.2](#)). On the other hand, if the model is too simple it leads the test error also to large values; for the MSE case, this time the contribution is mainly due to the bias term. The idea is to have a model complexity that corresponds to the minimum of the respective curve. As a matter of fact, this is the point that various model selection techniques try to predict.

For some simple cases and under certain assumptions concerning the underlying models, we are able to have analytical formulas that quantify the average performance as we change data sets. However, in practice, this is hardly the case, and one must have a way to test the performance of an obtained

**FIGURE 3.12**

The training error tends to zero as the model complexity increases; for complex enough models with a large number of free parameters, a perfect fit to the training data is possible. However, the test error initially decreases, because more complex models “learn” the data better, to a point. After that point of complexity, the test error increases.

classifier/predictor using different data sets. The process is known as *validation* and there are a number of alternatives that one can resort to.

Assuming that enough data are at the designer’s disposal, one can split the data into one part, to be used for training, and another part for testing the performance. For example, the probability of error is computed over the test data set for the case of a classifier, or the MSE for the case of a regression task; other measures of fit can also been used. If this path is taken, one has to make sure that both the size of the training set as well as the size of the test set are large enough, with respect to the model complexity; a large test data set is required in order to provide a statistically sound result on the test error. Especially if different methods are compared, the smaller the difference in their comparative performance is expected to be, the larger the size of the test set must be made, in order to guarantee reliable conclusions [35, Chapter 10].

Cross-validation

In practice, very often the size of the available data is not sufficient and one cannot afford to “lose” part of it from the training set for the sake of testing. *Cross-validation* is a very common technique that is usually employed. Cross-validation has been rediscovered a number of times; however, to our knowledge, the first published description can be traced back to [25]. According to this method, the data set is split into, say K , roughly equal-sized parts. We repeat training K times, each time selecting one (different each time) part of the data for testing and the remaining $K - 1$ parts for training. This gives us the advantage of testing with a part of the data that has not been involved in the training, hence it can be considered as being independent, and at the same time using, eventually, all the data both for training and testing. Once we finish, we can (a) combine the obtained K estimates by averaging or via another more advanced way and (b) combine the test errors to get a better estimate of the generalization error that our estimator is expected to exhibit in real-life applications. The method is known as K -fold cross-validation. An extreme case is when we use $K = N$, so that each time one sample is left for testing.

This is sometimes referred to as the *leave-one-out* (LOO) cross-validation method. The price one pays for K -fold cross-validation is the complexity of training K times. In practice, the value of K depends very much on the application, but typical values are of the order of 5 to 10.

The cross-validation estimator of the generalization error is very nearly unbiased. The reason for the slight bias is that the training set in cross-validation is slightly smaller than the actual data set. The effect of this bias will be conservative in the sense that the estimated fit will be slightly biased in the direction suggesting a poorer fit. In practice, this bias is rarely a concern, especially in the LOO case, where each time only one sample is left out. The variance, however, of the cross-validation estimator can be large, and this has to be taken into account when comparing different methods. In [12], the use of *bootstrap* techniques is suggested in order to reduce the variance of the obtained error predictions by the cross-validation method.

Moreover, besides complexity and high variance, cross-validation schemes are not beyond criticisms. Unfortunately, the overlap among the training sets introduces unknowable dependencies between runs, making the use of formal statistical tests difficult [10]. All this discussion reveals that the validation task is far from innocent. Ideally, one should have at her/his disposal large data sets and divide them in several *nonoverlapping* training sets, of whatever size is appropriate, along with separate test sets (or a single one) that are (is) large enough. More on different validation schemes and their properties can be found in, e.g., [2, 11, 17, 35] and an insightful related discussion in [26].

3.14 EXPECTED AND EMPIRICAL LOSS FUNCTIONS

What was said before in our discussion concerning the generalization and the training set-based performance of an estimator, can be given a more formal statement via the notion of *expected loss*. Adopting a loss function, $\mathcal{L}(\cdot, \cdot)$, in order to quantify the deviation between the predicted value, $\hat{y} = f(\mathbf{x})$, and the respective true one, y , the corresponding expected loss is defined as

$$J(f) := \mathbb{E} [\mathcal{L}(y, f(\mathbf{x}))], \quad (3.76)$$

or more explicitly

$$J(f) = \int \dots \int \mathcal{L}(y, f(\mathbf{x})) p(y, \mathbf{x}) dy d\mathbf{x} : \text{ Expected Loss Function,} \quad (3.77)$$

where the integration is replaced by summation whenever the respective variables are discrete. As a matter of fact, this is the ideal cost function one would like to optimize with respect to $f(\cdot)$, in order to get the optimal estimator over *all* possible values of the input-output pairs. However, such an optimization would in general be a very hard task, even if one knew the functional form of the joint distribution. Thus, in practice, one has to be content with two approximations. First, the functions to be searched are constrained within a certain family, \mathcal{F} , (in this chapter, we focused on parametrically described families of functions). Second, because the joint distribution is either unknown and/or the integration may not be analytically tractable, the expected loss is approximated by the so-called *empirical loss* version, defined as

$$J_N(f) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) : \text{ Empirical Loss Function.} \quad (3.78)$$

As an example, the MSE function, discussed earlier, is the expected loss associated with the squared error loss function and the LS cost is the respective empirical version. For large enough values of N and provided that the family of functions is restricted enough,⁶ we expect that the outcome from optimizing J_N to be close to that which would be obtained by optimizing J [38].

From the validation point of view, given any prediction function, $f(\cdot)$, what we called generalization error corresponds to the corresponding value of J in Eq. (3.77) and the training error to that of J_N in Eq. (3.78).

We can now take the discussion a little further, which will reveal some more secrets concerning the accuracy-complexity tradeoff in machine learning. Let f_* be the function that optimizes the expected loss,

$$f_* := \arg \min_f J(f), \quad (3.79)$$

and $f_{\mathcal{F}}$ the optimal after *constraining* the task within the family of functions \mathcal{F} ,

$$f_{\mathcal{F}} := \arg \min_{f \in \mathcal{F}} J(f). \quad (3.80)$$

Let us also define

$$f_N := \arg \min_{f \in \mathcal{F}} J_N(f). \quad (3.81)$$

Then, we can readily write that

$$\mathbb{E}[J(f_N) - J(f_*)] = \underbrace{\mathbb{E}[J(f_{\mathcal{F}}) - J(f_*)]}_{\text{approximation error}} + \underbrace{\mathbb{E}[J(f_N) - J(f_{\mathcal{F}})]}_{\text{estimation error}}. \quad (3.82)$$

The *approximation error* measures the deviation in the generalization error, if instead of the overall optimal function one uses the optimal obtained within a certain family of functions. The *estimation error* measures the deviation due to optimizing the empirical instead of the expected loss. If one chooses the family of functions to be very large, then it is expected that the approximation error will be small, because there is high probability f_* will be close to one of the members of the family. However, the estimation error is expected to be large, because for a fixed number of data points, N , fitting a complex function is likely to lead to overfitting. For example, if the family of functions is the class of polynomials of a very large order, a very large number of parameters are to be estimated and overfitting will occur. The opposite is true if the class of functions is a small one. In parametric modeling, complexity of a family of functions is related to the number of free parameters. However, this is not the whole story. As a matter of fact, complexity is really measured by the so-called *capacity* of the associated set of functions. The VC-dimension mentioned in Section 3.12 is directly related to the capacity of the family of the considered classifiers. More concerning the theoretical treatment of these issues can be obtained from [9, 38, 39].

⁶ That is, the family of functions is not very large. To keep the discussion simple, take the example of quadratic class of functions. This is larger than that of the linear ones, because the latter is a special case (subset) of the former.

3.15 NONPARAMETRIC MODELING AND ESTIMATION

The focus of this chapter is on the task of parameter estimation and on techniques that spring from the idea of parametric functional modeling of an input-output dependence. To put the final touches on this chapter, we shift our attention to the alternative philosophy that runs across the field of statistical estimation; that of *nonparametric modeling*. In contrast to parametric modeling, either no parameters are involved or if parameters pop in, their number is not fixed but grows with the number of training samples. We will treat such models in the context of reproducing kernel Hilbert spaces (RKHS) in [Chapter 11](#). There, instead of parameterizing the family of functions, in which one constrains the search for finding the prediction model, the candidate solution is constrained to lie within a specific functional space.

In this section, the nonparametric modeling rationale is demonstrated in the framework of approximating an unknown pdf. Although such techniques are very old, they can still be used and they are also the focus of more recent research efforts [13].

Our kick off point is the classical *histogram* approximation of an unknown pdf. Let us assume that we are given a set of points, $x_n \in \mathbb{R}$, $n = 1, 2, \dots, N$, which have been independently drawn from an unknown distribution. [Figure 3.13a](#) illustrates a pdf approximation using the histogram technique. The

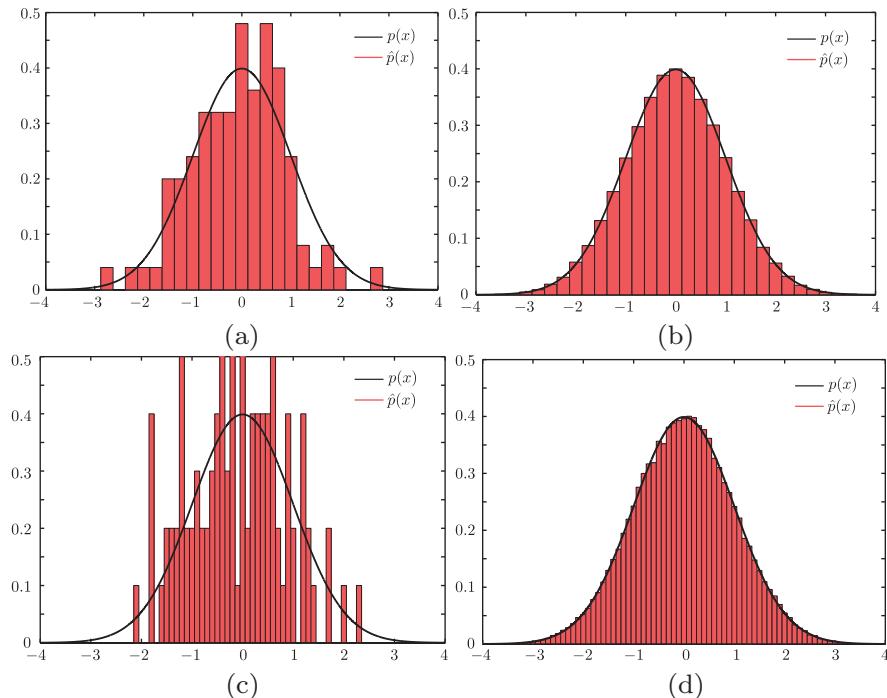


FIGURE 3.13

The gray curve corresponds to the true pdf. The red curves correspond to the histogram approximation method, for various values of the pair (h, N) . (a) $h = 0.25$ and $N = 100$, (b) $h = 0.25$ and $N = 10^5$, (c) $h = 0.1$ and $N = 100$, (d) $h = 0.1$ and $N = 10^5$. The larger the data size and the smaller the size of the bin the better the approximation becomes.

real axis is divided into a number of successive interval bins, each of length h , which is a user-defined constant. Let us focus on one of these interval bins and denote its middle point as \hat{x} . Count the number of observations that lie inside this bin, say, k_N . Then, the pdf approximation for all the points within this specific interval is given by

$$\hat{p}(x) = \frac{1}{h} \frac{k_N}{N}, \quad \text{if } |x - \hat{x}| \leq \frac{h}{2}. \quad (3.83)$$

This results to a “staircase” approximation of the continuous pdf function. It turns out that this very simple rule converges to $p(x)$, provided that $h \rightarrow 0$, $k_N \rightarrow \infty$ and $k_N/N \rightarrow 0$. That is, when the length of the bins become small, the number of observations in each bin is large enough to guarantee that the *frequency ratio* k_N/N is a good estimate of the probability of a point lying in the bin, and the number of observations tends to infinity faster than the number of points in the bins. [Figure 3.13a](#) corresponds to a (relatively) large value of h and a (relatively) small number of points. [Figure 3.13d](#) corresponds to a (relatively) small value of h and a relatively large number of points. Observe that, in the latter case, the approximation of the pdf is smoother and closer to the true curve. The training points were generated by a Gaussian of variance equal to one.

To apply the histogram approximation method, in practice, we first select h . Then, given a point x , we count the number of the observations that lie within the interval $[x - h/2, x + h/2]$, and we make use of the ratio in Eq. (3.83) to obtain the estimate $\hat{p}(x)$. To dress up what we have just described with a mathematical formalism, define

$$\phi(x) := \begin{cases} 1, & \text{if } |x| \leq 1/2, \\ 0, & \text{otherwise.} \end{cases} \quad (3.84)$$

Then, it is easily checked out that the histogram estimate at x is given by

$$\hat{p}(x) = \frac{1}{h} \frac{1}{N} \sum_{n=1}^N \phi\left(\frac{x - x_n}{h}\right). \quad (3.85)$$

Indeed, the summation is equal to the number of observations that lie within the interval $[x - h/2, x + h/2]$. An alternative way to view Eq. (3.85) is as an expansion over a set of functions, each one centered at an observation point. However, although such an expansion converges to the true value, it is a bit unorthodox, because it attempts to approximate a continuous function in terms of discontinuous ones. As a matter of fact, this is a reason that the convergence of histogram methods is slow, in the sense that many points, N , are required for a reasonably good approximation. This can also be verified from [Figure 3.13](#), where in spite of the fact that 10^5 points have been used, the approximation is still not very good. Note that what we have said so far can be generalized to any Euclidean space \mathbb{R}^l .

Parzen [28] in order to bypass the drawback that we have just stated, proved that the approximation is still possible if one replaces the discontinuous function $\phi(\cdot)$, by a smooth one. Such functions are known as *kernels*, *potential functions*, or *Parzen windows*, and must satisfy the following conditions:

$$\phi(x) \geq 0 \text{ and} \quad (3.86)$$

$$\int \phi(x) dx = 1, \quad (3.87)$$

where the more general case of a Euclidean space, \mathbb{R}^l , has been chosen as the data space. The Gaussian pdf is, obviously, such a function. For such a choice, one can write the Parzen approximation of an unknown pdf as

$$\hat{p}(\mathbf{x}) = \frac{1}{N} \sum_{n=1}^N \frac{1}{(2\pi)^{l/2} h^l} \exp\left(-\frac{(\mathbf{x} - \mathbf{x}_n)^T (\mathbf{x} - \mathbf{x}_n)}{2h^2}\right). \quad (3.88)$$

In words, according to the Parzen approximation, a kernel function (the Gaussian in this case) is centered at each one of the observations and we take their sum. Such types of expansions will be a popular theme in this book. An interesting issue is to search for ways to reduce the number of points that contribute into the summation, by selecting the most important ones. That is, we will try to make such expansions more *sparse*. As we will see, besides issues related to the computational load, reducing the number of terms is in line with our effort to be more robust against overfitting; Occam's razor rule once again.

The way we have approached the task of pdf function approximation, so far in this section, was to select a bin of *fixed size*, h , centered at the point of interest \mathbf{x} . In higher-dimensional spaces, the interval bin becomes a square of length h , in the two-dimensional case, a cube in three dimensions, and a hypercube in higher dimensions. The other alternative is to fix the number of points, k , and try to increase the volume of the hypercube around \mathbf{x} , so that k points are included. Because the approximation depends on the ratio $\frac{1}{N} \frac{k}{V}$, this is also a good idea. In dense (high values of pdf) areas, k points will be clustered within regions of small volume, and in less dense (low values of pdf) areas they will fill in regions of larger volume. Moreover, one can now consider alternatives to hypercube shapes, such as hyperspheres, hyperellipsoids, and so on. The algorithmic procedure is simple. Search for the k -nearest neighbors of \mathbf{x} , among the available observations, and compute the volume in the space within which they are located. Then, estimate the value of the pdf at \mathbf{x} , using the previously stated ratio. This is known as the *k -nearest neighbor density estimation*. More on the topic can be found in, e.g., [35, 36].

The previous technique of the k -nearest neighbors can be further relaxed and be emancipated from the idea of estimating pdfs. Then, it gives birth to one of the most widely known and used methods for classification, known as the k -nearest neighbor classification rule, which is discussed in [Chapter 7](#).

PROBLEMS

- 3.1** Let $\hat{\theta}_i, i = 1, 2, \dots, m$, be unbiased estimators of a parameter vector $\boldsymbol{\theta}$, so that $\mathbb{E}[\hat{\theta}_i] = \boldsymbol{\theta}$, $i = 1, \dots, m$. Moreover, assume that the respective estimators are uncorrelated to each other and that all have the same (total) variance, $\sigma^2 = \mathbb{E}[(\hat{\theta}_i - \boldsymbol{\theta})^T (\hat{\theta}_i - \boldsymbol{\theta})]$. Show that by averaging the estimates, e.g.,

$$\hat{\theta} = \frac{1}{m} \sum_{i=1}^m \hat{\theta}_i,$$

the new estimator has total variance $\sigma_c^2 := \mathbb{E}[(\hat{\theta} - \boldsymbol{\theta})^T (\hat{\theta} - \boldsymbol{\theta})] = \frac{1}{m} \sigma^2$.

- 3.2** Let a random variable x being described by a uniform pdf in the interval $[0, \frac{1}{\theta}]$, $\theta > 0$. Assume a function⁷ g , which defines an estimator $\hat{\theta} := g(x)$ of θ . Then, for such an estimator to be unbiased, the following must hold:

$$\int_0^{\frac{1}{\theta}} g(x) dx = 1.$$

However, such a function g does not exist.

- 3.3** A family $\{p(\mathcal{D}; \theta) : \theta \in \mathcal{A}\}$ is called *complete* if, for any vector function $\mathbf{h}(\mathcal{D})$ such that $\mathbb{E}_{\mathcal{D}}[\mathbf{h}(\mathcal{D})] = \mathbf{0}$, $\forall \theta$, then $\mathbf{h} = \mathbf{0}$.

Show that if $\{p(\mathcal{D}; \theta) : \theta \in \mathcal{A}\}$ is complete, and there exists an MVU estimator, then this estimator is unique.

- 3.4** Let $\hat{\theta}_u$ be an unbiased estimator, so that $\mathbb{E}[\hat{\theta}_u] = \theta_o$. Define a biased one by $\hat{\theta}_b = (1 + \alpha)\hat{\theta}_u$. Show that the range of α where the MSE of $\hat{\theta}_b$ is smaller than that of $\hat{\theta}_u$ is

$$-2 < -\frac{2\text{MSE}(\hat{\theta}_u)}{\text{MSE}(\hat{\theta}_u) + \theta_o^2} < \alpha < 0.$$

- 3.5** Show that for the setting of the Problem 3.4, the optimal value of α is equal to

$$\alpha_* = -\frac{1}{1 + \frac{\theta_o^2}{\text{var}(\hat{\theta}_u)}},$$

where, of course, the variance of the unbiased estimator is equal to the corresponding MSE.

- 3.6** Show that the regularity condition for the Cramér-Rao bound holds true if the order of integration and differentiation can be interchanged.

- 3.7** Derive the Cramér-Rao bound for the LS estimator, when the training data result from the linear model

$$y_n = \theta x_n + \eta_n, \quad n = 1, 2, \dots,$$

where x_n and η_n are i.i.d. samples of a zero mean random variable, with variance σ_x^2 , and a Gaussian one with zero mean and variance σ_η^2 , respectively. Assume, also, that x and η are independent. Then, show that the LS estimator achieves the CR bound only asymptotically.

- 3.8** Let us consider the regression model

$$y_n = \boldsymbol{\theta}^T \mathbf{x}_n + \eta_n, \quad n = 1, 2, \dots, N,$$

where the noise samples $\boldsymbol{\eta} = [\eta_1, \dots, \eta_N]^T$ come from a zero mean Gaussian random vector, with covariance matrix Σ_η . If $X = [\mathbf{x}_1, \dots, \mathbf{x}_N]^T$ stands for the input matrix, and $\mathbf{y} = [y_1, \dots, y_N]^T$, then show that,

$$\hat{\boldsymbol{\theta}} = \left(X^T \Sigma_\eta^{-1} X \right)^{-1} X^T \Sigma_\eta^{-1} \mathbf{y},$$

is an efficient estimate.

⁷ To avoid any confusion, let g be Lebesgue integrable on intervals of \mathbb{R} .

Notice, here, that the previous estimate coincides with the ML one. Moreover, bear in mind that in the case where $\Sigma_\eta = \sigma^2 I$ then the ML estimate becomes equal to the LS one.

- 3.9** Assume a set of i.i.d. $\mathcal{X} = \{x_1, x_2, \dots, x_N\}$ samples of a random variable, with mean μ and variance σ^2 . Define also the quantities

$$\begin{aligned} S_\mu &:= \frac{1}{N} \sum_{n=1}^N x_n, & S_{\sigma^2} &:= \frac{1}{N} \sum_{n=1}^N (x_n - S_\mu)^2, \\ \bar{S}_{\sigma^2} &:= \frac{1}{N} \sum_{n=1}^N (x_n - \mu)^2. \end{aligned}$$

Show that if μ is considered to be known, a sufficient statistic for σ^2 is \bar{S}_{σ^2} . Moreover, in the case where both (μ, σ^2) are unknown, then a sufficient statistic is the pair (S_μ, S_{σ^2}) .

- 3.10** Show that solving the task

$$\text{minimize } L(\theta, \lambda) = \sum_{n=1}^N \left(y_n - \theta_0 - \sum_{i=1}^l \theta_i x_{ni} \right)^2 + \lambda \sum_{i=1}^l |\theta_i|^2,$$

is equivalent with minimizing

$$\text{minimize } L(\theta, \lambda) = \sum_{n=1}^N \left((y_n - \bar{y}) - \sum_{i=1}^l \theta_i (x_{ni} - \bar{x}_i) \right)^2 + \lambda \sum_{i=1}^l |\theta_i|^2,$$

and the estimate of θ_0 is given by

$$\hat{\theta}_0 = \bar{y} - \sum_{i=1}^l \hat{\theta}_i \bar{x}_i.$$

- 3.11** This problem refers to [Example 3.4](#), where a linear regression task with a real valued unknown parameter θ_o is considered. Show that $\text{MSE}(\hat{\theta}_b(\lambda)) < \text{MSE}(\hat{\theta}_{\text{MVU}})$ or the ridge regression estimate shows a lower MSE performance than the one for the MVU estimate, if

$$\begin{cases} \lambda \in (0, \infty), & \theta_o^2 \leq \frac{\sigma_\eta^2}{N}, \\ \lambda \in \left(0, \frac{2\sigma_\eta^2}{\theta_o^2 - \frac{\sigma_\eta^2}{N}} \right), & \theta_o^2 > \frac{\sigma_\eta^2}{N}. \end{cases}$$

Moreover, the minimum MSE performance for the ridge regression estimate is attained at $\lambda_* = \sigma_\eta^2 / \theta_o^2$.

- 3.12** Assume that the model that generates the data is

$$y_n = A \sin \left(\frac{2\pi}{N} kn + \phi \right) + \eta_n,$$

where $A > 0$, and $k \in \{1, 2, \dots, N-1\}$. Assume that η_n are i.i.d. samples from a Gaussian noise, of variance σ_η^2 . Show that there is no unbiased estimator for the phase, ϕ , based on N measurement points, $y_n, n = 0, 1, \dots, N-1$, that attains the Cramér-Rao bound.

- 3.13** Show that if (\mathbf{y}, \mathbf{x}) are two jointly distributed random vectors, with values in $\mathbb{R}^k \times \mathbb{R}^l$, then the MSE optimal estimator of \mathbf{y} given the value $\mathbf{x} = \mathbf{x}$ is the regression of \mathbf{y} conditioned on \mathbf{x} , or $\mathbb{E}[\mathbf{y}|\mathbf{x}]$.

- 3.14** Assume that \mathbf{x}, \mathbf{y} are jointly Gaussian random vectors, with covariance matrix

$$\Sigma := \mathbb{E} \left[\begin{bmatrix} \mathbf{x} - \boldsymbol{\mu}_x \\ \mathbf{y} - \boldsymbol{\mu}_y \end{bmatrix} \begin{bmatrix} (\mathbf{x} - \boldsymbol{\mu}_x)^T, (\mathbf{y} - \boldsymbol{\mu}_y)^T \end{bmatrix} \right] = \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix}.$$

Assuming also that the matrices Σ_x and $\bar{\Sigma} := \Sigma_y - \Sigma_{yx}\Sigma_x^{-1}\Sigma_{xy}$ are nonsingular, then show that the optimal MSE estimator $\mathbb{E}[\mathbf{y}|\mathbf{x}]$ takes the following form

$$\mathbb{E}[\mathbf{y}|\mathbf{x}] = \mathbb{E}[\mathbf{y}] + \Sigma_{yx}\Sigma_x^{-1}(\mathbf{x} - \boldsymbol{\mu}_x).$$

Notice that $\mathbb{E}[\mathbf{y}|\mathbf{x}]$ is an affine function of \mathbf{x} . In other words, for the case where \mathbf{x} and \mathbf{y} are jointly Gaussian, the optimal estimator of \mathbf{y} , in the MSE sense, which is in general a nonlinear function, becomes an affine function of \mathbf{x} .

In the special case where \mathbf{x}, \mathbf{y} are scalar random variables, then

$$\mathbb{E}[y|x] = \mu_y + \frac{\alpha\sigma_y}{\sigma_x}(x - \mu_x),$$

where α stands for the *correlation coefficient*, defined as

$$\alpha := \frac{\mathbb{E}[(x - \mu_x)(y - \mu_y)]}{\sigma_x\sigma_y},$$

with $|\alpha| \leq 1$. Notice, also, that the previous assumption on the nonsingularity of Σ_x and $\bar{\Sigma}$ translates, in this special case, to $\sigma_x \neq 0 \neq \sigma_y$, and $|\alpha| < 1$.

Hint: Use the matrix inversion lemma from Appendix A, in terms of the Schur complement $\bar{\Sigma}$ of Σ_x in Σ and the fact that $\det(\Sigma) = \det(\Sigma_y)\det(\bar{\Sigma})$.

- 3.15** Assume a number l of jointly Gaussian random variables $\{x_1, x_2, \dots, x_l\}$, and a nonsingular matrix $A \in \mathbb{R}^{l \times l}$. If $\mathbf{x} := [x_1, x_2, \dots, x_l]^T$, then show that the components of the vector \mathbf{y} , obtained by $\mathbf{y} = A\mathbf{x}$, are also jointly Gaussian random variables.

A direct consequence of this result is that any linear combination of jointly Gaussian variables is also Gaussian.

- 3.16** Let \mathbf{x} be a vector of jointly Gaussian random variables of covariance matrix Σ_x . Consider the general linear regression model

$$\mathbf{y} = \Theta\mathbf{x} + \boldsymbol{\eta},$$

where $\Theta \in \mathbb{R}^{k \times l}$ is a parameter matrix and $\boldsymbol{\eta}$ is the noise vector which is considered to be Gaussian, with zero mean, and with covariance matrix Σ_η , independent of \mathbf{x} . Then show that \mathbf{y} and \mathbf{x} are jointly Gaussian, with covariance matrix given by

$$\Sigma = \begin{bmatrix} \Theta\Sigma_x\Theta^T + \Sigma_\eta & \Theta\Sigma_x \\ \Sigma_x\Theta^T & \Sigma_x \end{bmatrix}.$$

- 3.17** Show that a linear combination of Gaussian independent variables is also Gaussian.

- 3.18** Show that if a sufficient statistic $T(\mathcal{X})$ for a parameter estimation problem exists, then $T(\mathcal{X})$ suffices to express the respective ML estimate.

- 3.19** Show that if an efficient estimator exists, then it is also optimal in the ML sense.
- 3.20** Let the observations resulting from an experiment be $x_n, n = 1, 2, \dots, N$. Assume that they are independent and that they originate from a Gaussian pdf $\mathcal{N}(\mu, \sigma^2)$. Both, the mean and the variance, are unknown. Prove that the ML estimates of these quantities are given by

$$\hat{\mu}_{\text{ML}} = \frac{1}{N} \sum_{n=1}^N x_n, \quad \hat{\sigma}_{\text{ML}}^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \hat{\mu}_{\text{ML}})^2.$$

- 3.21** Let the observations $x_n, n = 1, 2, \dots, N$, come from the uniform distribution

$$p(x; \theta) = \begin{cases} \frac{1}{\theta}, & 0 \leq x \leq \theta, \\ 0, & \text{otherwise.} \end{cases}$$

Obtain the ML estimate of θ .

- 3.22** Obtain the ML estimate of the parameter $\lambda > 0$ of the exponential distribution

$$p(x) = \begin{cases} \lambda \exp(-\lambda x), & x \geq 0, \\ 0, & x < 0, \end{cases}$$

based on a set of measurements, $x_n, n = 1, 2, \dots, N$.

- 3.23** Assume an $\mu \sim \mathcal{N}(\mu_0, \sigma_0^2)$, and a stochastic process $\{x_n\}_{n=-\infty}^\infty$, consisting of i.i.d. random variables, such that $p(x_n | \mu) = \mathcal{N}(\mu, \sigma^2)$. Consider a number of N members of the process $\{x_n\}_{n=-\infty}^\infty$, so that $\mathcal{X} := \{x_1, x_2, \dots, x_N\}$, and prove that the posterior $p(x | \mathcal{X})$, of any $x = x_{n_0}$ conditioned on \mathcal{X} , turns out to be Gaussian with mean μ_N and variance $\sigma^2 + \sigma_N^2$, where

$$\mu_N := \frac{N\sigma_0^2 \bar{x} + \sigma^2 \mu_0}{N\sigma_0^2 + \sigma^2}, \quad \sigma_N^2 := \frac{\sigma^2 \sigma_0^2}{N\sigma_0^2 + \sigma^2}.$$

- 3.24** Show that for the linear regression model,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta},$$

the a posteriori probability $p(\boldsymbol{\theta} | \mathbf{y})$ is a Gaussian one, if the prior distribution probability is given by $p(\boldsymbol{\theta}) = N(\boldsymbol{\theta}_0, \Sigma_0)$, and the noise samples follow the multivariate Gaussian distribution $p(\boldsymbol{\eta}) = N(\mathbf{0}, \Sigma_\eta)$. Compute the mean vector and the covariance matrix of the posterior distribution.

- 3.25** Assume that $x_n, n = 1, 2, \dots, N$, are i.i.d. observations from a Gaussian $\mathcal{N}(\mu, \sigma^2)$. Obtain the MAP estimate of μ , if the prior follows the exponential distribution

$$p(\mu) = \lambda \exp(-\lambda \mu), \quad \lambda > 0, \mu \geq 0.$$

- 3.26** Consider, once more, the same regression model as that of [Problem 3.8](#), but with $\Sigma_\eta = I_N$. Compute the MSE of the predictions $\mathbb{E}[(\mathbf{y} - \hat{\mathbf{y}})^2]$, where \mathbf{y} is the true response and $\hat{\mathbf{y}}$ is the predicted value, given a test point \mathbf{x} and using the LS estimator,

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}.$$

The LS estimator has been obtained via a set of N measurements, collected in the (fixed) input matrix X and y , where the notation has been introduced previously in this chapter. The expectation $\mathbb{E}[\cdot]$ is taken with respect to y , the training data, \mathcal{D} and the test points x . Observe the dependence of the MSE on the dimensionality of the space.

Hint. Consider, first, the MSE, given the value of a test point x , and then take the average over all the test points.

REFERENCES

- [1] H. Akaike, A new look at the statistical model identification, *IEEE Trans. Autom. Control* 19 (6) (1970) 716-723.
- [2] S. Arlot, A. Celisse, A survey of cross-validation procedures for model selection, *Stat. Surv.* 4 (2010) 40-79.
- [3] R.E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, 1957.
- [4] A. Ben-Israel, T.N.E. Greville, *Generalized Inverses: Theory and Applications*, second ed., Springer-Verlag, New York, 2003.
- [5] D. Bertsekas, A. Nedic, O. Ozdaglar, *Convex Analysis and Optimization*, Athena Scientific, Belmont, MA, 2003.
- [6] S. Boyd, L. Vandenberghe, *Convex Optimization*, Cambridge University Press, Cambridge, 2004.
- [7] O. Chapelle, B. Scholkopf, A. Zien, *Semisupervised Learning*, MIT Press, Cambridge, 2006.
- [8] H. Cramer, *Mathematical Methods of Statistics*, Princeton University Press, Princeton, 1946.
- [9] L. Devroye, L. Györfi, G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer, New York, 1991.
- [10] T.G. Dietterich, Approximate statistical tests for comparing supervised classification learning algorithms, *Neural Comput.* 10 (1998) 1895-1923.
- [11] R. Duda, P. Hart, D. Stork, *Pattern Classification*, second ed., Wiley, New York, 2000.
- [12] A. Efron, R. Tibshirani, Improvements on cross-validation: the .632+ bootstrap method, *J. Am. Stat. Assoc.* 92 (438) (1997) 548-560.
- [13] D. Erdogmus, J.C. Principe, From linear adaptive filtering to nonlinear information processing, *IEEE Signal Process. Mag.* 23 (6) (2006) 14-33.
- [14] J.H. Friedman, Regularized discriminant analysis, *J. Am. Stat. Assoc.* 84 (1989) 165-175.
- [15] A. Gelman, J.B. Carlin, H.S. Stern, D.B. Rubin, *Bayesian Data Analysis*, second ed., CRC Press, Boca Raton, FL, 2003.
- [16] S. Geman, E. Bienenstock, R. Doursat, Neural networks and the bias-variance dilemma, *Neural Comput.* 4 (1992) 1-58.
- [17] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, second ed., Springer, New York, 2009.
- [18] A.E. Hoerl, R.W. Kennard, Ridge regression: biased estimation for nonorthogonal problems, *Technometrics* 12 (1) (1970) 55-67.
- [19] S. Kay, Y. Eldar, Rethinking biased estimation, *IEEE Signal Process. Mag.* 25 (6) (2008) 133-136.
- [20] S. Kay, *Statistical Signal Processing*, Prentice Hall, Upper Saddle River, NJ, 1993.
- [21] M. Kendall, A. Stuart, *The Advanced Theory of Statistics*, vol. 2, MacMillan, New York, 1979.
- [22] S.C. Larson, The shrinkage of the coefficient of multiple correlation, *J. Educ. Psychol.* 22 (1931) 45-55.
- [23] E.L. Lehmann, H. Scheffé, Completeness, similar regions, and unbiased estimation: Part II, *Sankhyā* 15 (3) (1955) 219-236.
- [24] D. McKay, Probable networks and plausible predictions—a review of practical Bayesian methods for supervised neural networks, *Netw. Comput. Neural Syst.* 6 (1995) 169-505.

- [25] F. Mosteller, J.W. Tukey, *Handbook of Social Psychology*, Chap. Data Analysis, Including Statistics, Addison-Wesley, Reading, MA, 1954.
- [26] R.M. Neal, Assessing relevance determination methods using DELVE, in: C.M. Bishop (Ed.), *Neural Networks and Machine Learning*, Springer-Verlag, New York, 1998, pp. 97-129.
- [27] A. Papoulis, P. Unnikrishna, *Probability, Random Variables, and Stochastic Processes*, fourth ed., McGraw Hill, New York, NY, 2002.
- [28] E. Parzen, On the estimation of a probability density function and mode, *Ann. Math. Stat.*, 33 (1962) 1065-1076.
- [29] D.L. Phillips, A technique for the numerical solution of certain integral equations of the first kind, *J. Assoc. Comput. Mach.* 9 (1962) 84-97.
- [30] H. Raiffa, R. Schlaifer, *Applied Statistical Decision Theory*, Division of Research, Graduate School of Business Administration, Harvard University, Boston, 1961.
- [31] R.C. Rao, Information and the accuracy attainable in the estimation of statistical parameters, *Bull. Calcutta Math. Soc.* 37 (1945) 81-89.
- [32] J. Rissanen, A universal prior for integers and estimation by minimum description length, *Ann. Stat.* 11 (2) (1983) 416-431.
- [33] G. Schwartz, Estimating the dimension of the model, *Ann. Stat.* 6 (1978) 461-464.
- [34] J. Shao, *Mathematical Statistics*, Springer, New York, 1998.
- [35] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, fourth ed., Academic Press, New York, 2009.
- [36] S. Theodoridis, A. Pikrakis, K. Koutroumbas, D. Cavouras, *An Introduction to Pattern Recognition: A MATLAB Approach*, Academic Press, New York, 2010.
- [37] A.N. Tychonoff, V.Y. Arsenin, *Solution of Ill-posed Problems*, Winston & Sons, Washington, 1977.
- [38] V.N. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.
- [39] V.N. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, New York, 1998.

This page intentionally left blank

MEAN-SQUARE ERROR LINEAR ESTIMATION

4

CHAPTER OUTLINE

4.1	Introduction	105
4.2	Mean-Square Error Linear Estimation: The Normal Equations	106
4.2.1	The Cost Function Surface	107
4.3	A Geometric Viewpoint: Orthogonality Condition	109
4.4	Extension to Complex-Valued Variables	111
4.4.1	Widely Linear Complex-Valued Estimation	113
	<i>Circularity Conditions</i>	114
4.4.2	Optimizing with Respect to Complex-Valued Variables: Wirtinger Calculus	116
4.5	Linear Filtering	118
4.6	MSE Linear Filtering: A Frequency Domain Point of View	120
	<i>Deconvolution: Image Deblurring</i>	121
4.7	Some Typical Applications	124
4.7.1	Interference Cancellation	124
4.7.2	System Identification	125
4.7.3	Deconvolution: Channel Equalization	126
4.8	Algorithmic Aspects: The Levinson and the Lattice-Ladder Algorithms	132
	<i>Forward and Backward MSE Optimal Predictors</i>	134
4.8.1	The Lattice-Ladder Scheme	137
	<i>Orthogonality of the Optimal Backward Errors</i>	138
4.9	Mean-Square Error Estimation of Linear Models	140
4.9.1	The Gauss-Markov Theorem	143
4.9.2	Constrained Linear Estimation: The Beamforming Case	145
4.10	Time-Varying Statistics: Kalman Filtering	148
Problems		154
	<i>MATLAB Exercises</i>	156
References		158

4.1 INTRODUCTION

Mean-square error linear estimation is a topic of fundamental importance for parameter estimation in statistical learning. Besides historical reasons, which take us back to the pioneering works of Kolmogorov, Wiener, and Kalman, who laid the foundations of the optimal estimation field, understanding

mean-square error estimation is a must, prior to studying more recent techniques. One always has to grasp the basics and learn the classics prior to getting involved with new “adventures.” Many of the concepts to be discussed in this chapter are also used in the next chapters.

Optimizing via a loss function, which builds around the square of the error, has a number of advantages such as a single optimal value, which can be obtained via the solution of a linear set of equations; this is a very attractive feature in practice. Moreover, due to the relative simplicity of the resulting equations, the newcomer in the field can get a better feeling of the various notions associated with optimal parameter estimation. The elegant geometric interpretation of the mean-square error solution, via the orthogonality theorem, is presented and discussed. In the chapter, emphasis is also given to computational complexity issues while solving for the optimal solution. The essence behind these techniques remains exactly the same as that inspiring a number of computationally efficient schemes for online learning, to be discussed later in this book.

The development of the chapter is around real-valued variables, something that will be true for most of the book. However, complex-valued signals are particularly useful in a number of areas, with communications being a typical example, and the generalization from the real to the complex domain may not always be trivial. Although in most of the cases, the difference lies in changing matrix transpositions by Hermitian ones, this is not the whole story. This is the reason that we chose to deal with complex-valued data in separate sections, whenever the differences from the real data are not trivial and some subtle issues are involved.

4.2 MEAN-SQUARE ERROR LINEAR ESTIMATION: THE NORMAL EQUATIONS

The general estimation task has been introduced in [Chapter 3](#). There, it was stated that given two dependent random vectors, \mathbf{y} and \mathbf{x} , the goal of the estimation task is to obtain a function, g , so as, given a value \mathbf{x} of \mathbf{x} , to be able to predict (estimate), in some optimal sense, the corresponding value \mathbf{y} of \mathbf{y} , or $\hat{\mathbf{y}} = g(\mathbf{x})$. The mean-square error (MSE) estimation was also presented in [Chapter 3](#) and it was shown that the optimal MSE estimate of \mathbf{y} given the value $\mathbf{x} = \mathbf{x}$ is

$$\hat{\mathbf{y}} = \mathbb{E}[\mathbf{y}|\mathbf{x}].$$

In general, this is a nonlinear function. We now turn our attention to the case where g is *constrained* to be a linear function. For simplicity and in order to pay more attention to the concepts, we will restrict our discussion to the case of scalar dependent variables. The more general case will be discussed later on.

Let $(\mathbf{y}, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^l$ be two jointly distributed random entities of *zero mean values*. In case the mean values are not zero, they are subtracted. Our goal is to obtain an estimate of $\boldsymbol{\theta} \in \mathbb{R}^l$ in the linear estimator model,

$$\hat{\mathbf{y}} = \boldsymbol{\theta}^T \mathbf{x}, \quad (4.1)$$

so that

$$J(\boldsymbol{\theta}) = \mathbb{E}[(\mathbf{y} - \hat{\mathbf{y}})^2], \quad (4.2)$$

is minimum, or

$$\boldsymbol{\theta}_* := \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \quad (4.3)$$

In other words, the optimal estimator is chosen so as to minimize the variance of the error random variable

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}. \quad (4.4)$$

Minimizing the cost function $J(\boldsymbol{\theta})$ is equivalent with setting its gradient with respect to $\boldsymbol{\theta}$ equal to zero,

$$\begin{aligned} \nabla J(\boldsymbol{\theta}) &= \nabla \mathbb{E} [(\mathbf{y} - \boldsymbol{\theta}^T \mathbf{x}) (\mathbf{y} - \boldsymbol{\theta}^T \mathbf{x})] \\ &= \nabla \left\{ \mathbb{E}[y^2] - 2\boldsymbol{\theta}^T \mathbb{E}[\mathbf{xy}] + \boldsymbol{\theta}^T \mathbb{E}[\mathbf{xx}^T] \boldsymbol{\theta} \right\} \\ &= -2\mathbf{p} + 2\Sigma_x \boldsymbol{\theta} = \mathbf{0} \end{aligned}$$

or

$$\boxed{\Sigma_x \boldsymbol{\theta}_* = \mathbf{p} : \text{Normal Equations}}, \quad (4.5)$$

where the input-output cross-correlation vector \mathbf{p} is given by¹

$$\mathbf{p} = [\mathbb{E}[x_1 y], \dots, \mathbb{E}[x_l y]]^T = \mathbb{E}[\mathbf{xy}], \quad (4.6)$$

and the respective covariance matrix is given by

$$\Sigma_x = \mathbb{E}[\mathbf{xx}^T].$$

Thus, the weights of the optimal linear estimator are obtained via a linear system of equations, provided that the covariance matrix is *positive definite* and hence it can be inverted. Moreover, in this case, the solution is *unique*. On the contrary, if Σ_x is singular and hence cannot be inverted, there are infinitely many solutions ([Problem 4.1](#)).

4.2.1 THE COST FUNCTION SURFACE

Elaborating on the cost function, $J(\boldsymbol{\theta})$, as it is defined in [\(4.2\)](#), we get that

$$J(\boldsymbol{\theta}) = \sigma_y^2 - 2\boldsymbol{\theta}^T \mathbf{p} + \boldsymbol{\theta}^T \Sigma_x \boldsymbol{\theta}. \quad (4.7)$$

¹ The cross-correlation vector is usually denoted as \mathbf{r}_{xy} . Here we will use \mathbf{p} , in order to simplify the notation.

Adding and subtracting the term $\theta_*^T \Sigma_x \theta_*$ and taking into account the definition of θ_* from (4.5), it is readily seen that

$$J(\theta) = J(\theta_*) + (\theta - \theta_*)^T \Sigma_x (\theta - \theta_*), \quad (4.8)$$

where

$$J(\theta_*) = \sigma_y^2 - p^T \Sigma_x^{-1} p = \sigma_y^2 - \theta_*^T \Sigma_x \theta_* = \sigma_y^2 - p^T \theta_*, \quad (4.9)$$

is the minimum achieved at the optimal solution. From (4.8) and (4.9), the following remarks can be made.

Remarks 4.1.

- The cost at the optimal value θ_* is always less than the variance $\mathbb{E}[y^2]$ of the output variable. This is guaranteed by the positive definite nature of Σ_x or Σ_x^{-1} , which makes the second term on the right-hand side in (4.9) always positive, unless $p = \mathbf{0}$; however, the cross-correlation vector will only be zero if x and y are uncorrelated. Well, in this case, one cannot say anything (make any prediction) about y by observing samples of x , at least as far as the MSE criterion is concerned, which turns out to involve information residing up to the second order statistics. In this case, the variance of the error, which coincides with $J(\theta_*)$, will be equal to the variance σ_y^2 ; the latter is a measure of the “intrinsic” uncertainty of y around its (zero) mean value. On the contrary, if the input-output variables are correlated, then observing x removes part of the uncertainty associated with y .
- For any value θ , other than the optimal θ_* , the error variance increases as (4.8) suggests, due to the positive definite nature of Σ_x . Figure 4.1 shows the cost function (mean-square error) surface defined by $J(\theta)$ in (4.8). The corresponding isovalue contours are shown in Figure 4.2. In general, they are ellipses, whose axes are determined by the eigenstructure of Σ_x . For $\Sigma_x = \sigma^2 I$, where all eigenvalues are equal to σ^2 , the contours are circles (Problem 4.3).

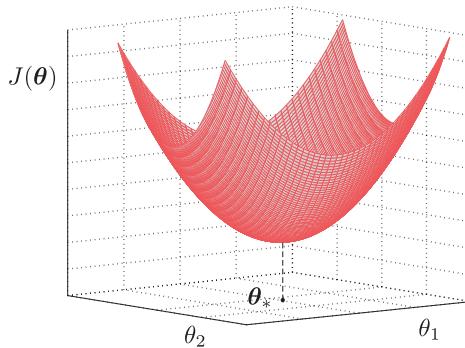
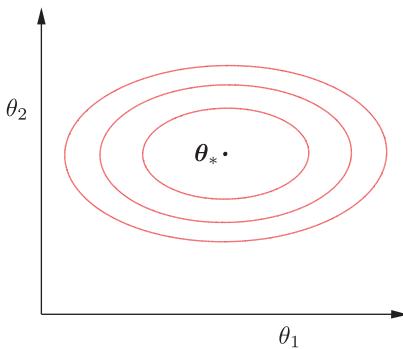


FIGURE 4.1

The MSE cost function has the form of a (hyper) paraboloid.

**FIGURE 4.2**

The isovalue contours for the cost function surface corresponding to Figure 4.1. They are ellipses; the major axis of each ellipse is determined by the maximum eigenvalue λ_{\max} and the minor one by the smaller λ_{\min} of the Σ of the input random variables. The largest the ratio $\frac{\lambda_{\max}}{\lambda_{\min}}$ is the more elongated the ellipse becomes. The ellipses become circles, if the covariance matrix has the special form of $\sigma^2 I$. That is, all variables are mutually uncorrelated and they have the same variance. By varying Σ , different shapes of the ellipses and different orientations result.

4.3 A GEOMETRIC VIEWPOINT: ORTHOGONALITY CONDITION

A very intuitive view of what we have said so far comes from the geometric interpretation of the random variables. The reader can easily check out that the set of random variables is a *vector space* over the field of real (and complex) numbers. Indeed, if x and y are any two random variables then $x + y$, as well as αx , are also random variables for every $\alpha \in \mathbb{R}$.² We can now equip this vector space with an inner product operation, which also implies a norm and makes it a *Euclidean space*. The reader can easily check out that the mean value operation has all the properties required for an operation to be called an inner product. Indeed, for any subset of random variables,

- $\mathbb{E}[xy] = \mathbb{E}[yx]$,
- $\mathbb{E}[(\alpha_1 x_1 + \alpha_2 x_2)y] = \alpha_1 \mathbb{E}[x_1 y] + \alpha_2 \mathbb{E}[x_2 y]$,
- $\mathbb{E}[x^2] \geq 0$, with equality if and only if $x = 0$.

Thus, the norm induced by this inner product,

$$\|x\| := \sqrt{\mathbb{E}[x^2]},$$

coincides with the respective standard deviation (assuming $\mathbb{E}[x] = 0$). From now on, given two uncorrelated random variables, x , y , or $\mathbb{E}[xy] = 0$, we can call them *orthogonal*, because their inner product is zero. We are now free to apply to our task of interest any one of the theorems that have been derived for Euclidean spaces.

² These operations also satisfy all the properties required for a set to be a vector space, including associativity, commutativity, and so on (see [47] and Section 8.15).

Let us now rewrite (4.1) as

$$\hat{y} = \theta_1 x_1 + \cdots + \theta_l x_l.$$

Thus, the random variable, \hat{y} , which is now interpreted as a point in a vector space, results as a linear combination of l elements in this space. Thus, the estimate, \hat{y} , will necessarily lie in the subspace spanned by these points. In contrast, the true variable, y , will not lie, in general, in this subspace. Because our goal is to obtain a \hat{y} that is a good approximation of y , we have to seek the specific linear combination that makes the norm of the error, $e = y - \hat{y}$, minimum. This specific linear combination corresponds to the *orthogonal* projection of y onto the subspace spanned by the points x_1, x_2, \dots, x_l . This is equivalent with requiring

$$\boxed{\mathbb{E}[e x_k] = 0, \quad k = 1, \dots, l : \text{ Orthogonality Condition.}} \quad (4.10)$$

The error variable being orthogonal to every point x_k , $k = 1, 2, \dots, l$, will be orthogonal to the respective subspace. This is illustrated in Figure 4.3. Such a choice guarantees that the resulting error will have the minimum norm; by the definition of the norm, this corresponds to the minimum MSE, or $\mathbb{E}[e^2]$.

The set of equations in (4.10) can now be written as

$$\mathbb{E}\left[\left(y - \sum_{i=1}^l \theta_i x_i\right) x_k\right] = 0, \quad k = 1, 2, \dots, l,$$

or

$$\sum_{i=1}^l \mathbb{E}[x_i x_k] \theta_i = \mathbb{E}[x_k y], \quad k = 1, 2, \dots, l, \quad (4.11)$$

which leads to the linear set of equations in (4.5).

This is the reason that this elegant set of equations is known as *normal equations*. Another name is *Wiener-Hopf equations*. Strictly speaking, the Wiener-Hopf equations were first derived for continuous time processes in the context of the causal estimation task [49, 50]; for a discussion see [16, 44].

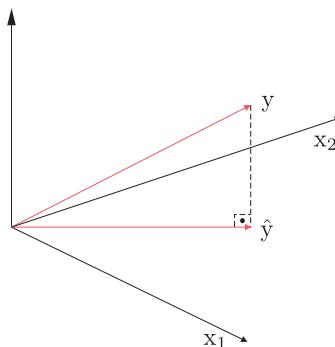


FIGURE 4.3

Projecting y on the subspace spanned by x_1, x_2 guarantees that the deviation between y and \hat{y} corresponds to the minimum MSE.

Nobert Wiener was a mathematician and philosopher. He was awarded a PhD at Harvard at the age of 17 in mathematical logic. During the Second World War, he laid the foundations of linear estimation theory in a classified work, independently of Kolmogorov. Later on, Wiener was involved in pioneering work embracing automation, artificial intelligence, and cognitive science. Being a pacifist, he was regarded with suspicion during the Cold War years.

The other pillar on which linear estimation theory is based is the pioneering work of Andrey Nikolaevich Kolmogorov (1903–1987) [24], who developed his theory independent of Wiener. Kolmogorov's contributions cover a wide range of topics in mathematics, including probability, computational complexity, and topology. He is the father of the modern axiomatic foundation of the notion of probability, see [Chapter 2](#).

Remarks 4.2.

- So far, in our theoretical findings, we have assumed that \mathbf{x} and y are jointly distributed (correlated) variables. If, in addition, we assume that they are linearly related according to the linear regression model,

$$y = \boldsymbol{\theta}_o^T \mathbf{x} + \eta, \quad \boldsymbol{\theta}_o \in \mathbb{R}^k, \quad (4.12)$$

where η is a zero mean noise variable independent of \mathbf{x} , then, if the dimension, k , of the true system, $\boldsymbol{\theta}_o$, is equal to the number of parameters, l , adopted for the model, so that the $k = l$, it turns out that ([Problem 4.4](#))

$$\boldsymbol{\theta}_* = \boldsymbol{\theta}_o,$$

and the optimal MSE is equal to the variance of the noise, σ_η^2 .

- *Undermodeling.* If $k > l$, then the order of the model is less than that of the true system, which relates y and \mathbf{x} in (4.12); this is known as *undermodeling*. It is easy to show that if the variables comprising \mathbf{x} are uncorrelated then ([Problem 4.5](#)),

$$\boldsymbol{\theta}_* = \boldsymbol{\theta}_o^1,$$

where

$$\boldsymbol{\theta}_o := \begin{bmatrix} \boldsymbol{\theta}_o^1 \\ \boldsymbol{\theta}_o^2 \end{bmatrix}, \quad \boldsymbol{\theta}_o^1 \in \mathbb{R}^l, \quad \boldsymbol{\theta}_o^2 \in \mathbb{R}^{k-l}.$$

In other words, the MSE optimal estimator identifies the first l components of $\boldsymbol{\theta}_o$.

4.4 EXTENSION TO COMPLEX-VALUED VARIABLES

Everything that has been said so far can be extended to complex-valued signals. However, there are a few subtle points involved and this is the reason that we chose to treat this case separately. Complex-valued variables are very common in a number of applications, as for example in communications and fMRI [2, 41].

Given two real-valued variables, (x, y) , one can consider them either as a vector quantity in the two-dimensional space, $[x, y]^T$, or can describe them as a complex variable, $z = x + jy$, where $j^2 := -1$.

Adopting the latter approach offers the luxury of exploiting the operations available in the field \mathbb{C} of complex numbers, in other words multiplication and division. The existence of such operations greatly facilitates the algebraic manipulations. Recall that such operations are not defined in vector spaces.³

Let us assume that we are given a complex-valued (output) random variable

$$\mathbf{y} := \mathbf{y}_r + j\mathbf{y}_i, \quad (4.13)$$

and a complex-valued (input) random vector

$$\mathbf{x} = \mathbf{x}_r + j\mathbf{x}_i. \quad (4.14)$$

The quantities y_r, y_i, \mathbf{x}_r , and \mathbf{x}_i are real-valued random variables/vectors. The goal is to compute a linear estimator defined by a complex-valued parameter vector $\boldsymbol{\theta} = \boldsymbol{\theta}_r + j\boldsymbol{\theta}_i \in \mathbb{C}^l$, so as to minimize the respective mean-square error,

$$\mathbb{E}[|\mathbf{e}|^2] := \mathbb{E}[\mathbf{e}\mathbf{e}^*] = \mathbb{E}[|\mathbf{y} - \boldsymbol{\theta}^H \mathbf{x}|^2]. \quad (4.15)$$

Looking at (4.15), it is readily observed that in the case of complex variables the inner product operation between two complex-valued random variables should be defined as $\mathbb{E}[\mathbf{x}\mathbf{y}^*]$, so as to guarantee that the implied norm by the inner product, $\|\mathbf{x}\| = \sqrt{\mathbb{E}[\mathbf{x}\mathbf{x}^*]}$, is a valid quantity. Applying the orthogonality condition as before, we rederive the normal equations as in (4.11),

$$\boldsymbol{\Sigma}_x \boldsymbol{\theta}_* = \mathbf{p}, \quad (4.16)$$

where now the covariance matrix and cross-correlation vector are given by

$$\boldsymbol{\Sigma}_x = \mathbb{E}[\mathbf{x}\mathbf{x}^H], \quad (4.17)$$

$$\mathbf{p} = \mathbb{E}[\mathbf{x}\mathbf{y}^*]. \quad (4.18)$$

Note that (4.16)-(4.18) can alternatively be obtained by minimizing (4.15) (Problem 4.6). Moreover, the counterpart of (4.9) is given by

$$J(\boldsymbol{\theta}_*) = \sigma_y^2 - \mathbf{p}^H \boldsymbol{\Sigma}_x^{-1} \mathbf{p} = \sigma_y^2 - \mathbf{p}^H \boldsymbol{\theta}_*. \quad (4.19)$$

Using the definitions in (4.13) and (4.14), the cost in (4.15) is written as,

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}[|\mathbf{e}|^2] = \mathbb{E}[|\mathbf{y} - \hat{\mathbf{y}}|^2] \\ &= \mathbb{E}[|\mathbf{y}_r - \hat{\mathbf{y}}_r|^2] + \mathbb{E}[|\mathbf{y}_i - \hat{\mathbf{y}}_i|^2], \end{aligned} \quad (4.20)$$

where

$\hat{\mathbf{y}} := \hat{\mathbf{y}}_r + j\hat{\mathbf{y}}_i = \boldsymbol{\theta}^H \mathbf{x} : \text{ Complex Linear Estimator,}$

(4.21)

³ Multiplication and division can also be defined for groups of four variables (x, ϕ, z, y) known as quaternions; the related algebra was introduced by Hamilton in 1843. The real and complex numbers as well as quaternions are all special cases of the so-called Clifford algebras [39].

or

$$\begin{aligned}\hat{\mathbf{y}} &= (\boldsymbol{\theta}_r^T - j\boldsymbol{\theta}_i^T)(\mathbf{x}_r + j\mathbf{x}_i) \\ &= (\boldsymbol{\theta}_r^T \mathbf{x}_r + \boldsymbol{\theta}_i^T \mathbf{x}_i) + j(\boldsymbol{\theta}_r^T \mathbf{x}_i - \boldsymbol{\theta}_i^T \mathbf{x}_r).\end{aligned}\quad (4.22)$$

Equation (4.22) reveals the true flavor behind the complex notation; that is, its *multichannel* nature. In multichannel estimation, we are given more than one set of input variables, namely \mathbf{x}_r and \mathbf{x}_i , and we want to generate, jointly, more than one output variable, namely $\hat{\mathbf{y}}_r$ and $\hat{\mathbf{y}}_i$. Equation (4.22) can equivalently be written as

$$\begin{bmatrix} \hat{\mathbf{y}}_r \\ \hat{\mathbf{y}}_i \end{bmatrix} = \Theta \begin{bmatrix} \mathbf{x}_r \\ \mathbf{x}_i \end{bmatrix}, \quad (4.23)$$

where

$$\Theta := \begin{bmatrix} \boldsymbol{\theta}_r^T & \boldsymbol{\theta}_i^T \\ -\boldsymbol{\theta}_i^T & \boldsymbol{\theta}_r^T \end{bmatrix}. \quad (4.24)$$

Multichannel estimation can be generalized to more than two outputs and to more than two input sets of variables. We will come back to the more general multichannel estimation task toward the end of this chapter.

Looking at (4.23), we observe that starting from the direct generalization of the linear estimation task for real-valued signals, which led to the adoption of $\hat{\mathbf{y}} = \boldsymbol{\theta}^H \mathbf{x}$, resulted in a matrix, Θ , of a *very special structure*.

4.4.1 WIDELY LINEAR COMPLEX-VALUED ESTIMATION

Let us define the linear two-channel estimation task starting from the definition of a linear operation in vector spaces. The task is to generate a vector output, $\hat{\mathbf{y}} = [\hat{\mathbf{y}}_r, \hat{\mathbf{y}}_i]^T \in \mathbb{R}^2$ from the input vector variables, $\mathbf{x} = [\mathbf{x}_r^T, \mathbf{x}_i^T]^T \in \mathbb{R}^{2l}$, via the linear operation,

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{\mathbf{y}}_r \\ \hat{\mathbf{y}}_i \end{bmatrix} = \Theta \begin{bmatrix} \mathbf{x}_r \\ \mathbf{x}_i \end{bmatrix}, \quad (4.25)$$

where

$$\Theta := \begin{bmatrix} \boldsymbol{\theta}_{11}^T & \boldsymbol{\theta}_{12}^T \\ \boldsymbol{\theta}_{21}^T & \boldsymbol{\theta}_{22}^T \end{bmatrix}, \quad (4.26)$$

and compute the matrix Θ so as to minimize the total error variance

$$\Theta_* := \arg \min_{\Theta} \left\{ \mathbb{E}[(\mathbf{y}_r - \hat{\mathbf{y}}_r)^2] + \mathbb{E}[(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2] \right\}. \quad (4.27)$$

Note that (4.27) can equivalently be written as

$$\Theta_* := \arg \min_{\Theta} \left\{ \mathbb{E}[\mathbf{e}^T \mathbf{e}] \right\} = \arg \min_{\Theta} \left\{ \text{trace}[\mathbb{E}[\mathbf{e} \mathbf{e}^T]] \right\},$$

where

$$\mathbf{e} := \mathbf{y} - \hat{\mathbf{y}}.$$

Minimizing (4.27) is equivalent with minimizing the two terms individually; in other words, treating each channel separately (Problem 4.7). Thus, the task can be tackled by solving two sets of normal equations, namely

$$\Sigma_e \begin{bmatrix} \theta_{11} \\ \theta_{12} \end{bmatrix} = \mathbf{p}_r, \quad \Sigma_e \begin{bmatrix} \theta_{21} \\ \theta_{22} \end{bmatrix} = \mathbf{p}_i, \quad (4.28)$$

where

$$\begin{aligned} \Sigma_e &:= \mathbb{E} \left[\begin{bmatrix} \mathbf{x}_r \\ \mathbf{x}_i \end{bmatrix} \begin{bmatrix} \mathbf{x}_r^T & \mathbf{x}_i^T \end{bmatrix} \right] \\ &= \begin{bmatrix} \mathbb{E}[\mathbf{x}_r \mathbf{x}_r^T] & \mathbb{E}[\mathbf{x}_r \mathbf{x}_i^T] \\ \mathbb{E}[\mathbf{x}_i \mathbf{x}_r^T] & \mathbb{E}[\mathbf{x}_i \mathbf{x}_i^T] \end{bmatrix} := \begin{bmatrix} \Sigma_r & \Sigma_{ri} \\ \Sigma_{ir} & \Sigma_i \end{bmatrix}, \end{aligned} \quad (4.29)$$

and

$$\mathbf{p}_r := \mathbb{E} \begin{bmatrix} \mathbf{x}_r \mathbf{y}_r \\ \mathbf{x}_i \mathbf{y}_r \end{bmatrix}, \quad \mathbf{p}_i := \mathbb{E} \begin{bmatrix} \mathbf{x}_r \mathbf{y}_i \\ \mathbf{x}_i \mathbf{y}_i \end{bmatrix}. \quad (4.30)$$

The obvious question that is now raised is whether we can tackle this more general task of the two-channel linear estimation task by employing complex-valued arithmetic. The answer is in the affirmative. Let us define

$$\boldsymbol{\theta} := \boldsymbol{\theta}_r + j\boldsymbol{\theta}_i, \quad \mathbf{v} := \mathbf{v}_r + j\mathbf{v}_i, \quad (4.31)$$

and

$$\mathbf{x} = \mathbf{x}_r + j\mathbf{x}_i.$$

Then define

$$\boldsymbol{\theta}_r := \frac{1}{2}(\boldsymbol{\theta}_{11} + \boldsymbol{\theta}_{22}), \quad \boldsymbol{\theta}_i := \frac{1}{2}(\boldsymbol{\theta}_{12} - \boldsymbol{\theta}_{21}), \quad (4.32)$$

and

$$\mathbf{v}_r := \frac{1}{2}(\mathbf{v}_{11} - \mathbf{v}_{22}), \quad \mathbf{v}_i := -\frac{1}{2}(\mathbf{v}_{12} + \mathbf{v}_{21}). \quad (4.33)$$

Under the previous definitions, it is a matter of simple algebra (Problem 4.8) to prove that the set of equations in (4.25) is equivalent to

$\hat{\mathbf{y}} := \hat{\mathbf{y}}_r + j\hat{\mathbf{y}}_i = \boldsymbol{\theta}^H \mathbf{x} + \mathbf{v}^H \mathbf{x}^* :$ Widely Linear Complex Estimator.

(4.34)

To distinguish from (4.21), this is known as *widely linear* complex-valued estimator. Note that in (4.34), \mathbf{x} as well as its complex conjugate, \mathbf{x}^* , are *simultaneously* used in order to cover all possible solutions, as those are dictated by the vector space description, which led to the formulation in (4.25).

Circularity conditions

We now turn our attention into investigating conditions under which the widely linear formulation in (4.34) breaks down to (4.21); that is, the conditions for which the optimal widely linear estimator turns out to have $\mathbf{v} = \mathbf{0}$.

Let

$$\boldsymbol{\varphi} := \begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{v} \end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{x}} := \begin{bmatrix} \mathbf{x} \\ \mathbf{x}^* \end{bmatrix}. \quad (4.35)$$

Then the widely linear estimator is written as

$$\hat{y} = \boldsymbol{\varphi}^H \tilde{\mathbf{x}}.$$

Adopting the orthogonality condition in its complex formulation

$$\mathbb{E}[\tilde{\mathbf{x}}\mathbf{e}^*] = \mathbb{E}[\tilde{\mathbf{x}}(y - \hat{y})^*] = \mathbf{0},$$

we obtain the following set of normal equations for the optimal $\boldsymbol{\varphi}_*$,

$$\mathbb{E}[\tilde{\mathbf{x}}\tilde{\mathbf{x}}^H]\boldsymbol{\varphi}_* = \mathbb{E}[\tilde{\mathbf{x}}\tilde{\mathbf{x}}^H]\begin{bmatrix} \boldsymbol{\theta}_* \\ \mathbf{v}_* \end{bmatrix} = \begin{bmatrix} \mathbb{E}[\mathbf{x}\mathbf{y}^*] \\ \mathbb{E}[\mathbf{x}^*\mathbf{y}^*] \end{bmatrix},$$

or

$$\begin{bmatrix} \Sigma_x & P_x \\ P_x^* & \Sigma_x^* \end{bmatrix} \begin{bmatrix} \boldsymbol{\theta}_* \\ \mathbf{v}_* \end{bmatrix} = \begin{bmatrix} \mathbf{p} \\ \mathbf{q}^* \end{bmatrix}, \quad (4.36)$$

where Σ_x and \mathbf{p} have been defined in (4.17) and (4.18), respectively, and

$$P_x := \mathbb{E}[\mathbf{x}\mathbf{x}^T], \quad \mathbf{q} := \mathbb{E}[\mathbf{x}\mathbf{y}]. \quad (4.37)$$

The matrix P_x is known as the *pseudo covariance/autocorrelation* matrix of \mathbf{x} . Note that (4.36) is the equivalent of (4.28); to obtain the widely linear estimator, one needs to solve one set of complex-valued equations whose number is double compared to that of the linear (complex) formulation.

Assume now that

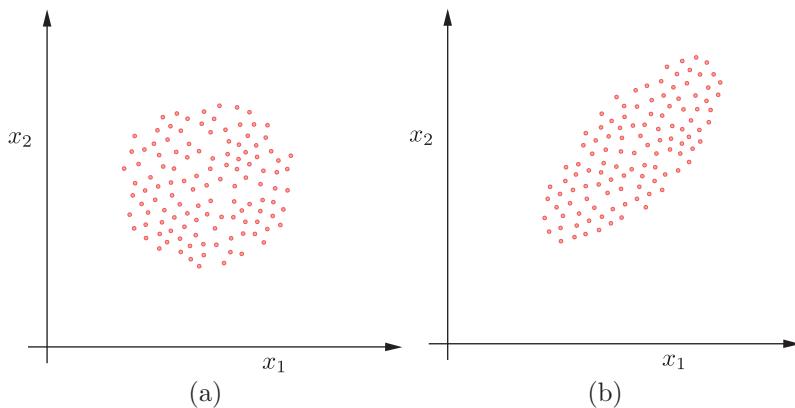
$$P_x = \mathbf{O} \text{ and } \mathbf{q} = \mathbf{0} : \quad \text{Circularity Conditions.} \quad (4.38)$$

We say that in this case, the input-output variables are *jointly circular* and the input variables in \mathbf{x} obey the (second order) *circular* condition. It is readily observed that, under the previous circularity assumptions, (4.36) leads to $\mathbf{v}_* = \mathbf{0}$ and the optimal $\boldsymbol{\theta}_*$ is given by the set of normal equations (4.16)-(4.18), which govern the more restricted linear case. Thus, adopting the linear formulation leads to optimality only under certain conditions, which do not always hold true in practice; a typical such example of variables, which do not respect circularity, are met in fMRI imaging (see [1] and the references therein). It can be shown that the MSE achieved by a widely linear estimator is always less than or equal to that obtained via a linear one (Problem 4.9).

The notions of circularity and of the widely linear estimation were treated in a series of fundamental papers [35, 36]. A stronger condition for circularity is based on the pdf of a complex random variable: A random variable x is circular (or strictly circular) if x and $xe^{j\phi}$ are distributed according to the same pdf; that is, the pdf is *rotationally invariant* [35]. Figure 4.4a shows the scatter plot of points generated by a circularly distributed variable and Figure 4.4b corresponds to a noncircular one. Strict circularity implies the second-order circularity, but the converse is not always true. For more on complex random variables, the interested reader may consult [3, 37]. In Ref. [28], it is pointed out that the full second-order statistics of the error, without doubling the dimension, can be achieved if instead of the MSE one employs the Gaussian entropy criterion.

Finally, note that substituting in (4.29) the second-order circularity conditions, given in (4.38), one obtains (Problem 4.10),

$$\Sigma_r = \Sigma_i, \quad \Sigma_{ri} = -\Sigma_{ir}, \quad \mathbb{E}[\mathbf{x}_r\mathbf{y}_r] = \mathbb{E}[\mathbf{x}_i\mathbf{y}_i], \quad \mathbb{E}[\mathbf{x}_i\mathbf{y}_r] = -\mathbb{E}[\mathbf{x}_r\mathbf{y}_i], \quad (4.39)$$

**FIGURE 4.4**

Scatter plots of points corresponding to (a) a circular process and (b) a noncircular one, in the two-dimensional space.

which then implies that $\theta_{11} = \theta_{22}$, and $\theta_{12} = -\theta_{21}$; in this case, (4.33) verifies that $v = \mathbf{0}$, and that the optimal in the MSE sense solution has the special structure of (4.23) and (4.24).

4.4.2 OPTIMIZING WITH RESPECT TO COMPLEX-VALUED VARIABLES: WIRTINGER CALCULUS

So far, in order to derive the estimates of the parameters, for both the linear as well as the widely linear estimators, the orthogonality condition was mobilized. For the complex linear estimation case, the normal equations were derived in [Problem 4.6](#), by direct minimization of the cost function in (4.20). Those who got involved with solving the problem have experienced a procedure that was more cumbersome compared to the real-valued linear estimation. This is because one has to use the real and imaginary parts of all the involved complex variables and express the cost function in terms of the equivalent real-valued quantities *only*; then the required gradients for the optimization have to be performed. Recall that any complex function $f : \mathbb{C} \rightarrow \mathbb{R}$ is not differentiable with respect to its complex argument, because the Cauchy-Riemann conditions are violated ([Problem 4.11](#)). The previously stated procedure of splitting the involved variables into their real and imaginary parts can become cumbersome with respect to algebraic manipulations. Wirtinger calculus provides an equivalent formulation that is based on simple rules and principles, which bear a great resemblance to the rules of standard complex differentiation.

Let $f : \mathbb{C} \mapsto \mathbb{C}$ be a complex function defined on \mathbb{C} . Obviously, such a function can be regarded as either defined on \mathbb{R}^2 or \mathbb{C} (i.e., $f(z) = f(x + jy) = f(x, y)$). Furthermore, it may be regarded as either complex-valued, $f(x, y) = f_r(x, y) + jf_i(x, y)$ or as vector-valued $f(x, y) = (f_r(x, y), f_i(x, y))$. We say that f is *differentiable in the real sense* if both f_r and f_i are differentiable. Wirtinger's calculus considers the complex structure of f and the real derivatives are described using an equivalent formulation that greatly simplifies calculations; moreover, this formulation bears a surprising similarity with the complex derivatives.

Definition 4.1. The *Wirtinger derivative* or *W-derivative* of a complex function f at a point $z_0 \in \mathbb{C}$ is defined as

$$\frac{\partial f}{\partial z}(z_0) = \frac{1}{2} \left(\frac{\partial f_r}{\partial x}(z_0) + \frac{\partial f_i}{\partial y}(z_0) \right) + \frac{j}{2} \left(\frac{\partial f_i}{\partial x}(z_0) - \frac{\partial f_r}{\partial y}(z_0) \right) : \text{ W-derivative.}$$

The *Conjugate Wirtinger's derivative* or *CW-derivative* of f at z_0 is defined as

$$\frac{\partial f}{\partial z^*}(z_0) = \frac{1}{2} \left(\frac{\partial f_r}{\partial x}(z_0) - \frac{\partial f_i}{\partial y}(z_0) \right) + \frac{j}{2} \left(\frac{\partial f_i}{\partial x}(z_0) + \frac{\partial f_r}{\partial y}(z_0) \right) : \text{ CW-derivative.}$$

For some of the properties and the related proofs regarding Wirtinger's derivatives see [Appendix A.3](#). An important property for us is that if f is real-valued (i.e., $\mathbb{C} \mapsto \mathbb{R}$) and z_0 is a (local) optimal point of f , it turns out that

$$\frac{\partial f}{\partial z}(z_0) = \frac{\partial f}{\partial z^*}(z_0) = 0 : \text{ Optimality Conditions.} \quad (4.40)$$

In order to apply Wirtinger's derivatives, the following simple *tricks* are adopted:

- express function f in terms of z and z^* ;
- to compute W-derivative apply the usual differentiation rule, treating z^* as a constant;
- to compute CW-derivative apply the usual differentiation rule, treating z as a constant.

It should be emphasized that all these statements must be regarded as useful computational tricks rather than rigorous mathematical rules. Analogous definitions and properties carry on for complex vectors z , and the W-gradient and CW-gradients

$$\nabla_z f(z_0), \quad \nabla_{z^*} f(z_0),$$

result from the respective definitions if partial derivatives are replaced by partial gradients, ∇_x, ∇_y .

Although Wirtinger's calculus has been known since 1927 [51], its use in applications has a rather recent history [7] and its revival was ignited by the widely linear filtering concept [27]. The interested reader may obtain more on this issue from [2, 25, 30]. Extensions of Wirtinger's derivative to general Hilbert (infinite dimensional) spaces was done more recently in [6] and to the subgradient notion in [46].

Application in Linear Estimation. The cost function in this case is

$$J(\boldsymbol{\theta}, \boldsymbol{\theta}^*) = \mathbb{E} \left[|y - \boldsymbol{\theta}^H \mathbf{x}|^2 \right] = \mathbb{E} \left[(y - \boldsymbol{\theta}^H \mathbf{x}) (y^* - \boldsymbol{\theta}^T \mathbf{x}^*) \right].$$

Thus, treating $\boldsymbol{\theta}$ as a constant, the optimal occurs at

$$\nabla_{\boldsymbol{\theta}^*} J = \mathbb{E}[\mathbf{x} \mathbf{e}^*] = \mathbf{0},$$

which is the orthogonality condition leading to the normal equations (4.16)-(4.18).

Application in Widely Linear Estimation. The cost function is now (see notation in (4.35))

$$J(\boldsymbol{\varphi}, \boldsymbol{\varphi}^*) = \mathbb{E} \left[(y - \boldsymbol{\varphi}^H \tilde{\mathbf{x}}) (y^* - \boldsymbol{\varphi}^T \tilde{\mathbf{x}}^*) \right],$$

and treating ϕ as a constant,

$$\nabla_{\theta^*} J = \mathbb{E}[\tilde{\mathbf{x}}\mathbf{e}^*] = \mathbb{E} \begin{bmatrix} \mathbf{x}\mathbf{e}^* \\ \mathbf{x}^*\mathbf{e}^* \end{bmatrix} = \mathbf{0},$$

which leads to the set derived in (4.36).

Wirtinger's calculus will prove very useful in subsequent chapters for deriving gradient operations in the context of online/adaptive estimation in Euclidean as well as in reproducing kernel Hilbert spaces.

4.5 LINEAR FILTERING

Linear statistical filtering is an instance of the general estimation task, when the notion of time evolution needs to be taken into consideration and estimates are obtained at each time instant. There are three major types of problems that emerge:

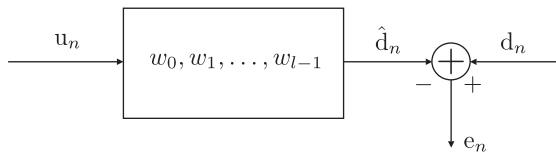
- *Filtering*, where the estimate at time instant n is based on all previously received (measured) input information *up to and including* the current time index, n .
- *Smoothing*, where data over a time interval, $[0, N]$, are first collected and an estimate is obtained at each time instant $n \leq N$, using *all* the available information in the interval $[0, N]$.
- *Prediction*, where estimates at times $n + \tau$, $\tau > 0$ are to be obtained based on the information up to and including time instant n .

To fit in the above definitions more with what has been said so far in the chapter, take for example a time-varying case, where the output variable at time instant n is y_n and its value depends on observations included in the corresponding input vector \mathbf{x}_n . In filtering, the latter can include measurements received only at time instants $n, n-1, \dots, 0$. This restriction in the index set is directly related to *causality*. In contrast, in smoothing, we can also include future time instants $n+2, n+1, n, n-1$.

Most of the effort in this book will be spent on filtering whenever time information enters into the picture. The reason is that this is the most commonly encountered task and, also, the techniques used for smoothing and prediction are similar in nature with that of filtering, with usually minor modifications.

In signal processing, the term filtering is usually used in a more specific context, and it refers to the operation of a *filter*, which acts on an input random process/signal (u_n), to transform it into another one (d_n), see [Section 2.4.3](#). Note that we have switched into the notation, introduced in [Chapter 2](#), used to denote random processes. We prefer to keep different notation for processes and random variables, because in the case of random processes, the filtering task obtains a special structure and properties, as we will soon see. Moreover, although the mathematical formulation of the involved equations, for both cases, may end up to be the same, we feel that it is good for the reader to keep in mind that there is a different underlying mechanism for generating the data.

The task in statistical linear filtering is to compute the coefficients (impulse response) of the filter so that the output process of the filter, \hat{d}_n , when the filter is excited by the input random process, u_n , to be as close as possible to a *desired* response process, d_n . In other words, the goal is to minimize, in some sense, the corresponding error processes, see [Figure 4.5](#). Assuming that the unknown filter is of

**FIGURE 4.5**

In statistical filtering, the impulse response coefficients are estimated so as to minimize the error between the output and the desired response processes. In MSE linear filtering, the cost function is $\mathbb{E}[e_n^2]$.

a finite impulse response (FIR) (see [Section 2.4.3](#) for related definitions), denoted as w_0, w_1, \dots, w_{l-1} , the output \hat{d}_n of the filter is given as

$$\hat{d}_n = \sum_{i=0}^{l-1} w_i u_{n-i} = \mathbf{w}^T \mathbf{u}_n : \text{ Convolution Sum,} \quad (4.41)$$

where

$$\mathbf{w} = [w_0, w_1, \dots, w_{l-1}]^T, \quad \text{and} \quad \mathbf{u}_n = [u_n, u_{n-1}, \dots, u_{n-l+1}]^T. \quad (4.42)$$

[Figure 4.6](#) illustrates the convolution operation of the linear filter, when the input is excited by a realization u_n of the input processes to provide in the output the signal/sequence \hat{d}_n .

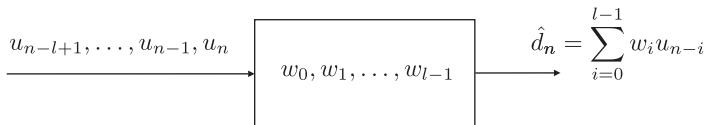
Alternatively, (4.41) can be viewed as the *linear estimator* function; given the jointly distributed variables, at time instant n , (d_n, \mathbf{u}_n) , (4.41) provides the estimator, \hat{d}_n , given the values of \mathbf{u}_n . In order to obtain the coefficients, \mathbf{w} , the mean-square error criterion will be adopted. Furthermore, we will assume that

- The processes, u_n, d_n are *wide-sense stationary* real random processes.
- Their mean values are equal to zero, in other words, $\mathbb{E}[u_n] = \mathbb{E}[d_n] = 0, \forall n$. If this is not the case, we can subtract the respective mean values from the processes, u_n and d_n , during a preprocessing stage. Due to this assumption, the autocorrelation and covariance matrices of \mathbf{u}_n coincide, so that

$$R_u = \Sigma_u.$$

The normal equations in (4.5) now take the form

$$\Sigma_u \mathbf{w} = \mathbf{p},$$

**FIGURE 4.6**

The linear filter is excited by a realization of an input process. The output signal is the convolution between the input sequence and the filter's impulse response.

where

$$\mathbf{p} = [\mathbb{E}[u_n d_n], \dots, \mathbb{E}[u_{n-l+1} d_n]]^T,$$

and the respective covariance/autocorrelation matrix, of order l , of the input process is given by

$$\Sigma_u := \mathbb{E}[\mathbf{u}_n \mathbf{u}_n^T] = \begin{bmatrix} r(0) & r(1) & \dots & r(l-1) \\ r(1) & r(0) & \dots & r(l-2) \\ \vdots & & \ddots & \\ r(l-1) & r(l-2) & \dots & r(0) \end{bmatrix}, \quad (4.43)$$

where $r(k)$ is the autocorrelation sequence of the input process. Because we have assumed that the involved processes are wide-sense stationary, we have that

$$r(n, n-k) := \mathbb{E}[u_n u_{n-k}] = r(k).$$

Also, recall that, for real wide-sense stationary processes, the autocorrelation sequence is symmetric, or $r(k) = r(-k)$ (Section 2.4.3). Observe that in this case, where the input vector results from a random process, the covariance matrix has a special structure, which will be exploited later on to derive efficient schemes for the solution of the normal equations.

For the complex linear filtering case, the only differences are

- The output is given as, $\hat{d}_n = \mathbf{w}^H \mathbf{u}_n$,
- $\mathbf{p} = \mathbb{E}[\mathbf{u}_n d_n^*]$,
- $\Sigma_u = \mathbb{E}[\mathbf{u}_n \mathbf{u}_n^H]$,
- $r(-k) = r^*(k)$.

4.6 MSE LINEAR FILTERING: A FREQUENCY DOMAIN POINT OF VIEW

Let us now turn our attention to the more general case, and assume that our filter is of *infinite impulse response* (IIR). Then, (4.41) now becomes

$$\hat{d}_n = \sum_{i=-\infty}^{+\infty} w_i u_{n-i}. \quad (4.44)$$

Moreover, we have allowed the filter to be *noncausal*.⁴ Following similar arguments as those used to prove the MSE optimality of $\mathbb{E}[y|x]$ in Section 3.15, it turns out that the optimal filter coefficients must satisfy the following condition, (Problem 4.12),

$$\mathbb{E} \left[(d_n - \sum_{i=-\infty}^{+\infty} w_i u_{n-i}) u_{n-j} \right] = 0, \quad j \in \mathbb{Z}. \quad (4.45)$$

⁴ A system is called *causal* if the output, \hat{d}_n , does depend *only* on input values u_m , $m \leq n$. A necessary and sufficient condition for causality is that the impulse response is zero for negative time instants, meaning that $w_n = 0$, $n < 0$. This can easily be checked out; try it.

Observe that this is a generalization (involving an infinite number of terms) of the orthogonality condition stated in (4.10). A rearrangement of the terms in (4.45) results in

$$\sum_{i=-\infty}^{+\infty} w_i \mathbb{E}[u_{n-i} u_{n-j}] = \mathbb{E}[d_n u_{n-j}], \quad j \in \mathbb{Z}, \quad (4.46)$$

and finally to

$$\sum_{i=-\infty}^{+\infty} w_i r(j-i) = r_{du}(j), \quad j \in \mathbb{Z}. \quad (4.47)$$

Equation (4.47) can be considered as the generalization of (4.5) to the case of random processes. The problem now is how one can solve (4.47). The way out is to cross into the frequency domain. Equation (4.47) can be seen as the convolution of the unknown sequence with the autocorrelation sequence of the input process, which gives rise to the cross-correlation sequence. However, we know that convolution of two sequences corresponds to the product of the respective Fourier transforms (e.g., [42]). Thus, we can now write that

$$W(\omega) S_u(\omega) = S_{du}(\omega), \quad (4.48)$$

where $W(\omega)$ is the Fourier transform of the sequence of the unknown parameters, and $S_u(\omega)$ is the *power spectral density* of the input process, defined in [Section 2.4.3](#). In analogy, the Fourier transform $S_{du}(\omega)$ of the cross-correlation sequence is known as the *cross-spectral density*. If the latter two quantities are available, then once $W(\omega)$ has been computed, the unknown parameters can be obtained via the inverse Fourier transform.

Deconvolution: image deblurring

We will now consider an important application in order to demonstrate the power of MSE linear estimation. Image deblurring is a typical *deconvolution* task. An image is degraded due to its transmission via a nonideal system; the task of deconvolution is to optimally recover (in the MSE sense in our case), the original undegraded image. [Figure 4.7a](#) shows the original image and [4.7b](#) a blurred version (e.g., taken by a nonsteady camera) with some small additive noise.

At this point, it is interesting to recall that deconvolution is a process that our human brain performs all the time. The human (and not only) vision system is one of the most complex and highly developed biological systems that has been formed over millions years of a continuous evolution process. Any raw image that falls on the retina of the eye is *severely blurred*. Thus, one of the main early processing activities of our visual system is to deblur it (see, e.g., [29] and the references therein for a related discussion).

Before we proceed any further, the following assumptions are adopted:

- The image is a *wide-sense stationary* two-dimensional random process. Two-dimensional random processes are also known as *random fields*, see [Chapter 15](#).
- The image is of an infinite extent; this can be justified for the case of large images. This assumption will grant us the “permission” to use (4.48). The fact that an image is a two-dimensional process does not change anything in the theoretical analysis; the only difference is that now the Fourier transforms involve two frequency variables, ω_1, ω_2 , one for each of the two dimensions.

**FIGURE 4.7**

(a) The original image and (b) its blurred and noisy version.

A gray image is represented as a two-dimensional array. To stay close to the notation used so far, let $d(n, m)$, $n, m \in \mathbb{Z}$ be the original undegraded image (which for us is now the desired response), and $u(n, m)$, $n, m \in \mathbb{Z}$ be the degraded one, obtained as

$$u(n, m) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} h(i, j)d(n - i, m - j) + \eta(n, m), \quad (4.49)$$

where $\eta(n, m)$ is the realization of a noise field, which is assumed to be zero mean and independent of the input (undegraded) image. The sequence $h(i, j)$ is the *point spread sequence* (impulse response) of the system (e.g., camera). We will assume that this is known and it has, somehow, been measured.⁵

Our task now is to estimate a two-dimensional filter, $w(n, m)$, which is applied to the degraded image to optimally reconstruct (in the MSE sense) the original undegraded image. In the current context, Eq. (4.48) is written as

$$W(\omega_1, \omega_2)S_u(\omega_1, \omega_2) = S_{du}(\omega_1, \omega_2).$$

Following similar arguments as those used to derive Eq. (2.130) of Chapter 2, it is shown that (Problem 4.13)

$$S_{du}(\omega_1, \omega_2) = H^*(\omega_1, \omega_2)S_d(\omega_1, \omega_2), \quad (4.50)$$

and

$$S_u(\omega_1, \omega_2) = |H(\omega_1, \omega_2)|^2 S_d(\omega_1, \omega_2) + S_\eta(\omega_1, \omega_2), \quad (4.51)$$

⁵ Note that this is not always the case.

where “ $*$ ” denotes complex conjugation and S_η is the power spectral density of the noise field. Thus, we finally obtain that

$$W(\omega_1, \omega_2) = \frac{1}{H(\omega_1, \omega_2)} \frac{|H(\omega_1, \omega_2)|^2}{|H(\omega_1, \omega_2)|^2 + \frac{S_\eta(\omega_1, \omega_2)}{S_d(\omega_1, \omega_2)}}. \quad (4.52)$$

Once $W(\omega_1, \omega_2)$ has been computed, the unknown parameters could be obtained via an inverse (two-dimensional) Fourier transform. The deblurred image then results as

$$\hat{d}(n, m) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} w(i, j) u(n-i, m-j). \quad (4.53)$$

In practice, because we are not really interested in obtaining the weights of the deconvolution filter, we implement (4.53) in the frequency domain

$$\hat{D}(\omega_1, \omega_2) = W(\omega_1, \omega_2) U(\omega_1, \omega_2),$$

and then obtain the inverse Fourier transform. Thus, all the processing is efficiently performed in the frequency domain. Software packages to perform Fourier transforms (via the Fast Fourier Transform, FFT) of an image array are “omnipresent” on the internet.

Another important issue is that in practice we do not know $S_d(\omega_1, \omega_2)$. An approximation, which is usually adopted that renders sensible results, is to assume that $\frac{S_\eta(\omega_1, \omega_2)}{S_d(\omega_1, \omega_2)}$ is a constant, C , and try different values of it. [Figure 4.8](#) shows the deblurred image for $C = 2.3 \times 10^{-6}$. The quality of the end result depends a lot on the choice of this value ([MATLAB exercise 4.25](#)). Other, more advanced,

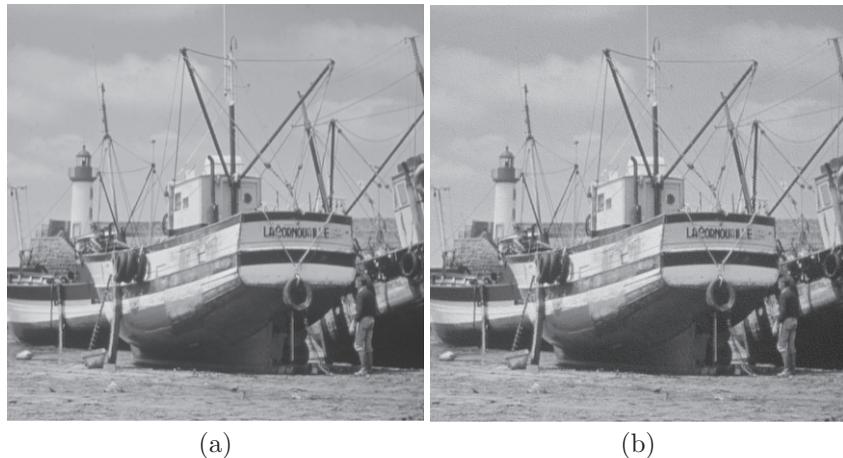


FIGURE 4.8

(a) The original image and (b) the deblurred one for $C = 2.3 \times 10^{-6}$. Observe that in spite of the simplicity of the method, the reconstruction is pretty good. The differences become more obvious to the eye when the images are enlarged.

techniques, have also been proposed. For example, one can get a better estimate of $S_d(\omega_1, \omega_2)$ by using information from $S_\eta(\omega_1, \omega_2)$ and $S_u(\omega_1, \omega_2)$. The interested reader can obtain more on the image deconvolution/restoration task from Refs. [14, 34].

4.7 SOME TYPICAL APPLICATIONS

Optimal linear estimation/filtering has been applied in a wide range of diverse applications of statistical learning, such as regression modeling, communications, control, biomedical signal processing, seismic signal processing, image processing. In the sequel, we present some typical applications in order for the reader to grasp the main rationale of how the previously stated theory can find its way in solving practical problems. In all cases, wide-sense stationarity of the involved random processes is assumed.

4.7.1 INTERFERENCE CANCELLATION

In interference cancellation, we have access to a mixture of two signals expressed as $d_n = y_n + s_n$. Ideally, we would like to remove one of them, say y_n . We will consider them as realizations of respective random processes/signals, or d_n , y_n and s_n . To achieve this goal, the only available information is another signal, say u_n , that is statistically related to the unwanted signal, y_n . For example, y_n may be a filtered version of u_n . This is illustrated in Figure 4.9, where the corresponding realizations of the involved random processes are shown.

Process y_n is the output of an unknown system H , whose input is excited by u_n . The task is to model H by obtaining estimates of its impulse response (assuming that it is LTI and of known order). Then, the output of the model will be an approximation of y_n , when this is activated by the same input, u_n . We will use d_n as the desired response process. The optimal estimates of w_0, \dots, w_{l-1} (assuming the order of the unknown system H to be l) are provided by the normal equations

$$\Sigma_u w_* = p.$$

However,

$$\begin{aligned} p &= \mathbb{E} [\mathbf{u}_n d_n] = \mathbb{E} [\mathbf{u}_n (y_n + s_n)] \\ &= \mathbb{E} [\mathbf{u}_n y_n], \end{aligned} \quad (4.54)$$

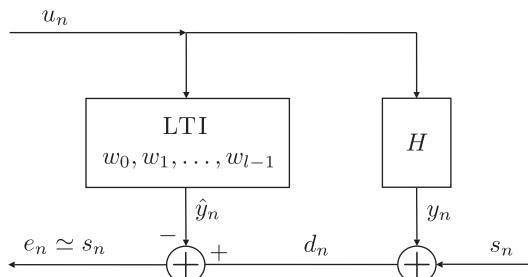
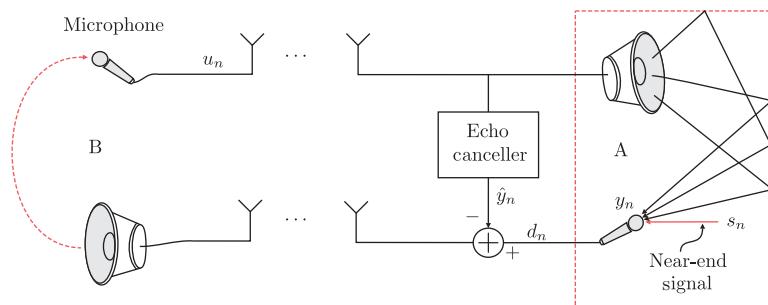


FIGURE 4.9

A basic block diagram illustrating the interference cancellation task.

**FIGURE 4.10**

The echo canceller is optimally designed to remove the part of the far-end signal, u_n , that interferes with the near-end signal, s_n .

because the respective input vector \mathbf{u}_n and \mathbf{s}_n are considered statistically independent. That is, the previous formulation of the problem leads to the same normal equations as if the desired response was the signal \mathbf{y}_n , which we want to remove! Hence, the output of our model will be an approximation (in the MSE sense), $\hat{\mathbf{y}}_n$, of \mathbf{y}_n , and if subtracted from \mathbf{d}_n the resulting (error) signal, \mathbf{e}_n , will be an approximation to \mathbf{s}_n . How good this approximation is depends on whether l is a good “estimate” of the true order of H . The cross-correlation in the right-hand side of (4.54) can be approximated by computing the respective sample mean values, in particular over periods where \mathbf{s}_n is absent. In practical systems, online/adaptive versions of this implementation are usually employed, as we will see [Chapter 5](#).

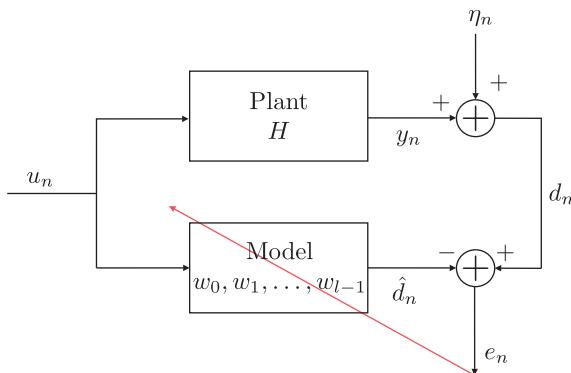
Interference cancellation schemes have been widely used in many systems such as noise cancellation, echo cancellation in telephone networks and video conferencing, and in biomedical applications; for example, in order to cancel the maternal interference in a fetal electrocardiograph.

[Figure 4.10](#) illustrates the echo cancellation task in a video conference application. The same set up applies to the hands-free telephone service in a car. The *far-end* speech signal is considered to be a realization u_n of a random process, u_n ; through the loudspeakers, it is broadcasted in room A (car) and it is reflected in the interior of the room. Part of it is absorbed and part of it enters the microphone; this is denoted as y_n . The equivalent response of the room (reflections) on u_n can be represented by a filter, H , as in [Figure 4.9](#). Signal y_n returns back and the speaker in location B listens to her or his own voice, together with the *near-end* speech signal, s_n of the speaker in A. In certain cases, this feedback path from the loudspeakers to the microphone can cause instabilities giving rise to a “howling” sound effect. The goal of the echo canceller is to optimally remove y_n .

4.7.2 SYSTEM IDENTIFICATION

System identification is similar in nature to the interference cancellation task. Note that in [Figure 4.9](#), one basically models the unknown system. However, the focus there was on replicating the output y_n and not on the system’s impulse response.

In system identification, the aim is to model the impulse response of an unknown plant. To this end, we have access to its input signal as well as to a *noisy* version of its output. The task is to design a model whose impulse response approximates that of the unknown plant. To achieve this, we optimally design

**FIGURE 4.11**

In system identification, the impulse response of the model is optimally estimated so that the output is close, in the MSE, to that of the unknown plant. The red line indicates that the error is used for the optimal estimation of the unknown parameters of the filter.

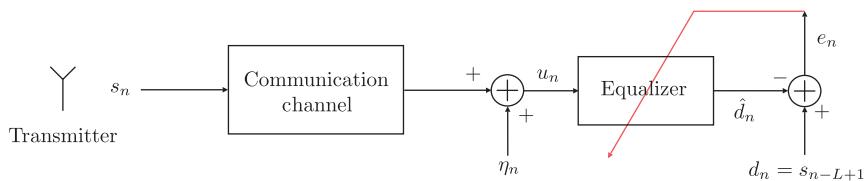
a linear filter whose input is the same signal as the one that activates the plant and its desired response is the noisy output of the plant, see [Figure 4.11](#). The associated normal equations are

$$\Sigma_u \mathbf{w}_* = \mathbb{E}[\mathbf{u}_n \mathbf{d}_n] = \mathbb{E}[\mathbf{u}_n y_n] + 0,$$

assuming the noise η_n is statistically independent of \mathbf{u}_n . Thus, once more, the resulting normal equations are the same as if we had provided the model with a desired response equal to the noiseless output of the unknown plant, expressed as $\mathbf{d}_n = y_n$. Hence, the impulse response of the model is estimated so that its output is close, in the MSE, to the true (noiseless) output of the unknown plant. System identification is of major importance in a number of applications. In control, it is used for driving the associated controllers. In data communications, for estimating the transmission channel in order to build up maximum likelihood estimators of the transmitted data. In many practical systems, adaptive versions of the system identification scheme are implemented, as we will discuss in following chapters.

4.7.3 DECONVOLUTION: CHANNEL EQUALIZATION

Note that in the cancellation task the goal was to “remove” the (filtered version) of the input signal (\mathbf{u}_n) to the unknown system H . In system identification, the focus was on the (unknown) system itself. In *deconvolution*, the emphasis is on the input of the unknown system. That is, our goal now is to recover, in the MSE optimal sense, the (delayed) input signal, \mathbf{u}_{n-L} , where L is the delay in units of the sampling period, T . The task is also called *inverse system identification*. The term *equalization* or *channel equalization* is used in communications. The deconvolution task was introduced in the context of image deblurring in [Section 4.6](#). There, the required information about the *unknown* input process was obtained via an approximation. In the current framework, this can be approached via the transmission of a training sequence.

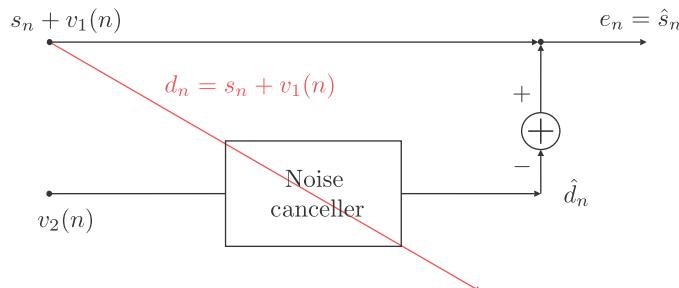
**FIGURE 4.12**

The task of an equalizer is to optimally recover the originally transmitted information sequence, s_n , delayed by L time lags.

The goal of an *equalizer* is to recover the transmitted information symbols, by mitigating the so-called *intersymbol interference* (ISI) that any (imperfect) dispersive communication channel imposes on the transmitted signal; besides ISI, additive noise is also present in the transmitted information bits (see [Example 4.2](#)). Equalizers are “omnipresent” in these days; in our mobile phones, in our modems, etc. [Figure 4.12](#) presents the basic scheme for an equalizer. The equalizer is trained so that its output is as close as possible to the transmitted data bits delayed by some time lag L ; the delay is used in order to account for the overall delayed imposed by the channel-equalizer system. Deconvolution/channel equalization is at the heart of a number of applications besides communications, such as acoustics, optics, seismic signal processing, and control. The channel equalization task will also be discussed in the next chapter in the context of online learning via the decision feedback equalization mode of operation.

Example 4.1 (Noise Cancellation). The noise cancellation application is illustrated in [Figure 4.13](#). The signal of interest is a realization of a process, s_n , that is contaminated by the noise sequence $v_1(n)$. For example, s_n may be the speech signal of the pilot in the cockpit and $v_1(n)$ the aircraft noise at the location of the microphone. We assume that $v_1(n)$ is an AR process of order one, expressed as

$$v_1(n) = a_1 v_1(n - 1) + \eta_n.$$

**FIGURE 4.13**

A block diagram for a noise canceller. Using as desired response the contaminated signal, the output of the optimal filter is an estimate of the noise component.

The signal $v_2(n)$ is a noise sequence,⁶ which is related to $v_1(n)$, but it is statistically independent of s_n . For example, it may be the noise picked up from another microphone positioned at a nearby location. This is also assumed to be an AR process of the first order,

$$v_2(n) = a_2 v_2(n-1) + \eta_n.$$

Note that both $v_1(n)$ and $v_2(n)$ are generated by the same noise source, η_n , that is assumed to be white of variance σ_η^2 . For example, in an aircraft it can be assumed that the noise at different points is due to a “common” source, especially for nearby locations.

The goal of the example is to compute estimates of the weights of the noise canceller, in order to optimally remove (in the MSE sense) the noise $v_1(n)$ from the mixture $s_n + v_1(n)$. Assume the canceller to be of order two.

The input to the canceller is $v_2(n)$ and as desired response the mixture signal, $d_n = s_n + v_1(n)$, will be used. To establish the normal equations, we need to compute the covariance matrix, Σ_2 , of $v_2(n)$ and the cross-correlation vector, p_2 , between the input random vector, $v_2(n)$, and d_n .

Because $v_2(n)$ is an AR process of the first order, recall from [Section 2.4.4](#) that, the autocorrelation sequence is given by

$$r_2(k) = \frac{a_2^k \sigma_\eta^2}{1 - a_2^2}, \quad k = 0, 1, \dots \quad (4.55)$$

Hence,

$$\Sigma_2 = \begin{bmatrix} r_2(0) & r_2(1) \\ r_2(1) & r_2(0) \end{bmatrix} = \begin{bmatrix} \frac{\sigma_\eta^2}{1 - a_2^2} & \frac{a_2 \sigma_\eta^2}{1 - a_2^2} \\ \frac{a_2 \sigma_\eta^2}{1 - a_2^2} & \frac{\sigma_\eta^2}{1 - a_2^2} \end{bmatrix}.$$

Next, we are going to compute the cross-correlation vector.

$$\begin{aligned} p_2(0) &:= \mathbb{E}[v_2(n)d_n] = \mathbb{E}[v_2(n)(s_n + v_1(n))] \\ &= \mathbb{E}[v_2(n)v_1(n)] + 0 = \mathbb{E}[(a_2 v_2(n-1) + \eta_n)(a_1 v_1(n-1) + \eta_n)] \\ &= a_2 a_1 p_2(0) + \sigma_\eta^2, \end{aligned}$$

or

$$p_2(0) = \frac{\sigma_\eta^2}{1 - a_2 a_1}. \quad (4.56)$$

We used the fact that $\mathbb{E}[v_2(n-1)\eta_n] = \mathbb{E}[v_1(n-1)\eta_n] = 0$, because $v_2(n-1)$ and $v_1(n-1)$ depend recursively on previous values, i.e., $\eta(n-1), \eta(n-2), \dots$, and also η_n is a white noise sequence, hence the respective correlation values are zero. Also, due to stationarity, $\mathbb{E}[v_2(n)v_1(n)] = \mathbb{E}[v_2(n-1)v_1(n-1)]$.

⁶ We use the index n in parenthesis to unclutter notation due to the presence of a second subscript.

For the other value of the cross-correlation vector we have

$$\begin{aligned} p_2(1) &= \mathbb{E}[v_2(n-1)d_n] = \mathbb{E}[v_2(n-1)(v_1(n) + \eta_n)] \\ &= \mathbb{E}[v_2(n-1)v_1(n)] + 0 = \mathbb{E}[v_2(n-1)(a_1v_1(n-1) + \eta_n)] \\ &= a_1p_2(0) = \frac{a_1\sigma_\eta^2}{1 - a_1a_2}. \end{aligned}$$

In general, it is easy to show that

$$p_2(k) = \frac{a_1^k\sigma_\eta^2}{1 - a_2a_1}, \quad k = 0, 1, \dots \quad (4.57)$$

Recall that because the processes are real-valued, the covariance matrix is symmetric, meaning $r_2(k) = r_2(-k)$. Also, for (4.55) to make sense, $(r_2(0) > 0)$, $|a_2| < 1$. The same holds true for $|a_1|$, following similar arguments for the autocorrelation process of $v_1(n)$.

Thus, the optimal weights of the noise canceller are given by the following set of normal equations,

$$\begin{bmatrix} \frac{\sigma_\eta^2}{1 - a_2^2} & \frac{a_2\sigma_\eta^2}{1 - a_2^2} \\ \frac{a_2\sigma_\eta^2}{1 - a_2^2} & \frac{\sigma_\eta^2}{1 - a_2^2} \end{bmatrix} \mathbf{w} = \begin{bmatrix} \frac{\sigma_\eta^2}{1 - a_1a_2} \\ \frac{a_1\sigma_\eta^2}{1 - a_1a_2} \end{bmatrix}.$$

Note that the canceller optimally “removes” from the mixture, $s_n + v_1(n)$, the component that is correlated to the input, $v_2(n)$; observe that $v_1(n)$ basically acts as the desired response.

[Figure 4.14a](#) shows a realization of the signal $d_n = s_n + v_1(n)$, where $s_n = \cos(\omega_0 n)$ with $\omega_0 = 2 * 10^{-3} * \pi$, $a_1 = 0.8$, and $\sigma_\eta^2 = 0.05$. [Figure 4.14b](#) is the respective realization of the signal $s_n + v_1(n) - \hat{d}(n)$ for $a_2 = 0.75$. The corresponding weights for the canceller are $\mathbf{w}_* = [1, 0.125]^T$. [Figure 4.14c](#) corresponds to $a_2 = 0.5$. Observe that the higher the cross-correlation between $v_1(n)$ and $v_2(n)$ the better the obtained result becomes.

Example 4.2 (Channel Equalization). Consider the channel equalization set up in [Figure 4.12](#), where the output of the channel, which is sensed by the receiver, is given by

$$\mathbf{u}_n = 0.5s_n + s_{n-1} + \eta_n. \quad (4.58)$$

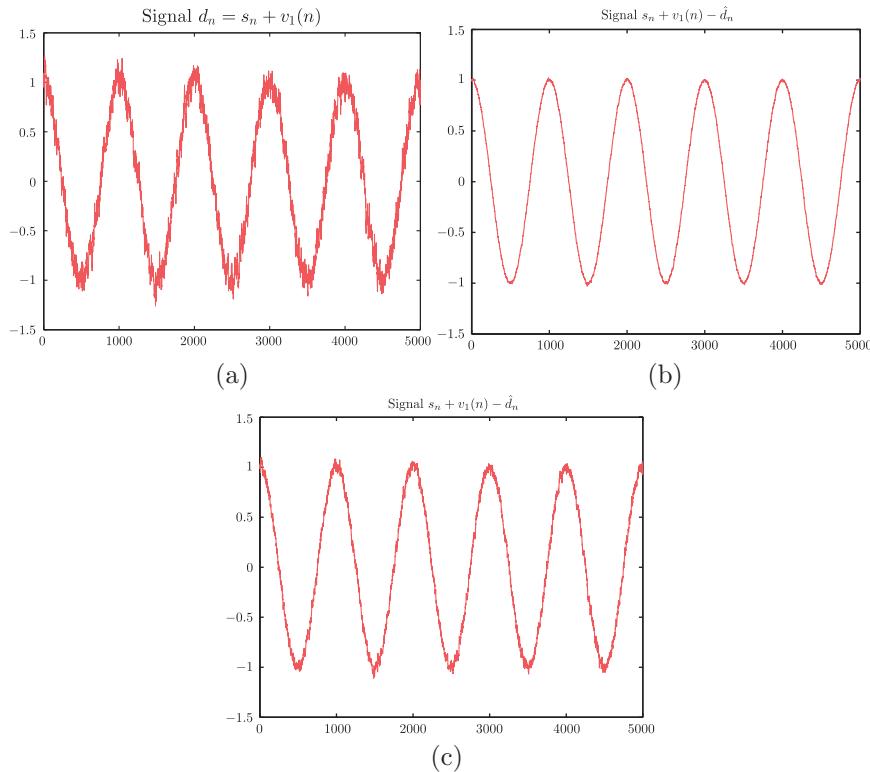
The goal is to design an equalizer comprising three taps, $\mathbf{w} = [w_0, w_1, w_2]^T$, so that

$$\hat{d}_n = \mathbf{w}^T \mathbf{u}_n,$$

and estimate the unknown taps using as a desired response sequence $d_n = s_{n-1}$. We are given that $\mathbb{E}[s_n] = \mathbb{E}[\eta_n] = 0$ and

$$\Sigma_s = \sigma_s^2 I, \quad \Sigma_\eta = \sigma_\eta^2 I.$$

Note that for the desired response we have used a delay $L = 1$. In order to better understand the reason that a delay is used and without going into many details (for the more experienced reader, note that the channel is nonminimum phase, e.g., [41]) observe that at time n , most of the contribution to \mathbf{u}_n in (4.58) comes from the symbol s_{n-1} , which is weighted by one, while the sample s_n is weighted

**FIGURE 4.14**

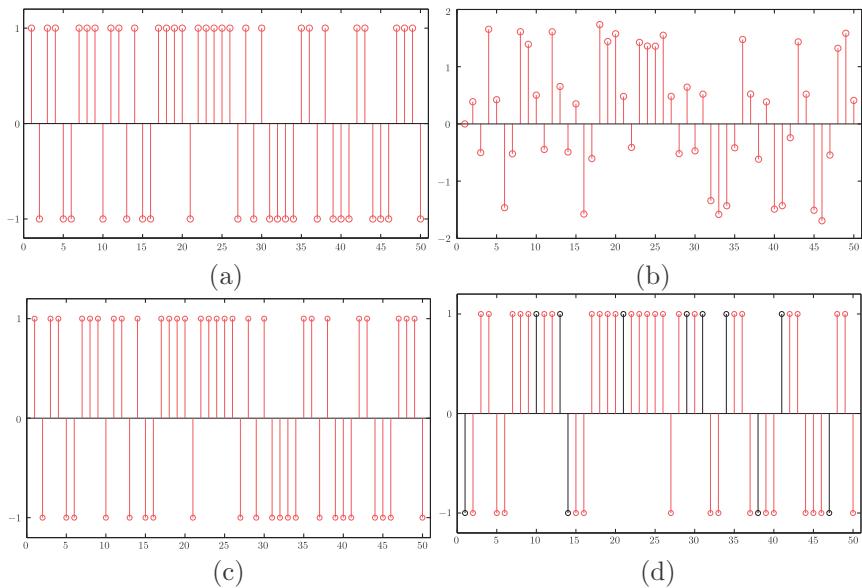
(a) The noisy sinusoid signal of [Example 4.1](#). (b) The de-noised signal for strongly correlated noise sources, v_1 and v_2 . (c) The obtained de-noised signal for less correlated noise sources.

by 0.5; hence, it is most natural from an intuitive point of view, at time n , having received u_n , to try to obtain an estimate for s_{n-1} . This justifies the use of the delay.

[Figure 4.15a](#) shows a realization of the input information sequence s_n . It consists of equiprobable ± 1 samples, randomly generated. The effect of the channel is (a) to combine successive information samples together (ISI) and (b) to add noise; the purpose of the equalizer is to optimally remove both of them. [Figure 4.15b](#) shows the respective realization sequence of u_n , which is received at the receiver's front end. Observe that, by looking at it, one cannot recognize in it the original sequence; the noise together with the ISI have really changed its "look."

Following a similar procedure as in the previous example, we obtain ([Problem 4.14](#))

$$\Sigma_u = \begin{bmatrix} 1.25\sigma_s^2 + \sigma_\eta^2 & 0.5\sigma_s^2 & 0 \\ 0.5\sigma_s^2 & 1.25\sigma_s^2 + \sigma_\eta^2 & 0.5\sigma_s^2 \\ 0 & 0.5\sigma_s^2 & 1.25\sigma_s^2 + \sigma_\eta^2 \end{bmatrix}, \quad p = \begin{bmatrix} \sigma_s^2 \\ 0.5\sigma_s^2 \\ 0 \end{bmatrix}.$$

**FIGURE 4.15**

(a) A realization of the information sequence comprising equiprobable, randomly generated, ± 1 samples of [Example 4.2](#). (b) The received at the receiver-end corresponding sequence. (c) The sequence at the output of the equalizer for a low channel noise case. The original sequence is fully recovered with no errors. (d) The output of the equalizer for high channel noise. The samples in gray are in error and of opposite polarity compared to the originally transmitted samples.

Solving the normal equations,

$$\Sigma_u \mathbf{w}_* = \mathbf{p},$$

for $\sigma_s^2 = 1$ and $\sigma_\eta^2 = 0.01$, results in

$$\mathbf{w}_* = [0.7462, 0.1195, -0.0474]^T.$$

[Figure 4.15c](#) shows the recovered sequence by the equalizer ($\mathbf{w}_*^T \mathbf{u}_n$). It is exactly the same with the transmitted one; no errors. [Figure 4.15d](#) shows the recovered sequence for the case where the variance of the noise was increased to $\sigma_\eta^2 = 1$. The corresponding MSE optimal equalizer is equal to

$$\mathbf{w}_* = [0.4132, 0.1369, -0.0304]^T.$$

This time, the sequence reconstructed by the equalizer has errors with respect to the transmitted one (gray lines).

A slightly alternative formulation for obtaining Σ_u , instead of computing each one of its elements individually, is the following. Verify that the input vector to the equalizer (with tree taps) at time, n , is given by

$$\mathbf{u}_n = \begin{bmatrix} 0.5 & 1 & 0 & 0 \\ 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0.5 & 1 \end{bmatrix} \begin{bmatrix} s_n \\ s_{n-1} \\ s_{n-2} \\ s_{n-3} \end{bmatrix} + \begin{bmatrix} \eta_n \\ \eta_{n-1} \\ \eta_{n-2} \\ \eta_{n-3} \end{bmatrix}$$

$$:= Hs_n + \eta_n, \quad (4.59)$$

which results in

$$\begin{aligned} \Sigma_u &= \mathbb{E}[\mathbf{u}_n \mathbf{u}_n^T] = H\sigma_s^2 H^T + \Sigma_\eta \\ &= \sigma_s^2 HH^T + \sigma_\eta^2 I. \end{aligned}$$

The reader can easily verify that this is the same as before. Note, however, that (4.59) reminds us of the linear regression model. Moreover, note the special structure of the matrix H . Such matrices are also known as *convolution* matrices. This structure is imposed by the fact that the elements of \mathbf{u}_n are time-shifted versions of the first element, because the input vector corresponds to a random process. This is exactly the property that will be exploited next to derive efficient schemes for the solution of the normal equations.

4.8 ALGORITHMIC ASPECTS: THE LEVINSON AND THE LATTICE-LADDER ALGORITHMS

The goal of this section is to present algorithmic schemes for the efficient solution of the normal equations in (4.16). The filtering case where the input and output entities are random processes will be considered. In this case, we have already pointed out that the input covariance matrix has a special structure. The main concepts to be presented here have a generality that goes beyond the specific form of the normal equations. A vast literature concerning efficient (fast) algorithms for the least-squares task as well as a number of its online/adaptive versions have their roots to the schemes to be presented here. At the heart of all these schemes lies the specific structure of the input vector, whose elements are *time-shifted versions* of its first element, u_n .

Recall from linear algebra that in order to solve a general linear system of l equations with l unknowns, one requires $O(l^3)$ operations (multiplications-additions (MADs)). Exploiting the rich structure of the autocorrelation/covariance matrix, associated with random processes, an algorithm with $O(l^2)$ operations will be derived. The more general complex-valued case will be considered.

The autocorrelation/covariance matrix of the input random vector has been defined in (4.17). That is, it is Hermitian as well as semipositive definite. From now on, we will assume that it is positive definite. The autocorrelation/covariance matrix in $\mathbb{C}^{m \times m}$, associated with a complex wide-sense stationary process, is given by

$$\Sigma_m = \begin{bmatrix} r(0) & r(1) & \cdots & r(m-1) \\ r(-1) & r(0) & \cdots & r(m-2) \\ \vdots & \vdots & \ddots & \vdots \\ r(-m+1) & r(-m+2) & \cdots & r(0) \end{bmatrix}$$

$$= \begin{bmatrix} r(0) & r(1) & \cdots & r(m-1) \\ r^*(1) & r(0) & \cdots & r(m-2) \\ \vdots & \vdots & \ddots & \vdots \\ r^*(m-1) & r^*(m-2) & \cdots & r(0) \end{bmatrix},$$

where the property

$$r(i) := \mathbb{E}[u_n u_{n-i}^*] = \mathbb{E}[(u_{n-i} u_n^*)^*] := r^*(-i)$$

has been used. We have relaxed the notational dependence of Σ on u and we have instead explicitly indicated the order of the matrix, because this will be a very useful index from now on.

We will follow a recursive approach, and our aim will be to express the optimal filter solution of order m , denoted from now on as w_m , in terms of the optimal one, w_{m-1} , of order $m-1$.

The covariance matrix of a wide-sense stationary process is a *Toeplitz* matrix; all the elements along *any* of its diagonals are equal. This property together with its Hermitian nature give rise to the following *nested* structure,

$$\Sigma_m = \begin{bmatrix} \Sigma_{m-1} & J_{m-1} \mathbf{r}_{m-1} \\ \mathbf{r}_{m-1}^H J_{m-1} & r(0) \end{bmatrix} \quad (4.60)$$

$$= \begin{bmatrix} r(0) & \mathbf{r}_{m-1}^T \\ \mathbf{r}_{m-1}^* & \Sigma_{m-1} \end{bmatrix}, \quad (4.61)$$

where

$$\mathbf{r}_{m-1} := \begin{bmatrix} r(1) \\ r(2) \\ \vdots \\ r(m-1) \end{bmatrix}, \quad (4.62)$$

and J_{m-1} is the antidiagonal matrix of dimension $(m-1) \times (m-1)$, defined as

$$J_{m-1} := \begin{bmatrix} 0 & 0 & \cdots & 1 \\ 0 & 0 & \ddots & 0 \\ 0 & 1 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \end{bmatrix}.$$

Note that right-multiplication of any matrix by J_{m-1} has as an effect to reverse the order of its columns, while multiplying it from the left reverses the order of the rows as follows

$$\mathbf{r}_{m-1}^H J_{m-1} = [r^*(m-1) \ r^*(m-2) \ \cdots \ r^*(1)],$$

and

$$J_{m-1} \mathbf{r}_{m-1} = [r(m-1) \ r(m-2) \ \cdots \ r(1)]^T.$$

Applying the matrix inversion lemma from [Appendix A.1](#) for the upper partition in (4.60), we obtain

$$\Sigma_m^{-1} = \begin{bmatrix} \Sigma_{m-1}^{-1} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + \begin{bmatrix} -\Sigma_{m-1}^{-1} J_{m-1} \mathbf{r}_{m-1} \\ 1 \end{bmatrix} \frac{1}{\alpha_{m-1}^b} \begin{bmatrix} -\mathbf{r}_{m-1}^H J_{m-1} \Sigma_{m-1}^{-1} & 1 \end{bmatrix}, \quad (4.63)$$

where for this case the so-called *Schur complement* is the scalar

$$\alpha_{m-1}^b = r(0) - \mathbf{r}_{m-1}^H J_{m-1} \Sigma_{m-1}^{-1} J_{m-1} \mathbf{r}_{m-1}. \quad (4.64)$$

The cross-correlation vector of order m , \mathbf{p}_m , admits the following partition,

$$\mathbf{p}_m = \begin{bmatrix} \mathbb{E}[u_n d_n^*] \\ \vdots \\ \mathbb{E}[u_{n-m+2} d_n^*] \\ \mathbb{E}[u_{n-m+1} d_n^*] \end{bmatrix} = \begin{bmatrix} \mathbf{p}_{m-1} \\ p_{m-1} \end{bmatrix}, \quad \text{where } p_{m-1} := \mathbb{E}[u_{n-m+1} d_n^*]. \quad (4.65)$$

Combining (4.63) and (4.65), the following elegant relation results:

$$\mathbf{w}_m := \Sigma_m^{-1} \mathbf{p}_m = \begin{bmatrix} \mathbf{w}_{m-1} \\ 0 \end{bmatrix} + \begin{bmatrix} -\mathbf{b}_{m-1} \\ 1 \end{bmatrix} k_{m-1}^w, \quad (4.66)$$

where

$$\mathbf{w}_{m-1} = \Sigma_{m-1}^{-1} \mathbf{p}_{m-1}, \quad \mathbf{b}_{m-1} := \Sigma_{m-1}^{-1} J_{m-1} \mathbf{r}_{m-1},$$

and

$$k_{m-1}^w := \frac{\mathbf{p}_{m-1} - \mathbf{r}_{m-1}^H J_{m-1} \mathbf{w}_{m-1}}{\alpha_{m-1}^b}. \quad (4.67)$$

Equation (4.66) is an order recursion that relates the optimal solution \mathbf{w}_m with \mathbf{w}_{m-1} . In order to obtain a complete recursive scheme, all one needs is a recursion for updating \mathbf{b}_m .

Forward and backward MSE optimal predictors

Backward Prediction: The vector $\mathbf{b}_m = \Sigma_m^{-1} J_m \mathbf{r}_m$ has an interesting physical interpretation: it is the MSE-optimal backward predictor of order m . That is, it is the linear filter, which optimally estimates/predicts the value of u_{n-m} given the values of $u_{n-m+1}, u_{n-m+2}, \dots, u_n$. Thus, in order to design the optimal backward predictor of order m , the desired response must be

$$d_n = u_{n-m},$$

and from the respective normal equations we get

$$\mathbf{b}_m = \Sigma_m^{-1} \begin{bmatrix} \mathbb{E}[u_n u_{n-m}^*] \\ \mathbb{E}[u_{n-1} u_{n-m}^*] \\ \vdots \\ \mathbb{E}[u_{n-m+1} u_{n-m}^*] \end{bmatrix} = \Sigma_m^{-1} J_m \mathbf{r}_m. \quad (4.68)$$

Hence, the MSE-optimal backward predictor coincides with \mathbf{b}_m , i.e.,

$$\boxed{\mathbf{b}_m = \Sigma_m^{-1} J_m \mathbf{r}_m : \text{MSE-Optimal Backward Predictor.}}$$

Moreover, the corresponding minimum mean-square error, adapting (4.19) to our current needs, is equal to

$$J(\mathbf{b}_m) = r(0) - \mathbf{r}_m^H J_m \Sigma_m^{-1} J_m \mathbf{r}_m = \alpha_m^b.$$

That is, the Schur complement in (4.64) is equal to the respective optimal mean-square error!

Forward Prediction: The goal of the forward prediction task is to predict the value u_{n+1} , given the values $u_n, u_{n-1}, \dots, u_{n-m+1}$. Thus, the MSE-optimal forward predictor of order m , \mathbf{a}_m , is obtained by selecting the desired response $d_n = u_{n+1}$, and the respective normal equations become

$$\mathbf{a}_m = \Sigma_m^{-1} \begin{bmatrix} \mathbb{E}[u_n u_{n+1}^*] \\ \mathbb{E}[u_{n-1} u_{n+1}^*] \\ \vdots \\ \mathbb{E}[u_{n-m+1} u_{n+1}^*] \end{bmatrix} = \Sigma_m^{-1} \begin{bmatrix} r^*(1) \\ r^*(2) \\ \vdots \\ r^*(m) \end{bmatrix} \quad (4.69)$$

or

$$\boxed{\mathbf{a}_m = \Sigma_m^{-1} \mathbf{r}_m^* : \text{MSE-Optimal Forward Predictor.}} \quad (4.70)$$

From (4.70), it is not difficult to show (Problem 4.16) that (recall that $J_m J_m = I_m$)

$$\mathbf{a}_m = J_m \mathbf{b}_m^* \Rightarrow \mathbf{b}_m = J_m \mathbf{a}_m^*, \quad (4.71)$$

and that the optimal mean-square error for the forward prediction, $J(\mathbf{a}_m) := \alpha_m^f$, is equal to that for the backward, i.e.,

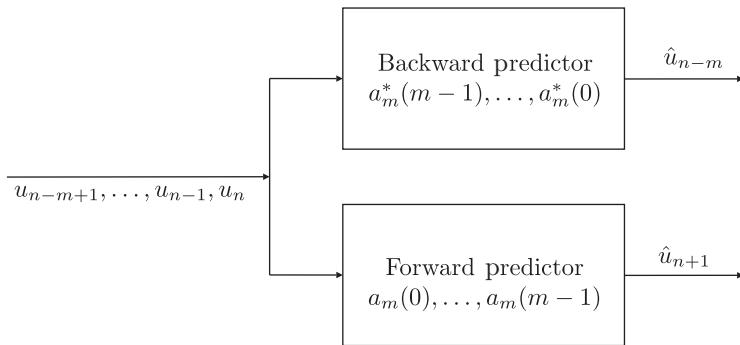
$$J(\mathbf{a}_m) = \alpha_m^f = \alpha_m^b = J(\mathbf{b}_m).$$

Figure 4.16 depicts the two prediction tasks. In other words, the optimal forward predictor is the conjugate reverse of the backward one, so that

$$\mathbf{a}_m := \begin{bmatrix} a_m(0) \\ \vdots \\ a_m(m-1) \end{bmatrix} = J_m \mathbf{b}_m^* := \begin{bmatrix} b_m^*(m-1) \\ \vdots \\ b_m^*(0) \end{bmatrix}.$$

This property is due to the stationarity of the involved process. Because the statistical properties only depend on the difference of the time instants, forward and backward predictions are not much different; in both cases, given a set of samples, u_{n-m+1}, \dots, u_n , we predict *one* sample ahead in the future (u_{n+1} in the forward prediction) or *one* sample back in the past (u_{n-m} in the backward prediction).

Having established the relationship between \mathbf{a}_m and \mathbf{b}_m in (4.71), we are ready to complete the missing step in (4.66); that is, to complete an order recursive step for the update of \mathbf{b}_m . Since (4.66)

**FIGURE 4.16**

The impulse response of the backward predictor is the conjugate reverse of that of the forward predictor.

holds true for any desired response, d_n , it also applies for the special case where the optimal filter to be designed is the forward predictor \mathbf{a}_m ; in this case, $d_n = u_{n+1}$. Replacing in (4.66) w_m (w_{m-1}) with \mathbf{a}_m (\mathbf{a}_{m-1}) results in

$$\mathbf{a}_m = \begin{bmatrix} \mathbf{a}_{m-1} \\ 0 \end{bmatrix} + \begin{bmatrix} -J_{m-1} \mathbf{a}_{m-1}^* \\ 1 \end{bmatrix} k_{m-1}, \quad (4.72)$$

where (4.71) has been used and

$$k_{m-1} = \frac{r^*(m) - \mathbf{r}_{m-1}^H J_{m-1} \mathbf{a}_{m-1}}{\alpha_{m-1}^b}. \quad (4.73)$$

Combining (4.66), (4.67), (4.71), (4.72), and (4.73) the following algorithm, known as *Levinson's* algorithm, for the solution of the normal equations results:

Algorithm 4.1 (Levinson's algorithm).

- Input
 - $r(0), r(1), \dots, r(l)$
 - $p_k = \mathbb{E}[u_{n-k} d_n^*], k = 0, 1, \dots, l-1$
- Initialize
 - $w_1 = \frac{p_0}{r(0)}, a_1 = \frac{r^*(1)}{r(0)}, \alpha_1^b = r(0) - \frac{|r(1)|^2}{r(0)}$
 - $k_1^w = \frac{p_1 - r^*(1)w_1}{\alpha_1^b}, k_1 = \frac{r^*(2) - r^*(1)a_1}{\alpha_1^b}$
- **For** $m = 2, \dots, l-1$, **Do**
 - $w_m = \begin{bmatrix} w_{m-1} \\ 0 \end{bmatrix} + \begin{bmatrix} -J_{m-1} \mathbf{a}_{m-1}^* \\ 1 \end{bmatrix} k_{m-1}^w$
 - $\mathbf{a}_m = \begin{bmatrix} \mathbf{a}_{m-1} \\ 0 \end{bmatrix} + \begin{bmatrix} -J_{m-1} \mathbf{a}_{m-1}^* \\ 1 \end{bmatrix} k_{m-1}$
 - $\alpha_m^b = \alpha_{m-1}^b (1 - |k_{m-1}|^2)$

- $k_m^w = \frac{p_m - \mathbf{r}_m^H \mathbf{J}_m \mathbf{w}_m}{\alpha_m^b}$
- $k_m = \frac{\mathbf{r}^*(m+1) - \mathbf{r}_m^H \mathbf{J}_m \mathbf{a}_m}{\alpha_m^b}$

- **End For**

Note that the update for α_m^b is a direct consequence of its definition in (4.64) and (4.72) (Problem 4.17). Also note that $\alpha_m^b \geq 0$ implies that $|k_m| \leq 1$.

Remarks 4.3.

- The complexity per order recursion is $4m$ MADS, hence for a system with l equations this amounts to $2l^2$ MADS. This computational saving is substantial compared to the $O(l^3)$ MADS, required by adopting a general purpose scheme. The previous very elegant scheme was proposed in 1947 by Levinson, [26]. A formulation of the algorithm was also independently proposed by Durbin, [12] and the algorithm is usually called the *Levinson-Durbin* algorithm. In [11], it was shown that Levinson's algorithm is redundant in its prediction part and the *split Levinson* algorithm was developed, whose recursions evolve around symmetric vector quantities leading to further computational savings.

4.8.1 THE LATTICE-LADDER SCHEME

So far, we have been involved with the so called *transversal* implementation of an LTI FIR filter; in other words, the output is expressed as a convolution between the impulse response and the input of the linear structure. Levinson's algorithm provided a computationally efficient scheme for obtaining the MSE-optimal estimate \mathbf{w}_* . We now turn our attention to an equivalent implementation of the corresponding linear filter, which comes as a direct consequence of Levinson's algorithm.

Define the error signals associated with the m th order optimal forward and backward predictors, at time instant n , as

$$\mathbf{e}_m^f(n) := \mathbf{u}_n - \mathbf{a}_m^H \mathbf{u}_m(n-1), \quad (4.74)$$

where $\mathbf{u}_m(n)$ is the input random vector of the m th order filter, and the order of the filter has been explicitly brought into the notation.⁷ The backward error is given by

$$\begin{aligned} \mathbf{e}_m^b(n) &:= \mathbf{u}_{n-m} - \mathbf{b}_m^H \mathbf{u}_m(n) \\ &= \mathbf{u}_{n-m} - \mathbf{a}_m^T \mathbf{J}_m \mathbf{u}_m(n). \end{aligned} \quad (4.75)$$

Employing in (4.74), (4.75), the order recursion in (4.72) and the partitioning of $\mathbf{u}_m(n)$, which is represented by

$$\mathbf{u}_m(n) = [\mathbf{u}_{m-1}^T(n), \mathbf{u}_{n-m+1}]^T = [\mathbf{u}_n, \mathbf{u}_{m-1}^T(n-1)]^T, \quad (4.76)$$

we readily obtain

$$\mathbf{e}_m^f(n) = \mathbf{e}_{m-1}^f(n) - \mathbf{e}_{m-1}^b(n-1) k_{m-1}^*, \quad m = 1, 2, \dots, l, \quad (4.77)$$

$$\mathbf{e}_m^b(n) = \mathbf{e}_{m-1}^b(n-1) - \mathbf{e}_{m-1}^f(n) k_{m-1}, \quad m = 1, 2, \dots, l, \quad (4.78)$$

⁷ The time index is now given in parentheses, to avoid having double subscripts.

with $e_0^f(n) = e_0^b(n) = u_n$, and $k_0 = \frac{r^*(1)}{r(0)}$. This pair of recursions is known as *lattice* recursions. Let us focus a bit more on this set of equations.

Orthogonality of the optimal backward errors

From the vector space interpretation of random signals, it is apparent that $e_m^b(n)$ lies in the subspace spanned by u_{n-m}, \dots, u_n , and we can write

$$e_m^b(n) \in \text{span}\{u(n-m), \dots, u(n)\}.$$

Moreover, because $e_m^b(n)$ is the error associated with the MSE-optimal backward predictor, $e_m^b(n) \perp \text{span}\{u(n-m+1), \dots, u(n)\}$. However, the latter subspace is the one where $e_{m-k}^b(n)$, $k = 1, 2, \dots, m$, lie. Hence, for $m = 1, 2, \dots, l-1$, we can write,

$e_m^b(n) \perp e_k^b(n), k < m : \quad \text{Orthogonality of the Backward Errors.}$

Moreover, it is obvious that

$$\text{span}\{e_0^b(n), e_1^b(n), \dots, e_{l-1}^b(n)\} = \text{span}\{u_n, u_{n-1}, \dots, u_{n-l+1}\}.$$

Hence, the normalized vectors

$\tilde{e}_m^b(n) := \frac{e_m^b(n)}{\|e_m^b(n)\|}, \quad m = 0, 1, \dots, l-1 : \quad \text{Orthonormal Basis,}$

form an *orthonormal* basis in $\text{span}\{u_n, u_{n-1}, \dots, u_{n-l+1}\}$, see Figure 4.17. As a matter of fact, the pair in (4.77), (4.78) comprise a *Gram-Schmidt* orthogonalizer [47].

Let us now express \hat{d}_n , or the projection of d_n in $\text{span}\{u_n, \dots, u_{n-l+1}\}$, in terms of the new set of orthogonal vectors,

$$\hat{d}_n = \sum_{m=0}^{l-1} h_m \tilde{e}_m^b(n), \quad (4.79)$$

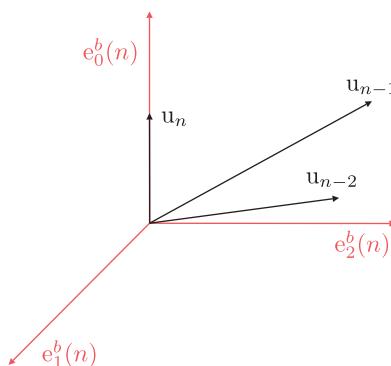


FIGURE 4.17

The optimal backward errors form an orthogonal basis in the respective input random signal space.

where the coefficients h_m are given by

$$\begin{aligned} h_m &= \langle \hat{\mathbf{d}}_n, \frac{\mathbf{e}_m^b(n)}{\|\mathbf{e}_m^b(n)\|^2} \rangle = \frac{\mathbb{E}[\hat{\mathbf{d}}_n \mathbf{e}_m^{b*}(n)]}{\|\mathbf{e}_m^b(n)\|^2} = \frac{\mathbb{E}[(\mathbf{d}_n - \mathbf{e}_n) \mathbf{e}_m^{b*}(n)]}{\|\mathbf{e}_m^b(n)\|^2} \\ &= \frac{\mathbb{E}[\mathbf{d}_n \mathbf{e}_m^{b*}(n)]}{\|\mathbf{e}_m^b(n)\|^2}, \end{aligned} \quad (4.80)$$

where the orthogonality of the error, \mathbf{e}_n , with the subspace spanned by the backward errors has been taken into account. From (4.67) and (4.80), and taking into account the respective definitions of the involved quantities, we readily obtain that

$$h_m = k_m^w.$$

That is, the coefficients k_m^w , $m = 0, 1, \dots, l-1$, in Levinson's algorithm are the parameters in the expansion of $\hat{\mathbf{d}}_n$ in terms of the orthogonal basis. Combining (4.77), (4.78) and (4.79) the *lattice-ladder* scheme of Figure 4.18 results, whose output is the MSE approximation $\hat{\mathbf{d}}_n$ of \mathbf{d}_n .

Remarks 4.4.

- The lattice-ladder scheme is a highly efficient, *modular* structure. It comprises a sequence of successive similar stages. To increase the order of the filter, it suffices to add an extra stage, which is a highly desirable property in VLSI implementations. Moreover, lattice-ladder schemes enjoy a higher robustness, compared to Levinson's algorithm, with respect to numerical inaccuracies.
- *Cholesky Factorization.* The orthogonality property of the optimal MSE backward errors leads to another interpretation of the involved parameters. From the definition in (4.75), we get

$$\mathbf{e}_l^b(n) := \begin{bmatrix} \mathbf{e}_0^b(n) \\ \mathbf{e}_1^b(n) \\ \vdots \\ \mathbf{e}_{l-1}^b(n) \end{bmatrix} = U^H \begin{bmatrix} \mathbf{u}_n \\ \mathbf{u}_{n-1} \\ \vdots \\ \mathbf{u}_{n-l+1} \end{bmatrix} = U^H \mathbf{u}_l(n), \quad (4.81)$$

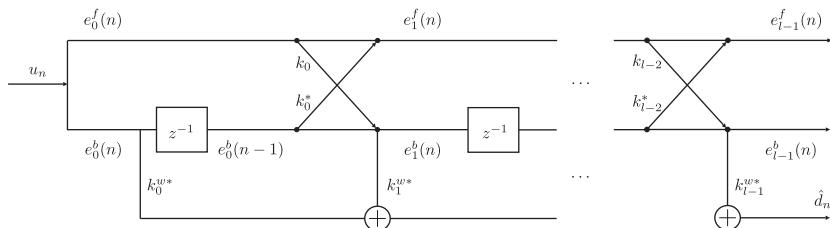


FIGURE 4.18

The lattice-ladder structure. In contrast to the transversal implementation in terms of w_m , the parameterization is now in terms of k_m , k_m^w , $m = 0, 1, \dots, l-1$, $k_0 = \frac{r^*(1)}{r(0)}$, and $k_0^w = \frac{p_0^*}{r(0)}$. Note the resulting highly modular structure.

where

$$U^H := \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -a_1(0) & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ -a_l(l-1) & -a_l(l-2) & \cdots & \cdots & 1 \end{bmatrix}$$

and

$$\mathbf{a}_m := [a_m(0), a_m(1), \dots, a_m(m-1)]^T, m = 1, 2, \dots, l.$$

Due to the orthogonality of the involved backward errors,

$$\mathbb{E}[\mathbf{e}_l^b(n)\mathbf{e}_l^{bH}(n)] = U^H \Sigma_l U = D$$

where

$$D := \text{diag} \left\{ \alpha_0^b, \alpha_1^b, \dots, \alpha_{l-1}^b \right\},$$

or

$$\Sigma_l^{-1} = UD^{-1}U^H = (UD^{-1/2})(UD^{-1/2})^H.$$

That is, the prediction error powers and the weights of the optimal forward predictor provide the *Cholesky factorization* of the inverse covariance matrix.

- *The Schur Algorithm.* In a parallel processing environment, the inner products involved in Levinson's algorithm pose a bottleneck in the flow of the algorithm. Note that the updates for \mathbf{w}_m and \mathbf{a}_m can be performed fully in parallel. Schur's algorithm [45] is an alternative scheme that overcomes the bottleneck, and in a multiprocessor environment the complexity can go down to $O(l)$. The parameters involved in Schur's algorithm perform a Cholesky factorization of Σ_l (e.g., [21, 22]).
- Note that all these algorithmic schemes for the efficient solution of the normal equations owe their existence to the rich structure that the (autocorrelation) covariance matrix as well as the cross-correlation vector acquire when the involved jointly distributed random entities are random processes; their *time sequential nature* imposes such a structure. The derivation of the Levinson and lattice-ladder schemes reveal the flavor of the type of techniques that can be (and have extensively been) used to derive computational schemes for the online/adaptive versions and the related least-squares error loss function, to be discussed in Chapter 6. There, the algorithms may be computationally more involved, but the essence behind them is the same as for those used in the current section.

4.9 MEAN-SQUARE ERROR ESTIMATION OF LINEAR MODELS

We now turn our attention to the case where the underlying model that relates the input-output variables is a linear one. Not to be confused with what was treated in the previous sections, it must be stressed that, so far, we have been concerned with the linear estimator task. At no point in this stage of our discussion has the generation model of the data been brought in (with the exception in the comment of Remarks 4.2). We just adopted a linear estimator and obtained the MSE solution for it. The focus was

on the solution and its properties. The emphasis here is on cases where the input-output variables are related via a linear data generation model.

Let us assume that we are given two jointly distributed random vectors, \mathbf{y} and $\boldsymbol{\theta}$, which are related according to the following linear model,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta}, \quad (4.82)$$

where $\boldsymbol{\eta}$ denotes the set of the involved noise variables. Note that such a model covers the case of our familiar regression task, where the unknown parameters $\boldsymbol{\theta}$ are considered random, which is in line with the Bayesian philosophy, as discussed in [Chapter 3](#). Once more, we assume zero-mean vectors; otherwise, the respective mean values are subtracted. The dimensions of \mathbf{y} ($\boldsymbol{\eta}$) and $\boldsymbol{\theta}$ may not necessarily be the same; to be in line with the notation used in [Chapter 3](#), let $\mathbf{y}, \boldsymbol{\eta} \in \mathbb{R}^N$ and $\boldsymbol{\theta} \in \mathbb{R}^l$. Hence, \mathbf{X} is a $N \times l$ matrix. Note that the matrix \mathbf{X} is considered to be deterministic and not a random one.

Assume the covariance matrices of our zero-mean variables,

$$\Sigma_{\boldsymbol{\theta}} = \mathbb{E}[\boldsymbol{\theta}\boldsymbol{\theta}^T], \quad \Sigma_{\boldsymbol{\eta}} = \mathbb{E}[\boldsymbol{\eta}\boldsymbol{\eta}^T],$$

are known. The goal is to compute a matrix, H , of dimension $l \times N$, so that the linear estimator

$$\hat{\boldsymbol{\theta}} = H\mathbf{y} \quad (4.83)$$

minimizes the mean-square error cost

$$J(H) := \mathbb{E}[(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})] = \sum_{i=1}^l \mathbb{E}[|\theta_i - \hat{\theta}_i|^2]. \quad (4.84)$$

Note that this is a *multichannel* estimation task and it is equivalent with solving l optimization tasks, one for each component, θ_i , of $\boldsymbol{\theta}$. Defining the error vector as

$$\boldsymbol{\varepsilon} := \boldsymbol{\theta} - \hat{\boldsymbol{\theta}},$$

the cost function is equal to the trace of the corresponding *error covariance matrix*, so that

$$J(H) := \text{trace}\{\mathbb{E}[\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T]\}.$$

Focusing on the i -th component in (4.83), we write

$$\hat{\theta}_i = \mathbf{h}_i^T \mathbf{y}, \quad i = 1, 2, \dots, l, \quad (4.85)$$

where \mathbf{h}_i^T is the i -th row of H and its optimal estimate is given by

$$\mathbf{h}_{*,i} := \arg \min_{\mathbf{h}_i} \mathbb{E}[|\theta_i - \hat{\theta}_i|^2] = \mathbb{E}[|\theta_i - \mathbf{h}_i^T \mathbf{y}|^2]. \quad (4.86)$$

Minimizing (4.86) is exactly the same task as that of the linear estimation considered in previous section (with \mathbf{y} in place of \mathbf{x} and θ_i in place of y), hence,

$$\Sigma_y \mathbf{h}_{*,i} = \mathbf{p}_i, \quad i = 1, 2, \dots, l,$$

where

$$\Sigma_y = \mathbb{E}[\mathbf{y}\mathbf{y}^T] \quad \text{and} \quad \mathbf{p}_i = \mathbb{E}[\mathbf{y}\theta_i], \quad i = 1, 2, \dots, l,$$

or

$$\mathbf{h}_{*,i}^T = \mathbf{p}_i^T \Sigma_y^{-1}, \quad i = 1, 2, \dots, l,$$

and finally,

$$H_* = \Sigma_{y\theta} \Sigma_y^{-1}, \quad \hat{\theta} = \Sigma_{y\theta} \Sigma_y^{-1} \mathbf{y}, \quad (4.87)$$

where

$$\Sigma_{y\theta} := \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_l^T \end{bmatrix} = \mathbb{E}[\theta \mathbf{y}^T] \quad (4.88)$$

is an $l \times N$ cross-correlation matrix. All that is now required is to compute Σ_y and $\Sigma_{y\theta}$. To this end,

$$\begin{aligned} \Sigma_y &= \mathbb{E}[\mathbf{y}\mathbf{y}^T] = \mathbb{E}[(X\theta + \eta)(\theta^T X^T + \eta^T)] \\ &= X\Sigma_\theta X^T + \Sigma_\eta, \end{aligned} \quad (4.89)$$

where the independence of the zero mean vectors θ and η has been used. Similarly,

$$\Sigma_{y\theta} = \mathbb{E}[\theta \mathbf{y}^T] = \mathbb{E}[\theta(\theta^T X^T + \eta^T)] = \Sigma_\theta X^T, \quad (4.90)$$

and combining (4.87), (4.89), and (4.90), we obtain

$$\hat{\theta} = \Sigma_\theta X^T (\Sigma_\eta + X\Sigma_\theta X^T)^{-1} \mathbf{y}. \quad (4.91)$$

Employing from [Appendix A.1](#) the matrix identity

$$(A^{-1} + B^T C^{-1} B)^{-1} B^T C^{-1} = A B^T (B A B^T + C)^{-1}$$

in (4.91) we obtain

$$\hat{\theta} = (\Sigma_\theta^{-1} + X^T \Sigma_\eta^{-1} X)^{-1} X^T \Sigma_\eta^{-1} \mathbf{y} : \quad \text{MSE Linear Estimator.}$$

(4.92)

In case of complex-valued variables, the only difference is that transposition is replaced by Hermitian transposition.

Remarks 4.5.

- Recall from [Chapter 3](#) that the optimal MSE estimator of θ given the values of \mathbf{y} is provided by

$$\mathbb{E}[\theta|\mathbf{y}].$$

However, as it was shown in [Problem 3.14](#), if θ and \mathbf{y} are jointly Gaussian vectors, then the optimal estimator is linear (affine for nonzero mean variables) and it coincides with the MSE linear estimator of (4.92).

- If we allow nonzero mean values, then instead of (4.83) the affine model should be adopted,

$$\hat{\theta} = H\mathbf{y} + \mu.$$

Then

$$\mathbb{E}[\hat{\theta}] = H\mathbb{E}[\mathbf{y}] + \mu \Rightarrow \mu = \mathbb{E}[\hat{\theta}] - H\mathbb{E}[\mathbf{y}].$$

Hence,

$$\hat{\theta} = \mathbb{E}[\hat{\theta}] + H(\mathbf{y} - \mathbb{E}[\mathbf{y}]),$$

and finally,

$$\hat{\theta} - \mathbb{E}[\hat{\theta}] = H(\mathbf{y} - \mathbb{E}[\mathbf{y}]),$$

which justifies our approach to subtract the means and work with zero-mean value variables. For nonzero mean values, the analogue of (4.92) is

$$\hat{\theta} = \mathbb{E}[\hat{\theta}] + \left(\Sigma_{\theta}^{-1} + X^T \Sigma_{\eta}^{-1} X \right)^{-1} X^T \Sigma_{\eta}^{-1} (\mathbf{y} - \mathbb{E}[\mathbf{y}]). \quad (4.93)$$

Note that for zero-mean noise η , $\mathbb{E}[\mathbf{y}] = X\mathbb{E}[\theta]$.

- Compare (4.93) with (3.71) for the Bayesian inference approach. They are identical, provided that the covariance matrix of the prior (Gaussian) pdf is equal to Σ_{θ} and $\theta_0 = \mathbb{E}[\hat{\theta}]$ for a zero-mean noise variable.

4.9.1 THE GAUSS-MARKOV THEOREM

We now turn our attention to the case where θ in the regression model is considered to be an (unknown) constant, instead of a random vector. Thus, the linear model is now written as

$$\mathbf{y} = X\theta + \eta, \quad (4.94)$$

and the randomness of \mathbf{y} is solely due to η , which is assumed to be zero-mean with covariance matrix Σ_{η} . The goal is to design an *unbiased linear* estimator of θ , that minimizes the mean-square error,

$$\hat{\theta} = Hy, \quad (4.95)$$

and select H such as

$$\begin{aligned} & \text{minimize} && \text{trace} \left\{ \mathbb{E}[(\theta - \hat{\theta})(\theta - \hat{\theta})^T] \right\} \\ & \text{subject to} && \mathbb{E}[\hat{\theta}] = \theta. \end{aligned} \quad (4.96)$$

From (4.94) and (4.95), we get that

$$\mathbb{E}[\hat{\theta}] = H\mathbb{E}[\mathbf{y}] = H\mathbb{E}[(X\theta + \eta)] = HX\theta,$$

which implies that the unbiased constraint is equivalent to

$$HX = I. \quad (4.97)$$

Employing (4.95), the error vector becomes

$$\epsilon = \theta - \hat{\theta} = \theta - Hy = \theta - H(X\theta + \eta) = H\eta. \quad (4.98)$$

Hence, the constrained minimization in (4.96) can now be written as

$$\begin{aligned} H_* &= \arg \min_H \text{trace}\{H\Sigma_{\eta}H^T\}, \\ \text{s.t. } & HX = I. \end{aligned} \quad (4.99)$$

Solving (4.99) results in (Problem 4.18)

$$H_* = (X^T \Sigma_\eta^{-1} X)^{-1} X^T \Sigma_\eta^{-1}, \quad (4.100)$$

and the associated minimum mean-square error is

$$J(H_*) := \text{MSE}(H_*) = \text{trace} \left\{ (X^T \Sigma_\eta^{-1} X)^{-1} \right\}. \quad (4.101)$$

The reader can verify that

$$J(H) \geq J(H_*),$$

for any other linear unbiased estimator (Problem 4.19).

The previous result is known as the *Gauss-Markov* theorem. The optimal MSE linear unbiased estimator is given by

$$\hat{\theta} = (X^T \Sigma_\eta^{-1} X)^{-1} X^T \Sigma_\eta^{-1} \mathbf{y} : \text{BLUE} \quad (4.102)$$

and it is also known as the *best linear unbiased estimator* (BLUE), or the *minimum variance unbiased linear estimator*. For complex-valued variables, the transposition is simply replaced by the Hermitian one.

Remarks 4.6.

- For the BLUE to exist, $X^T \Sigma_\eta^{-1} X$ must be invertible. This is guaranteed if Σ_η is positive definite and the $N \times l$ matrix X , $N \geq l$, is full rank (Problem 4.20).
- Observe that the BLUE coincides with the maximum likelihood estimator (Chapter 3), if η follows a multivariate Gaussian distribution; recall that under this assumption, the Cramér-Rao bound is achieved. If this is not the case, there may be another unbiased estimator (nonlinear), which results in lower MSE. Recall also from Chapter 3, that there may be a biased estimator that results in lower MSE; see [13, 38] and the references therein for a related discussion.

Example 4.3 (Channel Identification). The task is illustrated in Figure 4.11. Assume that we have access to a set of input-output observations, u_n and d_n , $n = 0, 1, 2, \dots, N - 1$. Moreover, we are given that the impulse response of the system comprises l taps and it is zero-mean and its covariance matrix is Σ_w . Also, the second-order statistics of the zero-mean noise are also known and we are given its covariance matrix, Σ_η . Then, assuming that the plant starts from zero initial conditions, we can adopt the following model relating the involved random variables (in line with the model in (4.82)),

$$\mathbf{d} := \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{l-1} \\ \vdots \\ d_{N-1} \end{bmatrix} = U \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{l-1} \end{bmatrix} + \begin{bmatrix} \eta_0 \\ \eta_1 \\ \vdots \\ \eta_{l-1} \\ \vdots \\ \eta_{N-1} \end{bmatrix}, \quad (4.103)$$

where

$$U := \begin{bmatrix} u_0 & 0 & 0 & \cdots & 0 \\ u_1 & u_0 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ u_{l-1} & u_{l-2} & \cdots & \cdots & u_0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ u_{N-1} & \cdots & \cdots & \cdots & u_{N-l} \end{bmatrix}.$$

Note that U is treated deterministically. Then, recalling (4.92) and plugging in the set of obtained measurements, the following estimate results:

$$\hat{\mathbf{w}} = (\Sigma_w^{-1} + U^T \Sigma_\eta^{-1} U)^{-1} U^T \Sigma_\eta^{-1} \mathbf{d}. \quad (4.104)$$

4.9.2 CONSTRAINED LINEAR ESTIMATION: THE BEAMFORMING CASE

We have already dealt with a constrained linear estimation task in Section 4.9.1 in our effort to obtain an unbiased estimator of a fixed-value parameter vector. In the current section, we will see that the procedure developed there is readily applicable for cases where the unknown parameter vector is required to respect certain linear constraints.

We will demonstrate such a constrained task in the context of *beamforming*. Figure 4.19 illustrates the basic block diagram of the beamforming task. A beamformer comprises a set of antenna elements. We consider the case where the antenna elements are uniformly spaced along a straight line. The goal is to linearly combine the signals received by the individual antenna elements, so as to

- turn the main beam of the array to a specific direction in space, and
- optimally reduce the noise.

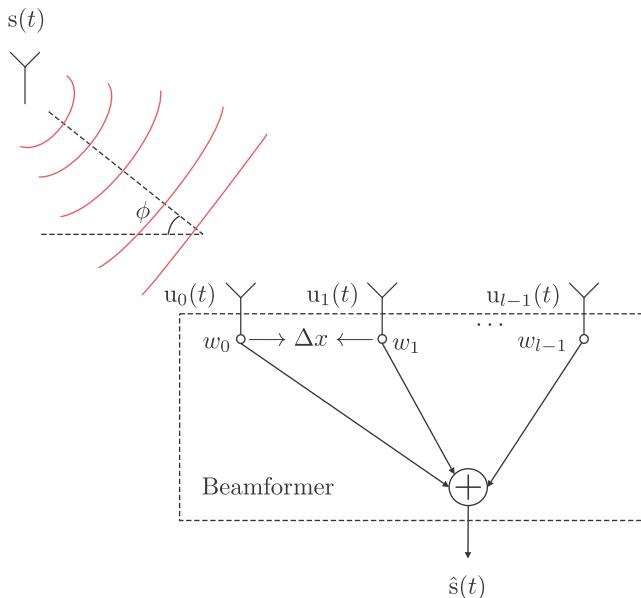
The first goal imposes a constraint to the designer, which will guarantee that the gain of the array is high for the specific desired direction; for the second goal, we will adopt MSE arguments.

In a more formal way, assume that the transmitter is far enough away, so as to guarantee that the wavefronts that the array “sees” are planar. Let $s(t)$ be the information random process transmitted at a carrier frequency, ω_c ; hence, the modulated signal is

$$\mathbf{r}(t) = s(t)e^{j\omega_ct}.$$

If Δx is the distance between successive elements of the array, then a wavefront that arrives at time t_0 at the first element will reach the i -th element delayed by

$$\Delta t_i = t_i - t_0 = i \frac{\Delta x \cos \phi}{c}, \quad i = 0, 1, \dots, l-1,$$

**FIGURE 4.19**

The task of the beamformer is to obtain estimates of the weights w_0, \dots, w_{l-1} , so as to minimize the effect of noise and at the same time to impose a constraint that, in the absence of noise, would leave signals impinging the array from the desired angle, ϕ , unaffected.

where c is the speed of propagation, ϕ is the angle formed by the array and the direction propagation of the wavefronts, and l the number of array elements. We know from our basic electromagnetic courses that

$$c = \frac{\omega_c \lambda}{2\pi},$$

where λ is the respective wavelength. Taking a snapshot at time t , the signal received from direction ϕ at the i -th element will be

$$\begin{aligned} r_i(t) &= s(t - \Delta t_i) e^{j\omega_c(t-i)\frac{2\pi\Delta x \cos \phi}{\omega_c \lambda}} \\ &\simeq s(t) e^{j\omega_c t} e^{-2\pi j \frac{i\Delta x \cos \phi}{\lambda}}, \quad i = 0, 1, \dots, l-1, \end{aligned}$$

where we have assumed a relatively low time signal variation. After converting the received signals in the baseband (multiplying by $e^{-j\omega_c t}$), the vector of the received signals (one per array element), at time t , can be written in the following linear regression-type formulation,

$$\mathbf{u}(t) := \begin{bmatrix} \mathbf{u}_0(t) \\ \mathbf{u}_1(t) \\ \vdots \\ \mathbf{u}_{l-1}(t) \end{bmatrix} = \mathbf{x}s(t) + \boldsymbol{\eta}(t), \quad (4.105)$$

where

$$\mathbf{x} := \begin{bmatrix} 1 \\ e^{-2\pi j} \frac{\Delta x \cos \phi}{\lambda} \\ \vdots \\ e^{-2\pi j} \frac{(l-1)\Delta x \cos \phi}{\lambda} \end{bmatrix},$$

and the vector $\boldsymbol{\eta}(t)$ contains the additive noise plus any other interference due to signals coming from directions other than ϕ , so that

$$\boldsymbol{\eta}(t) = [\eta_0(t), \dots, \eta_{l-1}(t)]^T,$$

and it is assumed to be of zero mean; \mathbf{x} is also known as the *steering vector*. The output of the beamformer, acting on the input vector signal, will be

$$\hat{s}(t) = \mathbf{w}^H \mathbf{u}(t),$$

where the Hermitian transposition has to be used, because now the involved signals are complex-valued.

We will first impose the constraint. Ideally, in the absence of noise, one would like to recover signals impinging on the array from the desired direction, ϕ , exactly. Thus, \mathbf{w} should satisfy the following constraint

$$\mathbf{w}^H \mathbf{x} = 1, \quad (4.106)$$

which guarantees that $\hat{s}(t) = s(t)$ in the absence of noise. Note that (4.106) is an instance of (4.97) if we consider \mathbf{w}^H and \mathbf{x} in place of H and X , respectively. To account for the noise, we require the MSE

$$\mathbb{E} [|s(t) - \hat{s}(t)|^2] = \mathbb{E} [|s(t) - \mathbf{w}^H \mathbf{u}(t)|^2],$$

to be minimized. However,

$$s(t) - \mathbf{w}^H \mathbf{u}(t) = s(t) - \mathbf{w}^H (\mathbf{x}s(t) + \boldsymbol{\eta}(t)) = -\mathbf{w}^H \boldsymbol{\eta}(t).$$

Hence, the optimal \mathbf{w}_* results from the following constrained task

$$\begin{aligned} \mathbf{w}_* &:= \arg \min_{\mathbf{w}} (\mathbf{w}^H \Sigma_{\eta} \mathbf{w}) \\ \text{s.t.} \quad \mathbf{w}^H \mathbf{x} &= 1, \end{aligned} \quad (4.107)$$

which is an instance of (4.99) and the solution is given by (4.100); adapting it to the current notation and to its complex-valued formulation, we get

$$\boxed{\mathbf{w}_*^H = \frac{\mathbf{x}^H \Sigma_{\eta}^{-1}}{\mathbf{x}^H \Sigma_{\eta}^{-1} \mathbf{x}}}, \quad (4.108)$$

and

$$\hat{s}(t) = \mathbf{w}_*^H \mathbf{u}(t) = \frac{\mathbf{x}^H \Sigma_{\eta}^{-1} \mathbf{u}(t)}{\mathbf{x}^H \Sigma_{\eta}^{-1} \mathbf{x}}. \quad (4.109)$$

The minimum MSE is equal to

$$\text{MSE}(\mathbf{w}_*) = \frac{1}{\mathbf{x}^H \Sigma_{\eta}^{-1} \mathbf{x}}. \quad (4.110)$$

An alternative formulation for the cost function in order to estimate the weights of the beamformer, which is often met in practice, builds upon the goal to minimize the output power, subject to the same constraint as before,

$$\begin{aligned} \mathbf{w}_* &:= \arg \min_{\mathbf{w}} \mathbb{E} [|\mathbf{w}^H \mathbf{u}(n)|^2], \\ \text{s.t. } \mathbf{w}^H \mathbf{x} &= 1, \end{aligned}$$

or equivalently

$$\begin{aligned} \mathbf{w}_* &:= \arg \min_{\mathbf{w}} \mathbf{w}^H \Sigma_u \mathbf{w}, \\ \text{s.t. } \mathbf{w}^H \mathbf{x} &= 1. \end{aligned} \quad (4.111)$$

This time, the beamformer is pushed to reduce its output signal, which, due to the presence of the constraint, is equivalent with optimally minimizing the contributions originating from the noise as well as from all other interference sources impinging on the array from different, to ϕ , directions. The resulting solution of (4.111) is obviously the same as (4.109) and (4.110) if one replaces Σ_{η} with Σ_u .

This type of linearly constrained task is known as *linearly constrained minimum variance* (LMV) or *Capon beamforming* or *minimum variance distortionless response* (MVDR) *beamforming*. For a concise introduction to beamforming, see, e.g., [48].

Widely linear versions for the beamforming task have also been proposed, e.g., [10, 32] (Problem 4.21).

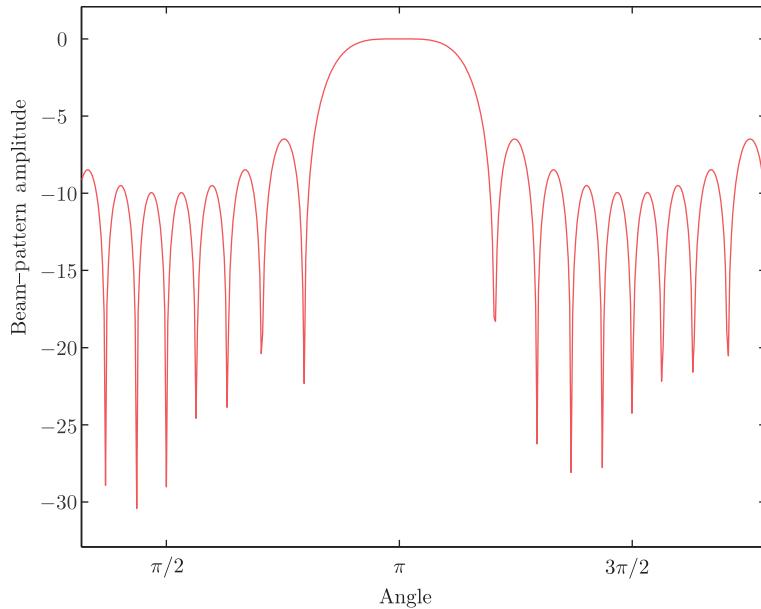
Figure 4.20 shows the resulting beam-pattern as a function of the angle ϕ . The desired angle for designing the optimal set of weights in (4.108) is $\phi = \pi$. The number of antenna elements is $l = 10$, the spacing has been chosen as $\frac{\Delta x}{\lambda} = 0.5$, and the noise covariance matrix as $0.1I$. The beam-pattern amplitude is in dBs, meaning the vertical axis shows $20 \log_{10}(|\mathbf{w}_*^H \mathbf{x}(\phi)|)$. Thus, any signal arriving from directions ϕ , not close to $\phi = \pi$, will be absorbed. The main beam can become sharper if more elements are used.

4.10 TIME-VARYING STATISTICS: KALMAN FILTERING

So far, our discussion about the linear estimation task has been limited to stationary environments, where the statistical properties of the involved random variables are assumed to be invariant with time. However, very often in practice this is not the case and the statistical properties may be different at different time instants. As a matter of fact, a large effort in the subsequent chapters will be devoted to studying the estimation task under time-varying environments.

Rudolf Kalman is the third scientist, after Wiener and Kolmogorov, whose significant contributions laid the foundations of estimation theory. Kalman is Hungarian-born and emigrated to the United States. He is the father of what is today known as system theory based on the state-space formulation, as opposed to the more limited input-output description of systems.

In 1960, in two seminal papers, Kalman proposed the celebrated Kalman filter, which exploits the state-space formulation in order to accommodate in an elegant way time-varying dynamics [18, 19].

**FIGURE 4.20**

The amplitude beam-pattern, in dBs, as a function of angle, ϕ with respect to the planar array.

We will derive the basic recursions of the Kalman filter in the general context of two jointly distributed random vectors \mathbf{y} , \mathbf{x} . The task is to estimate the values of \mathbf{x} given observations on \mathbf{y} . Let \mathbf{y} and \mathbf{x} be linearly related via the following set of recursions

$$\mathbf{x}_n = F_n \mathbf{x}_{n-1} + \boldsymbol{\eta}_n, \quad n \geq 0, \quad \text{State Equation,} \quad (4.112)$$

$$\mathbf{y}_n = H_n \mathbf{x}_n + \mathbf{v}_n, \quad n \geq 0, \quad \text{Output Equation,} \quad (4.113)$$

where $\boldsymbol{\eta}_n, \mathbf{x}_n \in \mathbb{R}^l$, $\mathbf{v}_n, \mathbf{y}_n \in \mathbb{R}^k$. The vector \mathbf{x}_n is known as the *state* of the system at time n and \mathbf{y}_n is the output, which is the vector that can be observed (measured); $\boldsymbol{\eta}_n$ and \mathbf{v}_n are the noise vectors, known as *process* noise and *measurement* noise, respectively. Matrices F_n and H_n are of appropriate dimensions and they are assumed to be known. Observe that the so-called *state equation* provides the information related to the time-varying dynamics of the corresponding system. It turns out that a large number of real-world tasks can be brought into the form of (4.112) and (4.113). The model is known as the *state-space* model for \mathbf{y}_n . In order to derive the time-varying estimator, $\hat{\mathbf{x}}_n$, given the measured values of \mathbf{y}_n , the following assumptions will be adopted:

- $\mathbb{E}[\boldsymbol{\eta}_n \boldsymbol{\eta}_n^\top] = Q_n$, $\mathbb{E}[\boldsymbol{\eta}_n \boldsymbol{\eta}_m^\top] = O$, $n \neq m$,
- $\mathbb{E}[\mathbf{v}_n \mathbf{v}_n^\top] = R_n$, $\mathbb{E}[\mathbf{v}_n \mathbf{v}_m^\top] = O$, $n \neq m$,
- $\mathbb{E}[\boldsymbol{\eta}_n \mathbf{v}_m^\top] = O$, $\forall n, m$,
- $\mathbb{E}[\boldsymbol{\eta}_n] = \mathbb{E}[\mathbf{v}_n] = \mathbf{0}$, $\forall n$,

where O denotes a matrix with zero elements. That is, η_n, v_n are uncorrelated; moreover, noise vectors at different time instants are also considered uncorrelated. Versions where some of these conditions are relaxed are also available. The respective covariance matrices, Q_n, R_n , are assumed to be known.

The development of the time-varying estimation task evolves around two types of estimators for the state variables:

- The first one is denoted as

$$\hat{\mathbf{x}}_{n|n-1},$$

and it is based on all information that has been received up to and including time instant $n - 1$; in other words, the obtained observations of $\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_{n-1}$. This is known as the *a priori* or *prior* estimator.

- The second estimator at time n is known as the *posterior* one, it is denoted as

$$\hat{\mathbf{x}}_{n|n},$$

and it is computed by updating $\hat{\mathbf{x}}_{n|n-1}$ after \mathbf{y}_n has been observed.

For the development of the algorithm, assume that at time $n - 1$ all required information is available; that is, the value of the posterior estimator as well the respective error covariance matrix

$$\hat{\mathbf{x}}_{n-1|n-1}, \quad P_{n-1|n-1} := \mathbb{E}[\mathbf{e}_{n-1|n-1}\mathbf{e}_{n-1|n-1}^T],$$

where

$$\mathbf{e}_{n-1|n-1} := \mathbf{x}_{n-1} - \hat{\mathbf{x}}_{n-1|n-1}.$$

Step 1: Using $\hat{\mathbf{x}}_{n-1|n-1}$, predict $\hat{\mathbf{x}}_{n|n-1}$ using the state equation; that is,

$$\hat{\mathbf{x}}_{n|n-1} = F_n \hat{\mathbf{x}}_{n-1|n-1}. \quad (4.114)$$

In other words, ignore the contribution from the noise. This is natural, because prediction cannot involve the unobserved variables.

Step 2: Obtain the respective error covariance matrix,

$$P_{n|n-1} = \mathbb{E}[(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1})^T]. \quad (4.115)$$

However,

$$\begin{aligned} \mathbf{e}_{n|n-1} &:= \mathbf{x}_n - \hat{\mathbf{x}}_{n|n-1} = F_n \mathbf{x}_{n-1} + \eta_n - F_n \hat{\mathbf{x}}_{n-1|n-1} \\ &= F_n \mathbf{e}_{n-1|n-1} + \eta_n. \end{aligned} \quad (4.116)$$

Combining (4.115) and (4.116), it is straightforward to see that

$$P_{n|n-1} = F_n P_{n-1|n-1} F_n^T + Q_n. \quad (4.117)$$

Step 3: Update $\hat{\mathbf{x}}_{n|n-1}$. To this end, adopt the following recursion

$$\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + K_n \mathbf{e}_n, \quad (4.118)$$

where

$$\mathbf{e}_n := \mathbf{y}_n - H_n \hat{\mathbf{x}}_{n|n-1}. \quad (4.119)$$

This time update recursion, once the observations for \mathbf{y}_n have been received, has a form that we will meet over and over again in this book. The “new” (posterior) estimate is equal to the “old” (prior) one, that is based on *the past history, plus a correction term*; the latter is proportional to the error \mathbf{e}_n in predicting the newly arrived observations vector and its prediction based on the “old” estimate. Matrix K_n , known as the *Kalman gain*, controls the amount of correction and its value is computed so as to minimize the mean-square error; in other words,

$$J(K_n) := \mathbb{E}[\mathbf{e}_{n|n}^T \mathbf{e}_{n|n}] = \text{trace}\{P_{n|n}\}, \quad (4.120)$$

where

$$P_{n|n} = \mathbb{E}[\mathbf{e}_{n|n} \mathbf{e}_{n|n}^T], \quad (4.121)$$

and

$$\mathbf{e}_{n|n} := \mathbf{x}_n - \hat{\mathbf{x}}_{n|n}.$$

It can be shown that, the optimal Kalman gain is equal to ([Problem 4.22](#))

$$K_n = P_{n|n-1} H_n^T S_n^{-1}, \quad (4.122)$$

where

$$S_n = R_n + H_n P_{n|n-1} H_n^T. \quad (4.123)$$

Step 4: The final recursion that is now needed in order to complete the scheme is that for the update of $P_{n|n}$. Combining the definitions in (4.119) and (4.121) with (4.118), the following results ([Problem 4.23](#)),

$$P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}. \quad (4.124)$$

The algorithm has now been derived. All that is now needed is to select the initial conditions, which are chosen such as

$$\hat{\mathbf{x}}_{1|0} = \mathbb{E}[\mathbf{x}_1] \quad (4.125)$$

$$P_{1|0} = \mathbb{E}[(\mathbf{x}_1 - \hat{\mathbf{x}}_{1|0})(\mathbf{x}_1 - \hat{\mathbf{x}}_{1|0})^T] = \Pi_0, \quad (4.126)$$

for some initial guess Π_0 . The Kalman algorithm is summarized in [Algorithm 4.2](#).

Algorithm 4.2 (Kalman filtering).

- Input: $F_n, H_n, Q_n, R_n, \mathbf{y}_n, n = 1, 2, \dots$
- Initialization:
 - $\hat{\mathbf{x}}_{1|0} = \mathbb{E}[\mathbf{x}_1]$
 - $P_{1|0} = \Pi_0$
- **For** $n = 1, 2, \dots$, **Do**
 - $S_n = R_n + H_n P_{n|n-1} H_n^T$
 - $K_n = P_{n|n-1} H_n^T S_n^{-1}$
 - $\hat{\mathbf{x}}_{n|n} = \hat{\mathbf{x}}_{n|n-1} + K_n (\mathbf{y}_n - H_n \hat{\mathbf{x}}_{n|n-1})$
 - $P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}$
 - $\hat{\mathbf{x}}_{n+1|n} = F_{n+1} \hat{\mathbf{x}}_{n|n}$
 - $P_{n+1|n} = F_{n+1} P_{n|n} F_{n+1}^T + Q_{n+1}$
- **End For**

For complex-valued variables, transposition is replaced by the Hermitian operation.

Remarks 4.7.

- Besides the previously derived basic scheme, there are a number of variants. Although, in theory, they are all equivalent, their practical implementation may lead to different performance. Observe that $P_{n|n}$ is computed as the difference of two positive definite matrices; this may lead to obtain a $P_{n|n}$ that is not positive definite, due to numerical errors. This can cause the algorithm to diverge. A popular alternative is the so-called *information filtering* scheme, which propagates the inverse state-error covariance matrices, $P_{n|n}^{-1}$, $P_{n|n-1}^{-1}$ [20]. In contrast, the scheme in [Algorithm 4.2](#) is known as the *covariance* Kalman algorithm ([Problem 4.24](#)).

To cope with the numerical stability issues, a family of algorithms propagates the factors of $P_{n|n}$ (or $P_{n|n}^{-1}$), resulting from the respective Cholesky factorization [5, 40].

- There are different approaches to arrive at the Kalman filtering recursions. An alternative derivation is based on the orthogonality principle applied to the so-called *innovations process* associated with the observation sequence, so that

$$\epsilon(n) = \mathbf{y}_n - \hat{\mathbf{y}}_{n|1:n-1},$$

where $\hat{\mathbf{y}}_{n|1:n-1}$ is the prediction based on the past observations history [17]. In [Chapter 17](#), we are going to rederive the Kalman recursions looking at it as a Bayesian network.

- Kalman filtering is a generalization of the optimal mean-square linear filtering. It can be shown that when the involved processes are stationary, Kalman filter converges in its steady-state to our familiar normal equations [31].
- Extended Kalman filters.* In (4.112) and (4.113), both the state as well as the output equations have a linear dependence on the state vector \mathbf{x}_n . Kalman filtering, in a more general formulation, can be cast as

$$\mathbf{x}_n = f_n(\mathbf{x}_{n-1}) + \eta_n,$$

$$\mathbf{y}_n = h_n(\mathbf{x}_n) + v_n,$$

where f_n and h_n are nonlinear vector functions. In the extended Kalman filtering (EKF), the idea is to *linearize* the functions $h_n(\cdot)$ and $f_n(\cdot)$, at each time instant, via their Taylor series expansions and keep the linear term only, so that

$$F_n = \frac{\partial f_n(\mathbf{x}_n)}{\partial \mathbf{x}_n} \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n-1|n-1}},$$

$$H_n = \frac{\partial h_n(\mathbf{x}_n)}{\partial \mathbf{x}_n} \Big|_{\mathbf{x}_n=\hat{\mathbf{x}}_{n|n-1}},$$

and then proceed by using the updates derived for the linear case.

By its very definition, the EKF is suboptimal and often in practice one may face divergence of the algorithm; in general, it must be stated that its practical implementation needs to be done with care. Having said that, it must be pointed out that it is heavily used in a number of practical systems.

Unscented Kalman Filters is an alternative way to cope with the nonlinearity, and the main idea springs from probabilistic arguments. A set of points are deterministically selected from a Gaussian approximation, of $p(\mathbf{x}_n | \mathbf{y}_1, \dots, \mathbf{y}_n)$; these points are propagated through the nonlinearities and estimates of the mean values and covariances are obtained [15]. Particle

filtering, to be discussed in [Chapter 17](#), is another powerful and popular approach to deal with nonlinear state-space models via probabilistic arguments.

More recently, extensions of Kalman filtering in reproducing kernel Hilbert spaces offers an alternative approach to deal with nonlinearities [\[52\]](#).

A number of Kalman filtering versions for distributed learning ([Chapter 5](#)) have appeared in, e.g., [\[9, 23, 33, 43\]](#). In the latter of the references, subspace learning methods are utilized in the prediction stage associated with the state variables.

- The literature on Kalman filtering is huge, especially when applications are concerned. The interested reader may consult more specialized texts, for example, [\[4, 8, 17\]](#) and the references therein.

Example 4.4 (Autoregressive Process Estimation). Let us consider an AR process ([Chapter 2](#)) of order l , represented as

$$\mathbf{x}_n = - \sum_{i=1}^l a_i \mathbf{x}_{n-i} + \eta_n, \quad (4.127)$$

where η_n is a white noise sequence of variance σ_η^2 . Our task is to obtain an estimate $\hat{\mathbf{x}}_n$ of \mathbf{x}_n , having observed a noisy version of it, y_n . The corresponding random variables are related as

$$y_n = \mathbf{x}_n + v_n. \quad (4.128)$$

To this end, the Kalman filtering formulation will be used. Note that the MSE linear estimation, presented in [Section 4.9](#), cannot be used here. As we have already discussed in [Chapter 2](#), an AR process is asymptotically stationary; for finite time samples, the initial conditions at time $n = 0$ are “remembered” by the process and the respective (second) order statistics are time dependent, hence it is a nonstationary process. However, Kalman filtering is specially suited for such cases.

Let us rewrite (4.127) and (4.128) as

$$\begin{bmatrix} \mathbf{x}_n \\ \mathbf{x}_{n-1} \\ \mathbf{x}_{n-2} \\ \vdots \\ \mathbf{x}_{n-l+1} \end{bmatrix} = \begin{bmatrix} -a_1 & -a_2 & \cdots & -a_{l-1} & -a_l \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_{n-1} \\ \mathbf{x}_{n-2} \\ \mathbf{x}_{n-3} \\ \vdots \\ \mathbf{x}_{n-l} \end{bmatrix} + \begin{bmatrix} \eta_n \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$y_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_n \\ \vdots \\ \mathbf{x}_{n-l+1} \end{bmatrix} + v_n$$

or

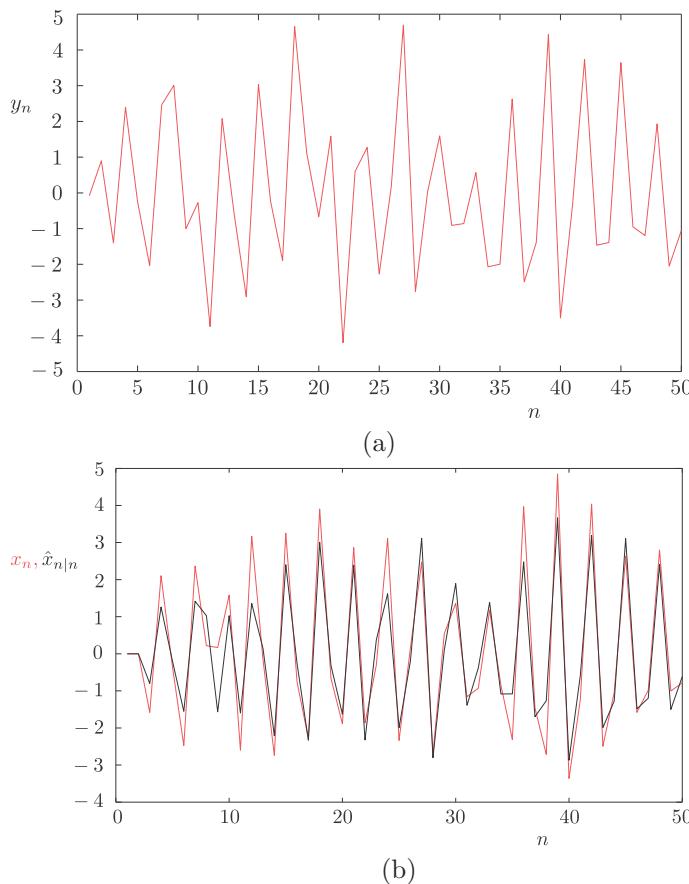
$$\mathbf{x}_n = F \mathbf{x}_{n-1} + \eta, \quad (4.129)$$

$$y_n = H \mathbf{x}_n + v_n, \quad (4.130)$$

where the definitions of $F_n \equiv F$ and $H_n \equiv H$ are obvious and

$$Q_n = \begin{bmatrix} \sigma_\eta^2 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \quad R_n = \sigma_v^2 \quad (\text{scalar}).$$

[Figure 4.21a](#) shows the values of a specific realization y_n , and [Figure 4.21b](#) the corresponding realization of the AR(2) (red) together with the predicted Kalman filter sequence $\hat{\mathbf{x}}_n$. Observe that the match is

**FIGURE 4.21**

(a) A realization of the observation sequence, y_n , which is used by the Kalman filter to obtain the predictions of the state variable. (b) The AR process (state variable) in red together with the predicted by the Kalman filter sequence (gray), for [Example 4.4](#). The Kalman filter has removed the effect of the noise v_n .

very good. For the generation of the AR process we used $l = 2$, $\alpha_1 = 0.95$, $\alpha_2 = 0.9$, $\sigma_\eta^2 = 0.5$. For the Kalman filter output noise, $\sigma_v^2 = 1$.

PROBLEMS

4.1 Show that the set of equations

$$\Sigma\theta = p$$

has a unique solution if $\Sigma > 0$ and infinite many if Σ is singular.

4.2 Show that the set of equations

$$\Sigma \theta = p$$

always has a solution.

4.3 Show that the shape of the isovalue contours of the mean-square error ($J(\theta)$) surface

$$J(\theta) = J(\theta_*) + (\theta - \theta_*)^T \Sigma (\theta - \theta_*)$$

are ellipses whose axes depend on the eigenstructure of Σ .

Hint. Assume that Σ has discrete eigenvalues.

4.4 Prove that if the true relation between the input x and the true output y is linear, meaning

$$y = \theta_o^T x + v_n, \quad \theta_o \in \mathbb{R}^l,$$

where v is independent of x , then, the optimal MSE estimate θ_* satisfies

$$\theta_* = \theta_o.$$

4.5 Show that if

$$y = \theta_o^T x + v, \quad \theta_o \in \mathbb{R}^k,$$

where v is independent of x , then the optimal MSE $\theta_* \in \mathbb{R}^l$, $l < k$ is equal to the top l components of θ_o , if the components of x are uncorrelated.

4.6 Derive the normal equations by minimizing the cost in (4.15).

Hint. Express the cost in terms of the real part θ_r and its imaginary part θ_i of θ and optimize with respect to θ_r, θ_i .

4.7 Consider the multichannel filtering task

$$\hat{y} = \begin{bmatrix} \hat{y}_r \\ \hat{y}_i \end{bmatrix} = \Theta \begin{bmatrix} x_r \\ x_i \end{bmatrix}.$$

Estimate Θ so that to minimize the error norm:

$$\mathbb{E}[||y - \hat{y}||^2].$$

4.8 Show that (4.34) is the same as (4.25).

4.9 Show that the MSE achieved by a linear complex-valued estimator is always larger than that obtained by a widely linear one. Equality is achieved only under the circularity conditions.

4.10 Show that under the second-order circularity assumption, the conditions in (4.39) hold true.

4.11 Show that if

$$f : \mathbb{C} \rightarrow \mathbb{R},$$

then the Cauchy-Riemann conditions are violated.

4.12 Derive the optimality condition in (4.45).

4.13 Show Eqs. (4.50) and (4.51).

4.14 Derive the normal equations for Example (4.2).

4.15 The input to the channel is a white noise sequence s_n of variance σ_s^2 . The output of the channel is the AR process

$$y_n = a_1 y_{n-1} + s_n. \tag{4.131}$$

The channel also adds white noise η_n of variance σ_η^2 . Design an optimal equalizer of order two, which at its output recovers an approximation of s_{n-L} . Sometimes, this equalization task is also known as whitening, because in this case the action of the equalizer is to “whiten” the AR process.

- 4.16** Show that the forward and backward MSE optimal predictors are conjugate reverse of each other.
4.17 Show that the MSE prediction errors ($\alpha_m^f = \alpha_m^b$) are updated according to the recursion

$$\alpha_m^b = \alpha_{m-1}^b (1 - |\kappa_{m-1}|^2).$$

- 4.18** Derive the BLUE for the Gauss-Markov theorem.
4.19 Show that the mean-square error (which in this case coincides with the variance of the estimator) of any linear unbiased estimator is higher than that associated with the BLUE.
4.20 Show that if Σ_η is positive definite, then $X^\top \Sigma_\eta^{-1} X$ is also positive definite if X is full rank.
4.21 Derive a MSE optimal linearly constrained widely linear beamformer.
4.22 Prove that the Kalman gain that minimizes the error covariance matrix

$$P_{n|n} = \mathbb{E}[(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})(\mathbf{x}_n - \hat{\mathbf{x}}_{n|n})^\top]$$

is given by

$$K_n = P_{n|n-1} H_n^\top (R_n + H_n P_{n|n-1} H_n^\top)^{-1}.$$

Hint. Use the following formulas

$$\frac{\partial \text{trace}\{AB\}}{\partial A} = B^\top \quad (\text{AB a square matrix})$$

$$\frac{\partial \text{trace}\{ACA^\top\}}{\partial A} = 2AC, \quad (C = C^\top).$$

- 4.23** Show that in Kalman filtering, the prior and posterior error covariance matrices are related as

$$P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}.$$

- 4.24** Derive the Kalman algorithm in terms of the inverse state-error covariance matrices, $P_{n|n}^{-1}$. In statistics, the inverse error covariance matrix is related to Fisher’s information matrix: hence, the name of the scheme.

MATLAB Exercises

- 4.25** Consider the image deblurring task described in [Section 4.6](#).
- Download the “boat” image from Waterloo’s Image repository.⁸ Alternatively, you may use any grayscale image of your choice. You can load the image into MATLAB’s memory using the “imread” function (also, you may want to apply the function “im2double” to get an array consisting of doubles).
 - Create a blurring point spread function (PSF) using MATLAB’s command “fspecial.” For example, you can write

⁸ <http://links.uwaterloo.ca/>.

```
PSF = fspecial('motion', 20, 45);
```

The blurring effect is produced using the “imfilter” function

```
J = imfilter(I, PSF, 'conv', 'circ');
```

where I is the original image.

- Add some white gaussian noise to the image using MATLAB’s function “imnoise,” as follows:

```
J = imnoise(J, 'gaussian', noise_mean, noise_var);
```

Use a small value of noise variance, such as 10^{-6} .

- To perform the deblurring, you need to employ the “deconvwnr” function. For example, if J is the array that contains the blurred image (with the noise) and PSF is the point spread function that produced the blurring, then the command

```
K = deconvwnr(J, PSF, C);
```

returns the deblurred image K, provided that the choice of C is reasonable. As a first attempt, select $C = 10^{-4}$. Use various values for C of your choice. Comment on the results.

- 4.26** Consider the noise cancelation task described in [Example 4.1](#). Write the necessary code to solve the problem using MATLAB according to the following steps:

- Create 5000 data samples of the signal $s_n = \cos(\omega_0 n)$, for $\omega_0 = 2 \times 10^{-3}\pi$.
- Create 5000 data samples of the AR process $v_1(n) = a_1 \cdot v_1(n-1) + \eta_n$ (initializing at zero), where η_n represents zero mean Gaussian noise with variance $\sigma_\eta^2 = 0.0025$ and $a_1 = 0.8$.
- Add the two sequences (i.e., $d_n = s_n + v_1(n)$) and plot the result. This represents the contaminated signal.
- Create 5000 data samples of the AR process $v_2(n) = a_2 v_2(n-1) + \eta_n$ (initializing at zero), where η_n represents the same noise sequence and $a_2 = 0.75$.
- Solve for the optimum (in the MSE sense) $w = [w_0, w_1]^T$. Create the sequence of the restored signal $\hat{s}_n = d_n - w_0 v_2(n) - w_1 v_2(n-1)$ and plot the result.
- Repeat steps 26b-26e using $a_2 = 0.9, 0.8, 0.7, 0.6, 0.5, 0.3$. Comment on the results.
- Repeat steps 26b-26e using $\sigma_\eta^2 = 0.01, 0.05, 0.1, 0.2, 0.5$, for $a_2 = 0.9, 0.8, 0.7, 0.6, 0.5, 0.3$. Comment on the results.

- 4.27** Consider the channel equalization task described in [Example 4.2](#). Write the necessary code to solve the problem using MATLAB according to the following steps:

- Create a signal s_n consisting of 50 equiprobable ± 1 samples. Plot the result using MATLAB’s function “stem.”
- Create the sequence $u_n = 0.5s_n + s_{n-1} + \eta_n$, where η_n denotes zero mean Gaussian noise with $\sigma_\eta^2 = 0.01$. Plot the result with “stem.”
- Find the optimal $w_* = [w_0, w_1, w_2]^T$, solving the normal equations.
- Construct the sequence of the reconstructed signal $\hat{s}_n = \text{sgn}(w_0 u_n + w_1 u_{n-1} + w_2 u_{n-2})$. Plot the result with “stem” using red color for the correctly reconstructed values (i.e., those that satisfy $s_n = \hat{s}_n$) and black color for errors.
- Repeat steps 27b-27d using different noise levels for σ_η^2 . Comment on the results.

- 4.28** Consider the autoregressive process estimation task described in [Example 4.4](#). Write the necessary code to solve the problem using MATLAB according to the following steps:
- Create 500 samples of the AR sequence $x_n = -a_1x_{n-1} - a_2x_{n-2} + \eta_n$ (initializing at zeros), where $a_1 = 0.2$, $a_2 = 0.1$, and η_n denotes zero mean Gaussian noise with $\sigma_n^2 = 0.5$.
 - Create the sequence $y_n = x_n + v_n$, where v_n denotes zero mean Gaussian noise with $\sigma_v^2 = 1$.
 - Implement the Kalman filtering algorithm as described in [Algorithm 4.2](#), using y_n as inputs and the matrices F, H, Q, R as described in [Example 4.4](#). To initialize the algorithm, you can use $\hat{x}_{1|0} = [0, 0]^T$ and $P_{1|0} = 0.1 \cdot I_2$. Plot the predicted values \hat{x}_n versus the original sequence x_n . Play with the values of the different parameters and comment on the obtained results.

REFERENCES

- [1] T. Adali, V.D. Calhoun, Complex ICA of brain imaging data, *IEEE Signal Process. Mag.* 24(5) (2007) 136-139.
- [2] T. Adali, H. Li, Complex-valued adaptive signal processing, in: T. Adali, S. Haykin (Eds.), *Adaptive Signal Processing: Next Generation Solutions*, John Wiley, 2010.
- [3] T. Adali, P. Schreier, Optimization and estimation of complex-valued signals: theory and applications in filtering and blind source separation, *IEEE Signal Process. Mag.* 31(5) (2014) 112-128.
- [4] B.D.O. Anderson, J.B. Moore, *Optimal Filtering*, Prentice Hall, Englewood Cliffs, NJ, 1979.
- [5] G.J. Bierman, *Factorization Methods for Discrete Sequential Estimation*, Academic Press, New York, 1977.
- [6] P. Bouboulis, S. Theodoridis, Extension of Wirtinger's calculus to Reproducing kernel Hilbert spaces and the complex kernel LMS, *IEEE Trans. Signal Process.* 53(3) (2011) 964-978.
- [7] D.H. Brandwood, A complex gradient operator and its application in adaptive array theory, *IEEE Proc.* 130(1) (1983) 11-16.
- [8] R.G. Brown, P.V.C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering*, second ed., John Wiley Sons, Inc., 1992.
- [9] F.S. Cattivelli, A.H. Sayed, Diffusion strategies for distributed Kalman filtering and smoothing, *IEEE Trans. Automat. Control* 55(9) (2010) 2069-2084.
- [10] P. Chevalier, J.P. Delmas, A. Oukaci, Optimal widely linear MVDR beamforming for noncircular signals, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, 2009, pp. 3573-3576.
- [11] P. Delsarte, Y. Genin, The split Levinson algorithm, *IEEE Trans. Signal Process.* 34 (1986) 470-478.
- [12] J. Dourbin, The fitting of time series models, *Rev. Int. Stat. Inst.* 28 (1960) 233-244.
- [13] Y.C. Eldar, Minimax, MSE estimation of deterministic parameters with noise covariance uncertainties, *IEEE Trans. Signal Process.* 54 (2006) 138-145.
- [14] R.C. Gonzalez, R.E. Woods, *Digital Image Processing*, Addison-Wesley, 1993.
- [15] S. Julier, A skewed approach to filtering, *Proc. SPIE* 3373 (1998) 271-282.
- [16] T. Kailath, An innovations approach to least-squares estimation: Part 1. Linear filtering in additive white noise, *IEEE Trans. Automat. Control* AC-13 (1968) 646-655.
- [17] T. Kailath, A.H. Sayed, B. Hassibi, *Linear Estimation*, Prentice Hall, Englewood Cliffs, 2000.
- [18] R.E. Kalman, A new approach to linear filtering and prediction problems, *Trans. ASME J. Basic Eng.* 82 (1960) 34-45.
- [19] R.E. Kalman, R.S. Bucy, New results in linear filtering and prediction theory, *Trans. ASME J. Basic Eng.* 83 (1961) 95-107.

- [20] P.G. Kaminski, A.E. Bryson, S.F. Schmidt, Discrete square root filtering: A survey, *IEEE Transactions on Automatic Control* 16 (1971) 727-735.
- [21] N. Kalouptsidis, S. Theodoridis, Parallel implementation of efficient LS algorithms for filtering and prediction, *IEEE Trans. Acoust. Speech Signal Process.* 35 (1987) 1565-1569.
- [22] N. Kalouptsidis, S. Theodoridis (Eds.), *Adaptive System Identification and Signal Processing Algorithms*, Prentice Hall, 1993.
- [23] U.A. Khan, J. Moura, Distributing the Kalman filter for large-scale systems, *IEEE Trans. Signal Process.* 56(10) (2008) 4919-4935.
- [24] A.N. Kolmogorov, Stationary sequences in Hilbert spaces, *Bull. Math. Univ. Moscow* 2 (1941) (in Russian).
- [25] K. Kreutz-Delgado, The complex Gradient Operator and the $\mathbb{C}\mathbb{R}$ -Calculus, <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.6515&rep=rep1&type=pdf>, 2006.
- [26] N. Levinson, The Wiener error criterion in filter design and prediction, *J. Math. Phys.* 25 (1947) 261-278.
- [27] H. Li, T. Adali, Optimization in the complex domain for nonlinear adaptive filtering, in: Proceedings, 33rd Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, 2006, pp. 263-267.
- [28] X.-L. Li, T. Adali, Complex-valued linear and widely linear filtering using MSE and Gaussian entropy, *IEEE Trans. Signal Process.* 60 (2012) 5672-5684.
- [29] D.J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.
- [30] D. Mandic, V.S.L. Guh, *Complex Valued Nonlinear Adaptive Filters*, John Wiley, 2009.
- [31] J.M. Mendel, *Lessons in Digital Estimation Theory*, Prentice Hall, Englewood Cliffs, NJ, 1995.
- [32] T. McWhorter, P. Schreier, Widely linear beamforming, in: Proceedings 37th Asilomar Conference on Signals, Systems, Computers, Pacific Grove, CA, 1993, pp. 759.
- [33] P.V. Overschee, B.D. Moor, *Subspace Identification for Linear Systems: Theory, Implementation, Applications*, Kluwer Academic Publishers, 1996.
- [34] M. Petrou, C. Petrou, *Image Processing: The fundamentals*, second ed., John Wiley, 2010.
- [35] B. Picinbono, On circularity, *IEEE Trans. Signal Process.* 42(12) (1994) 3473-3482.
- [36] B. Picinbono, P. Chevalier, Widely linear estimation with complex data, *IEEE Trans. Signal Process.* 43(8) (1995) 2030-2033.
- [37] B. Picinbono, *Random Signals and Systems*, Prentice Hall, 1993.
- [38] T. Piotrowski, I. Yamada, MV-PURE estimator: minimum-variance pseudo-unbiased reduced-rank estimator for linearly constrained ill-conditioned inverse problems, *IEEE Trans. Signal Process.* 56 (2008) 3408-3423.
- [39] I.R. Porteous, *Clifford Algebras and Classical Groups*, Cambridge University Press, 1995.
- [40] J.E. Potter, New statistical formulas, Space Guidance Analysis Memo, No 40, Instrumentation Laboratory, MIT, 1963.
- [41] J. Proakis, *Digital Communications*, second ed., McGraw Hill, 1989.
- [42] J.G. Proakis, D.G. Manolakis, *Digital Signal Processing: Principles, Algorithms and Applications*, second ed., MacMillan, 1992.
- [43] O.-S. Reza, Distributed Kalman filtering for sensor networks, in: Proceedings IEEE Conference on Decision and Control, 2007, pp. 5492-5498.
- [44] A.H. Sayed, *Fundamentals of Adaptive Filtering*, John Wiley, 2003.
- [45] J. Schur, Über Potenzreihen, die im Innern des Einheitskreises beschränkt sind, *J. Reine Angew. Math.* 147 (1917) 205-232.
- [46] K. Slavakis, P. Bouboulis, S. Theodoridis, Adaptive learning in complex reproducing kernel Hilbert spaces employing Wirtinger's Subgradients, *IEEE Trans. Neural Networks Learn. Syst.* 23(3) (2012) 425-438.
- [47] G. Strang, *Linear Algebra and its Applications*, fourth ed., Hartcourt Brace Jovanovich, 2005.
- [48] M. Viberg, *Introduction to Array Processing*, in: R. Chellappa, S. Theodoridis (Eds.), *Academic Library in Signal Processing*, vol. 3, Academic Press, 2014, pp. 463-499.

- [49] N. Wiener, E. Hopf, Über eine klasse singulärer integralgleichungen, S.B. Preuss. Akad. Wiss, 1931, pp. 696-706.
- [50] N. Wiener, Extrapolation, Interpolation and Smoothing of Stationary Time Series, MIT Press, Cambridge, MA, 1949.
- [51] W. Wirtinger, Zur formalen theorie der funktionen von mehr komplexen veränderlichen, Math. Ann. 97 (1927) 357-375.
- [52] P. Zhu, B. Chen, J.C. Principe, Learning nonlinear generative models of time series with a Kalman filter in RKHS, IEEE Trans. Signal Process. 62(1) (2014) 141-155.

STOCHASTIC GRADIENT DESCENT: THE LMS ALGORITHM AND ITS FAMILY

5

CHAPTER OUTLINE

5.1	Introduction	162
5.2	The Steepest Descent Method	163
5.3	Application to the Mean-Square Error Cost Function	167
	<i>Time-Varying Step-Sizes</i>	174
5.3.1	The Complex-Valued Case	175
5.4	Stochastic Approximation	177
	<i>Application to the MSE Linear Estimation</i>	178
5.5	The Least-Mean-Squares Adaptive Algorithm	179
5.5.1	Convergence and Steady-State Performance of the LMS in Stationary Environments ..	181
	<i>Convergence of the Parameter Error Vector</i>	181
5.5.2	Cumulative Loss Bounds	186
5.6	The Affine Projection Algorithm	188
	<i>Geometric Interpretation of APA</i>	189
	<i>Orthogonal Projections</i>	191
5.6.1	The Normalized LMS	193
5.7	The Complex-Valued Case	194
	<i>The Widely Linear LMS</i>	195
	<i>The Widely Linear APA</i>	195
5.8	Relatives of the LMS	196
	<i>The Sign-Error LMS</i>	196
	<i>The Least-Mean-Fourth (LMF) Algorithm</i>	196
	<i>Transform-Domain LMS</i>	197
5.9	Simulation Examples	199
5.10	Adaptive Decision Feedback Equalization	202
5.11	The Linearly Constrained LMS	204
5.12	Tracking Performance of the LMS in Nonstationary Environments	206
5.13	Distributed Learning: The Distributed LMS	208
5.13.1	Cooperation Strategies	209
	<i>Centralized Networks</i>	209
	<i>Decentralized Networks</i>	210
5.13.2	The Diffusion LMS	211
5.13.3	Convergence and Steady-State Performance: Some Highlights	218
5.13.4	Consensus-Based Distributed Schemes	220

5.14 A Case Study: Target Localization	222
5.15 Some Concluding Remarks: Consensus Matrix	223
Problems.....	224
MATLAB Exercises	226
References.....	227

5.1 INTRODUCTION

In Chapter 4, we introduced the notion of mean-square error (MSE) optimal linear estimation and stated the normal equations for computing the coefficients of the optimal estimator/filter. A prerequisite for the normal equations is the knowledge of the second order statistics of the involved processes/variables, so that the covariance matrix of the input and the input-output cross-correlation vector to be obtained. However, most often in practice, all the designer has at her/his disposal is a set of training points; thus, the covariance matrix and the cross-correlation vector have to be estimated somehow. More important, in a number of practical applications, the underlying statistics may be time varying. We discussed this scenario while introducing Kalman filtering. The path taken there was to adopt a state-space representation and assume that the time dynamics of the model were known. However, although Kalman filtering is an elegant tool, it does not scale well in high-dimensional spaces due to the involved matrix operations and inversions.

The focus of this chapter is to introduce *online learning* techniques for estimating the unknown parameter vector. These are time-iterative schemes, which update the available estimate every time a measurement set (input-output pair of observations) is acquired. Thus, in contrast to the so-called *batch processing* methods which process the whole block of data as a single entity, online algorithms operate on a single data point at a time; therefore, such schemes do not require the training data set to be known and stored in advance. Online algorithmic schemes learn the underlying statistics from the data in a *time iterative* fashion. Hence, one does not have to provide further statistical information. Another characteristic of the algorithmic family, to be developed and studied in this chapter, is its computational simplicity. The required complexity for updating the estimate of the unknown parameter vector is linear with respect to the number of the unknown parameters. This is one of the major reasons that have made such schemes very popular in a number of practical applications; besides complexity, we will discuss other reasons that have contributed to their popularity.

The fact that such learning algorithms work in a time-iterative mode gives them the agility to learn and *track* slow time variations of the statistics of the involved processes/variables; this is the reason these algorithms are also known as *time-adaptive* or simply *adaptive*, because they can adapt to the needs of a changing environment. Online/time-adaptive algorithms have been used extensively since the early 1960s in a wide range of applications including signal processing, control, and communications. More recently, the philosophy behind such schemes is gaining in popularity in the context of applications where data reside in large databases, with a massive number of training points; for such tasks, storing all the data points in the memory may not be possible, and they have to be considered one at a time. Moreover, the complexity of batch processing techniques can amount to prohibitive levels, for today's technology. The current trend is to refer to such applications as *big data* problems.

In this chapter, we focus on a very popular class of online/adaptive algorithms that springs from the classical gradient descent method for optimization. Although our emphasis will be on the squared error loss function, the same rationale can also be adopted for other (differentiable) loss functions. The case of nondifferentiable loss functions will be treated in [Chapter 8](#). The online processing rationale will be a recurrent theme in this book.

5.2 THE STEEPEST DESCENT METHOD

Our starting point is the method of *gradient descent*, one of the most widely used methods for iterative minimization of a differentiable cost function, $J(\theta)$, $\theta \in \mathbb{R}^l$. As does any other iterative technique, the method starts from an initial estimate, $\theta^{(0)}$, and generates a sequence, $\theta^{(i)}$, $i = 1, 2, \dots$, such that,

$$\theta^{(i)} = \theta^{(i-1)} + \mu_i \Delta\theta^{(i)}, \quad i > 0, \quad (5.1)$$

where $\mu_i > 0$. All the schemes for the iterative minimization of a cost function, which we will deal with in this book, have the general form of (5.1). Their differences are in the way that μ_i and $\Delta\theta^{(i)}$ are chosen; the latter vector is known as the *update direction* or the *search direction*. The sequence μ_i is known as the step-size or the *step length*, at the i th iteration; note that the values of μ_i may either be constant or change at each iteration. In the gradient descent method, the choice of $\Delta\theta^{(i)}$ is done to guarantee that

$$J(\theta^{(i)}) < J(\theta^{(i-1)}),$$

except at a minimizer, θ_* .

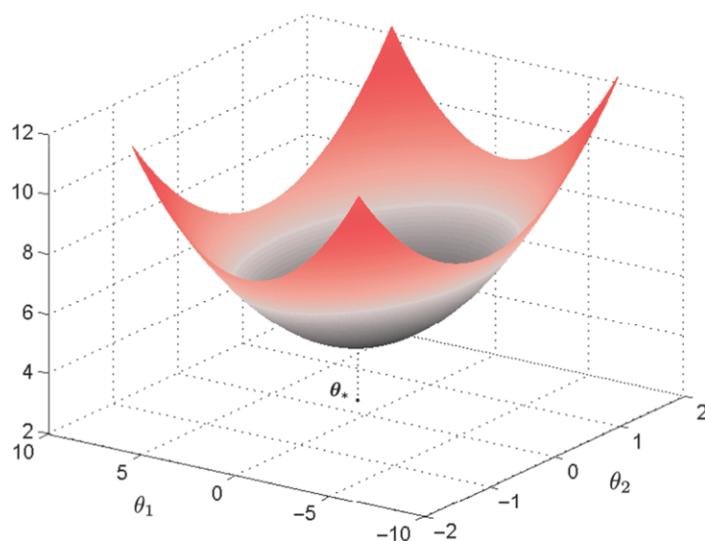
Assume that at the $i - 1$ iteration step the value $\theta^{(i-1)}$ has been obtained. Then, mobilizing a first order Taylor's expansion around $\theta^{(i-1)}$ we can write

$$J(\theta^{(i)}) = J(\theta^{(i-1)} + \mu_i \Delta\theta^{(i)}) \approx J(\theta^{(i-1)}) + \mu_i \nabla^T J(\theta^{(i-1)}) \Delta\theta^{(i)}.$$

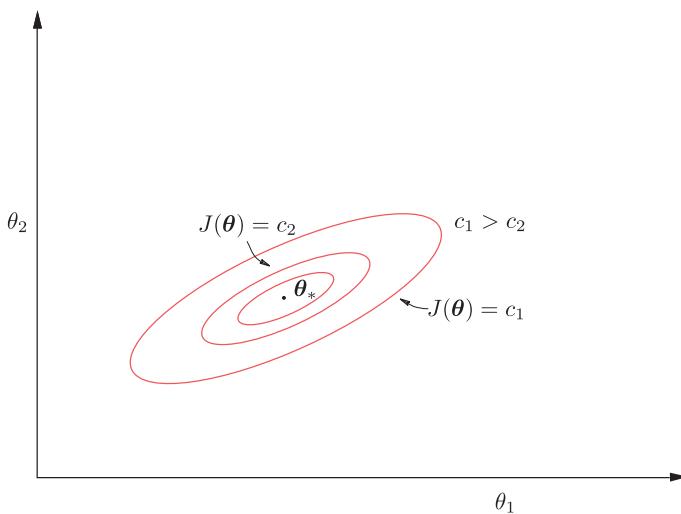
Selecting the search direction so that

$$\nabla^T J(\theta^{(i-1)}) \Delta\theta^{(i)} < 0, \quad (5.2)$$

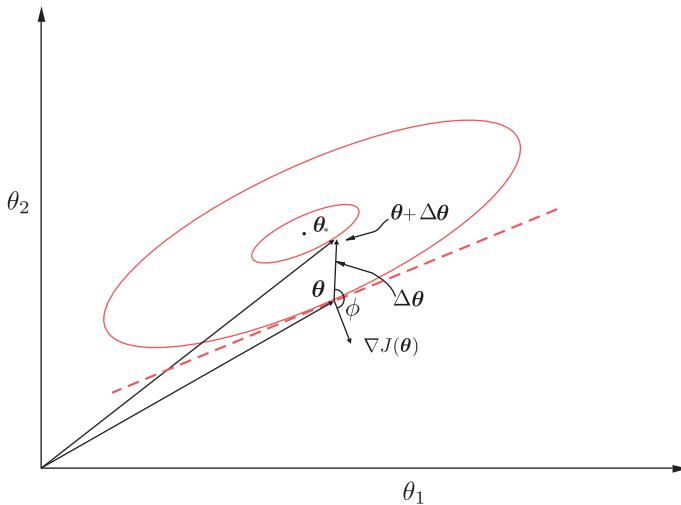
then it guarantees that $J(\theta^{(i-1)} + \mu_i \Delta\theta^{(i)}) < J(\theta^{(i-1)})$. For such a choice, $\Delta\theta^{(i)}$ and $\nabla J(\theta^{(i-1)})$ must form an *obtuse angle*. [Figure 5.1](#) shows the graph of a cost function in the two-dimensional case, $\theta \in \mathbb{R}^2$, and [Figure 5.2](#) shows the respective isovalue contours in the two-dimensional plane. Note that, in general, the contours can have any shape and are not necessarily ellipses; it all depends on the functional form of $J(\theta)$. However, because $J(\theta)$ has been assumed differentiable, the contours must be smooth and accept at any point a (unique) tangent plane, as this is defined by the respective gradient. Furthermore, recall from basic calculus that the gradient vector, $\nabla J(\theta)$, is perpendicular to the plane (line) tangent to the corresponding isovalue contour, at the point θ ([Problem 5.1](#)). The geometry is illustrated in [Figure 5.3](#); to facilitate the drawing and unclutter notation, we have removed the iteration index i . Note that by selecting the search direction which forms an obtuse angle with the gradient, it places $\theta^{(i-1)} + \mu_i \Delta\theta^{(i)}$ at a point on a contour which corresponds to a lower value of $J(\theta)$. Two issues are now raised: (a) to choose the best search direction along which to move and (b) to compute how far along this direction one can go. Even without much mathematics, it is obvious from [Figure 5.3](#) that if

**FIGURE 5.1**

A cost function in the two-dimensional parameter space.

**FIGURE 5.2**

The corresponding isovalue curves of the cost function of Figure 5.1, in the two-dimensional plane.

**FIGURE 5.3**

The gradient vector at a point θ is perpendicular to the tangent plane at the isovalue curve crossing θ . The descent direction forms an obtuse angle, ϕ , with the gradient vector.

$\mu_i \|\Delta\theta^{(i)}\|$ is too large, then the new point can be placed on a contour corresponding to a larger value than that of the current contour; after all, the first order Taylor's expansion holds approximately true for small deviations from $\theta^{(i-1)}$.

To address the first of the two issues, let us assume $\mu_i = 1$ and search for all vectors, z , with unit Euclidean norm, $\theta^{(i-1)}$. Then, it does not take long to see that for all possible directions, the one that gives the most negative value of the inner product, $\nabla^T J(\theta^{(i-1)}) z$, is that of the negative gradient

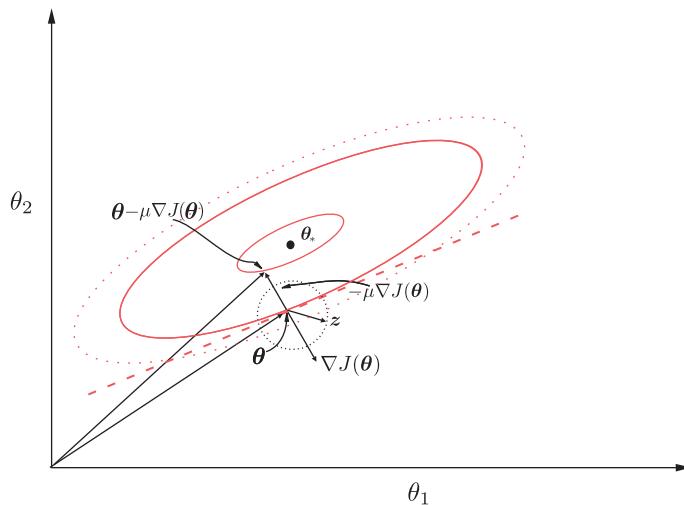
$$z = -\frac{\nabla J(\theta^{(i-1)})}{\|\nabla J(\theta^{(i-1)})\|}.$$

This is illustrated in [Figure 5.4](#). Center the unit Euclidean norm ball at $\theta^{(i-1)}$. Then from all the unit norm vectors having their origin at $\theta^{(i-1)}$ choose the one pointing to the negative gradient direction. Thus, for all unit Euclidean norm vectors, the steepest descent direction coincides with the (negative) *gradient descent* direction and the corresponding update recursion becomes

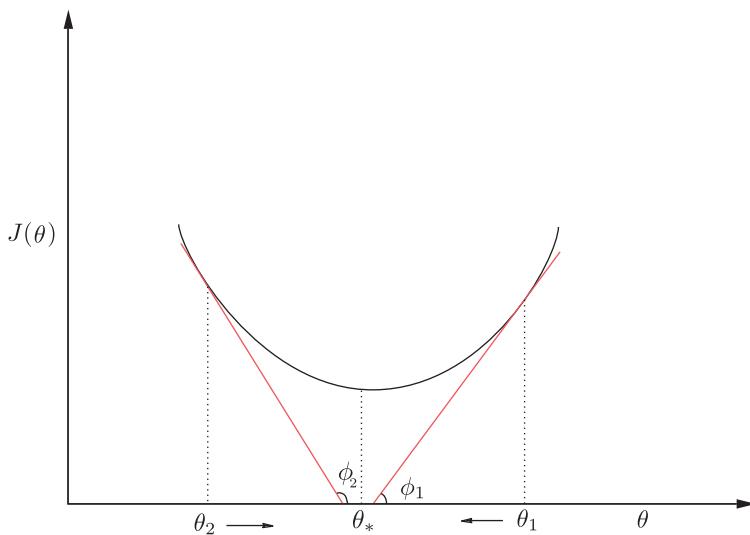
$$\theta^{(i)} = \theta^{(i-1)} - \mu_i \nabla J(\theta^{(i-1)}) : \quad \text{Gradient Descent Scheme.} \quad (5.3)$$

Note that we still have to address the second point, concerning the choice of μ_i . The choice must be done in such a way to guarantee convergence of the minimizing sequence. We will come to this issue soon.

Iteration (5.3) is illustrated in [Figure 5.5](#) for the one-dimensional case. If at the current iteration the algorithm has “landed” at θ_1 , then the derivative of $J(\theta)$ at this point is positive (the tangent of an acute angle, ϕ_1), and this will force the update to move to the left towards the minimum. The scenario is different if the current estimate was θ_2 . The derivative is negative (the tangent of an obtuse angle, ϕ_2) and this will push the update to the right toward, again, the minimum. Note, however, that it is important how far to the left or to the right one has to move. A large move from, say θ_1 , to the left may land the

**FIGURE 5.4**

From all the descent directions of unit Euclidean norm (dotted circle), the negative gradient one leads to the maximum decrease of the cost function.

**FIGURE 5.5**

Once the algorithm is at θ_1 , the gradient descent will move the point to the left, towards the minimum. The opposite is true for point θ_2 .

update on the other side of the optimal value. In such a case, the algorithm may oscillate around the minimum and never converge. A major effort in this chapter will be devoted in providing theoretical frameworks for establishing bounds for the values of the step-size that guarantee convergence.

The gradient descent method exhibits approximately *linear convergence*; that is, the error between $\boldsymbol{\theta}^{(i)}$ and the true minimum converges to zero asymptotically in the form of a *geometric series*. However, the convergence rate depends heavily on the condition number of the Hessian matrix of $J(\boldsymbol{\theta})$. For very large values of the condition number, such as 1000, the rate of convergence can become extremely slow. The great advantage of the method lies in its low computational requirements.

Finally, it has to be pointed out that we arrived at the scheme in Eq. (5.3) by searching all directions via the unit Euclidean norm. However, there is nothing “sacred” around Euclidean norms. One can employ other norms, such as the ℓ_1 or the quadratic $\mathbf{v}^T P \mathbf{v}$ norms, where P is a positive definite matrix. Under such choices, one will end up with alternative update iterations (see e.g., [23]). We will return to this point in [Chapter 6](#) when dealing with Newton’s iterative minimization scheme.

5.3 APPLICATION TO THE MEAN-SQUARE ERROR COST FUNCTION

Let us apply the gradient descent scheme to derive an iterative algorithm to minimize our familiar, from the previous chapter, cost function

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E}[(\mathbf{y} - \boldsymbol{\theta}^T \mathbf{x})^2] \\ &= \sigma_y^2 - 2\boldsymbol{\theta}^T \mathbf{p} + \boldsymbol{\theta}^T \Sigma_x \boldsymbol{\theta}, \end{aligned} \quad (5.4)$$

where

$$\nabla J(\boldsymbol{\theta}) = 2\Sigma_x \boldsymbol{\theta} - 2\mathbf{p}, \quad (5.5)$$

and the notation has been defined in [Chapter 4](#). In this chapter, we will also adhere to zero mean jointly distributed input-output random variables, except if otherwise stated. Thus, the covariance and correlation matrices coincide. If this is not the case, the covariance in (5.5) is replaced by the correlation matrix. The treatment is focused on real data and we will point out differences with the complex-valued data case whenever needed.

Employing (5.5), the update recursion in (5.3) becomes

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu (\Sigma_x \boldsymbol{\theta}^{(i-1)} - \mathbf{p}) \\ &= \boldsymbol{\theta}^{(i-1)} + \mu (\mathbf{p} - \Sigma_x \boldsymbol{\theta}^{(i-1)}), \end{aligned} \quad (5.6)$$

where the step-size has been considered constant and it has also absorbed the factor 2. The more general case of iteration-dependent values of the step-size will be discussed soon after. Our goal now becomes that of searching for all values of μ that guarantee convergence. To this end, define

$$\mathbf{c}^{(i)} := \boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*, \quad (5.7)$$

where $\boldsymbol{\theta}_*$ is the (unique) optimal MSE solution that results by solving the respective normal equations,

$$\Sigma_x \boldsymbol{\theta}_* = \mathbf{p}.$$

Subtracting θ_* from both sides of (5.6) and plugging in (5.7), we obtain

$$\begin{aligned}\mathbf{c}^{(i)} &= \mathbf{c}^{(i-1)} + \mu (\mathbf{p} - \Sigma_x \mathbf{c}^{(i-1)} - \Sigma_x \theta_*) \\ &= \mathbf{c}^{(i-1)} - \mu \Sigma_x \mathbf{c}^{(i-1)} = (I - \mu \Sigma_x) \mathbf{c}^{(i-1)}.\end{aligned}\quad (5.8)$$

Recall that Σ_x is a symmetric positive definite matrix (Chapter 2), hence (Appendix A.2) it can be written as

$$\Sigma_x = Q \Lambda Q^T, \quad (5.9)$$

where

$$\Lambda := \text{diag}\{\lambda_1, \dots, \lambda_l\} \quad \text{and} \quad Q := [\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_l],$$

with $\lambda_j, \mathbf{q}_j, j = 1, 2, \dots, l$, being the (positive) eigenvalues and the respective normalized (*orthogonal*) eigenvectors of the covariance matrix,¹ represented by

$$\mathbf{q}_k^T \mathbf{q}_j = \delta_{kj}, \quad k, j = 1, 2, \dots, l \implies Q^T = Q^{-1}.$$

That is, the matrix Q is orthogonal. Plugging the factorization of Σ_x into (5.8), we obtain

$$\mathbf{c}^{(i)} = Q (I - \mu \Lambda) Q^T \mathbf{c}^{(i-1)},$$

or

$$\mathbf{v}^{(i)} = (I - \mu \Lambda) \mathbf{v}^{(i-1)}, \quad (5.10)$$

where

$$\mathbf{v}^{(i)} := Q^T \mathbf{c}^{(i)}, \quad i = 1, 2, \dots. \quad (5.11)$$

The previously used “trick” is a standard one and its aim is to “decouple” the various components of $\theta^{(i)}$ in (5.6). Indeed, each one of the components, $v^{(i)}(j), j = 1, 2, \dots, l$, of $\mathbf{v}^{(i)}$ follows an iteration path, which is independent of the rest of the components; in other words,

$$\begin{aligned}v^{(i)}(j) &= (1 - \mu \lambda_j) v^{(i-1)}(j) = (1 - \mu \lambda_j)^2 v^{(i-2)}(j) \\ &\quad \cdots = (1 - \mu \lambda_j)^i v^{(0)}(j),\end{aligned}\quad (5.12)$$

where $v^{(0)}(j)$ is the j th component of $\mathbf{v}^{(0)}$, corresponding to the initial vector. It is readily seen that if

$$|1 - \mu \lambda_j| < 1 \iff -1 < 1 - \mu \lambda_j < 1, \quad j = 1, 2, \dots, l, \quad (5.13)$$

the geometric series tends to zero and

$$\mathbf{v}^{(i)} \rightarrow \mathbf{0} \implies Q^T (\theta^{(i)} - \theta_*) \rightarrow \mathbf{0} \implies \theta^{(i)} \rightarrow \theta_*. \quad (5.14)$$

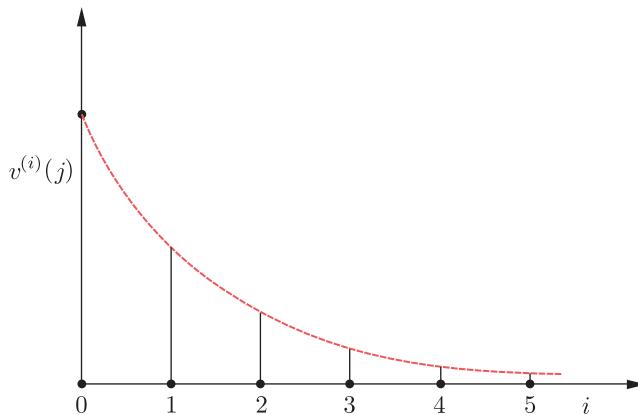
Note that (5.13) is equivalent to

$0 < \mu < 2/\lambda_{\max} : \quad \text{Condition for Convergence,}$

(5.15)

where λ_{\max} denotes the maximum eigenvalue of Σ_x .

¹ In contrast to other chapters, we denote eigenvectors with \mathbf{q} and not as \mathbf{u} , since at some places the latter is used to denote the input random vector.

**FIGURE 5.6**

Convergence curve for one of the components of the transformed error vector. Note that the curve is of an approximate exponentially decreasing type.

Time constant: Figure 5.6 shows a typical sketch of the evolution of $v^{(i)}(j)$ as a function of the iteration steps for the case $0 < 1 - \mu\lambda_j < 1$. Assume that the envelope, denoted by the red line is (approximately) of an exponential form, $f(t) = \exp(-t/\tau_j)$. Plug into $f(t)$, as the values corresponding at the time instants, $t = iT$ and $t = (i-1)T$, the values of $v^{(i)}(j)$, $v^{(i-1)}(j)$ from (5.12); then, the *time constant* results as

$$\tau_j = \frac{-1}{\ln(1 - \mu\lambda_j)},$$

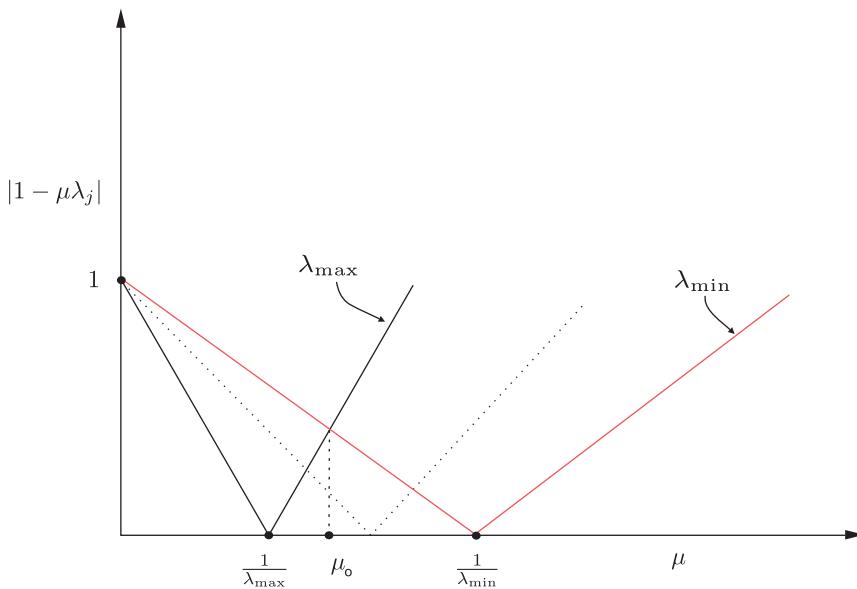
assuming that the sampling time between two successive iterations is $T = 1$. For small values of μ , we can write

$$\tau_j \approx \frac{1}{\mu\lambda_j}, \quad \text{for } \mu \ll 1.$$

That is, the slowest rate of convergence is associated with the component that corresponds to the smallest eigenvalue. However, this is only true for small enough values of μ . For the more general case, this may not be true. Recall that the rate of convergence depends on the value of the term $1 - \mu\lambda_j$. This is also known as the *jth mode*. Its value depends not only on λ_j but also on μ . Let us consider as an example the case of μ taking a value very close to the maximum allowable one, $\mu \simeq 2/\lambda_{\max}$. Then, the mode corresponding to the maximum eigenvalue will have an absolute value very close to one. On the other hand, the time constant of the mode corresponding to the minimum eigenvalue will be controlled by the value of $|1 - 2\lambda_{\min}/\lambda_{\max}|$, which can be much smaller than one. In such a case, the mode corresponding to the maximum eigenvalue exhibits slower convergence.

To obtain the optimum value for the step-size, one has to select its value in such a way that the resulting maximum absolute mode value is minimized. This is a min/max task,

$$\begin{aligned} \mu_o &= \arg \min_{\mu} \max_j |1 - \mu\lambda_j|, \\ \text{s.t.} \quad &|1 - \mu\lambda_j| < 1, j = 1, 2, \dots, l. \end{aligned}$$

**FIGURE 5.7**

For each mode, increasing the value of the step-size, the time constant starts decreasing and then after a point starts increasing. The full black line corresponds to the maximum eigenvalue, the red one to the minimum, and the dotted curve to an intermediate eigenvalue. The overall optimal, μ_o , corresponds to the value where the red and the full black curves intersect.

The task can be solved easily graphically. Figure 5.7 shows the absolute values of the modes (corresponding to the maximum, minimum, and an intermediate one eigenvalues). The (absolute) values of the modes initially decrease, as μ increases and then they start increasing. Observe that the optimal value results at the point where the curves for the maximum and minimum eigenvalues intersect. Indeed, this corresponds to the minimum-maximum value. Moving μ away from μ_o , the maximum mode value increases; increasing μ_o , the mode corresponding to the maximum eigenvalue becomes larger and decreasing it, the mode corresponding to the minimum eigenvalue is increased. At the intersection, we have

$$1 - \mu_o \lambda_{\min} = -(1 - \mu_o \lambda_{\max}),$$

which results in

$$\mu_o = \frac{2}{\lambda_{\max} + \lambda_{\min}}. \quad (5.16)$$

At the optimal value, μ_o , there are two slowest modes; one corresponding to λ_{\min} (i.e., $1 - \mu_o \lambda_{\min}$) and another one corresponding to λ_{\max} (i.e., $1 - \mu_o \lambda_{\max}$). They have equal magnitudes but opposite signs, and they are given by,

$$\pm \frac{\rho - 1}{\rho + 1},$$

where

$$\rho := \frac{\lambda_{\max}}{\lambda_{\min}}.$$

In other words, the *convergence rate depends on the eigenvalues spread of the covariance matrix*.

Parameter Error Vector Convergence: From the definitions in (5.7) and (5.11), we get

$$\begin{aligned}\boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}_* + Q\mathbf{v}^{(i)} \\ &= \boldsymbol{\theta}_* + [\mathbf{q}_1, \dots, \mathbf{q}_l][v^{(i)}(1), \dots, v^{(i)}(l)]^T \\ &= \boldsymbol{\theta}_* + \sum_{k=1}^l \mathbf{q}_k v^{(i)}(k),\end{aligned}\tag{5.17}$$

or

$$\theta^{(i)}(j) = \theta_*(j) + \sum_{k=1}^l q_k(j) v^{(0)}(k)(1 - \mu\lambda_k)^i, \quad j = 1, 2, \dots, l.\tag{5.18}$$

In other words, the components of $\boldsymbol{\theta}^{(i)}$ converge to the respective components of the optimum vector $\boldsymbol{\theta}_*$ as a weighted average of exponentials, $(1 - \mu\lambda_k)^i$. Computing the respective time constant in close form is not possible; however, we can state lower and upper bounds. The lower bound corresponds to the time constant of the fastest converging mode and the upper bound to the slowest of the modes. For small values of $\mu \ll 1$, we can write

$$\frac{1}{\mu\lambda_{\max}} \leq \tau \leq \frac{1}{\mu\lambda_{\min}}.\tag{5.19}$$

The Learning Curve: We now turn our focus on the mean-square error. Recall from (4.8) that

$$J(\boldsymbol{\theta}^{(i)}) = J(\boldsymbol{\theta}_*) + (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*)^T \Sigma_x (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*),\tag{5.20}$$

or, mobilizing (5.17) and (5.9) and taking into consideration the orthonormality of the eigenvectors, we obtain

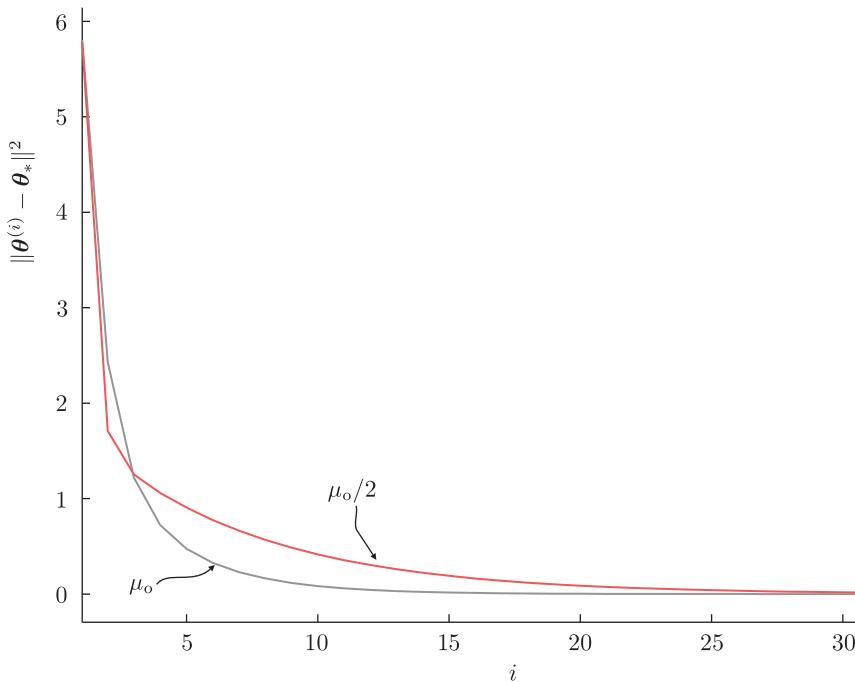
$$\begin{aligned}J(\boldsymbol{\theta}^{(i)}) &= J(\boldsymbol{\theta}_*) + \sum_{j=1}^l \lambda_j |v^{(i)}(j)|^2 \implies \\ J(\boldsymbol{\theta}^{(i)}) &= J(\boldsymbol{\theta}_*) + \sum_{j=1}^l \lambda_j (1 - \mu\lambda_j)^{2i} |v^{(0)}(j)|^2,\end{aligned}\tag{5.21}$$

which converges to the minimum value $J(\boldsymbol{\theta}_*)$ asymptotically. Moreover, observe that this convergence is monotonic, because $\lambda_j(1 - \mu\lambda_j)^2$ is positive. Following similar arguments as before, the respective time constants for each one of the modes are now,

$$\tau_j^{\text{mse}} = \frac{-1}{2 \ln(1 - \mu\lambda_j)} \approx \frac{1}{2\mu\lambda_j}.\tag{5.22}$$

Example 5.1. The aim of the example is to demonstrate what we have said so far, concerning the convergence issues of the gradient descent scheme in (5.6). The cross-correlation vector was chosen to be

$$\mathbf{p} = [0.05, 0.03]^T,$$

**FIGURE 5.8**

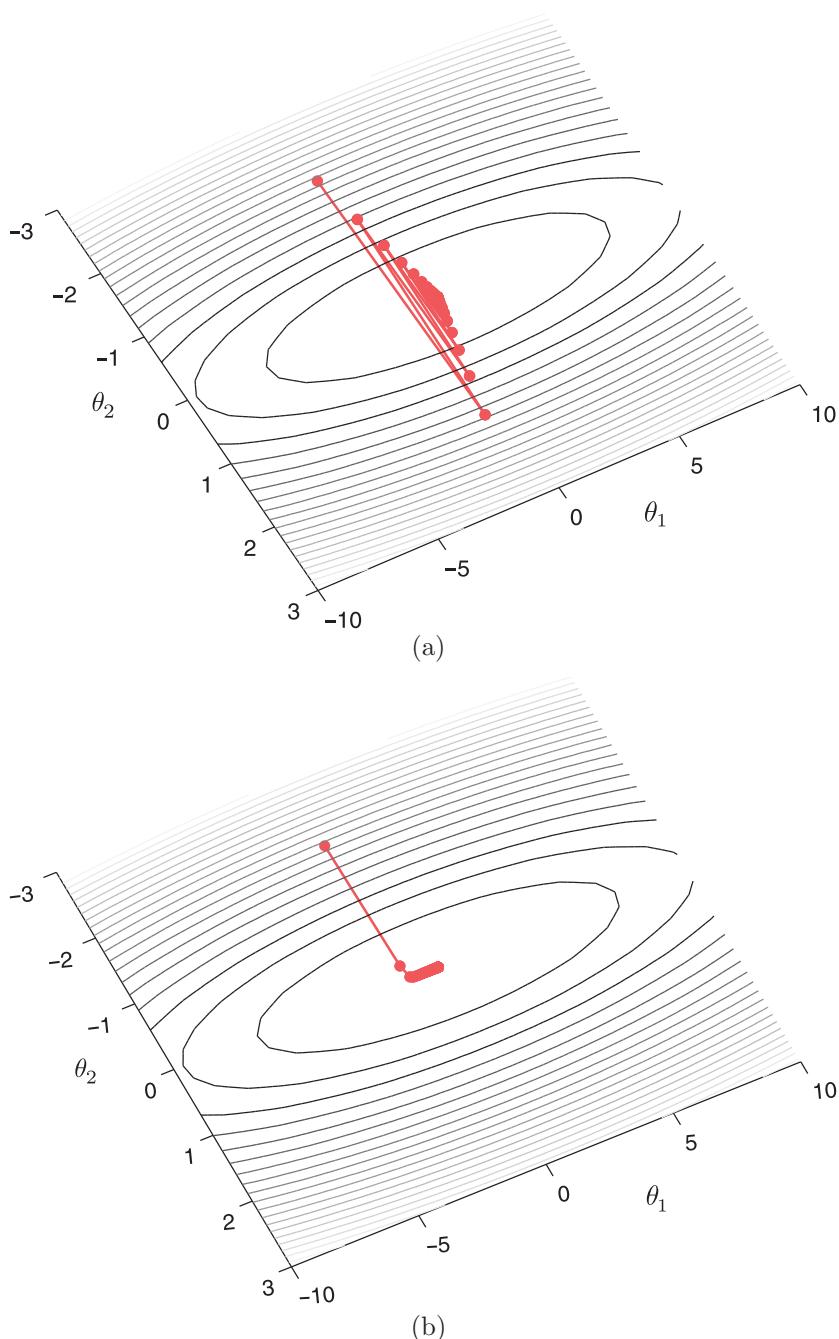
The black curve corresponds to the optimal value $\mu = \mu_o$ and the gray one to $\mu = \mu_o/2$, for the case of an input covariance matrix with unequal eigenvalues.

and we consider two different covariance matrices,

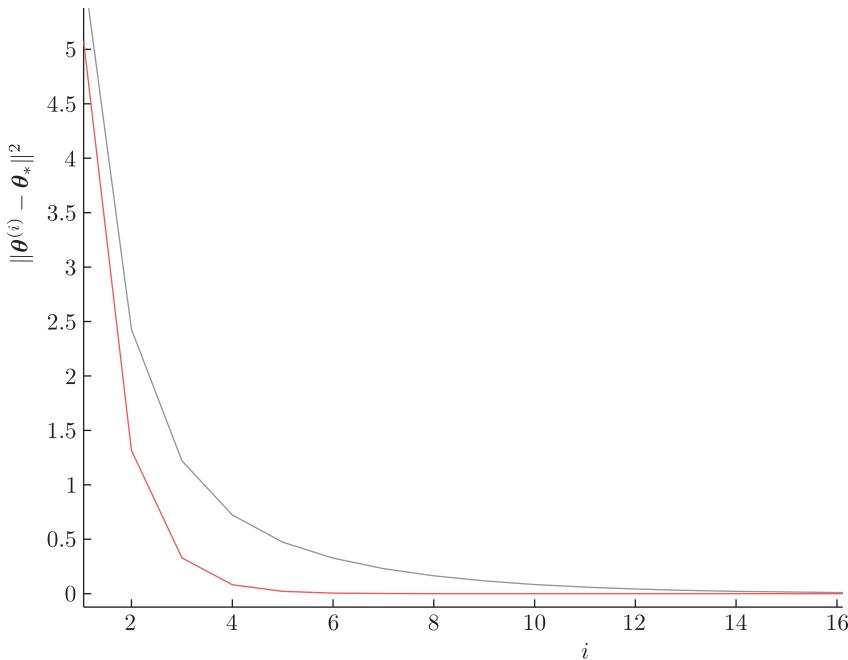
$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Note that, for the case of Σ_2 , both eigenvalues are equal to 1, and for Σ_1 they are $\lambda_1 = 1$ and $\lambda_2 = 0.1$ (for diagonal matrices the eigenvalues are equal to the diagonal elements of the matrix).

[Figure 5.8](#) shows the error curves for two values of μ , for the case of Σ_1 ; the gray one corresponds to the optimum value ($\mu_o = 1.81$) and the red one to $\mu = \mu_o/2 = 0.9$. Observe the faster convergence towards zero that is achieved by the optimal value. Note that it may happen, as is the case in [Figure 5.8](#), that initially the convergence for some $\mu \neq \mu_o$ will be faster compared to μ_o . What the theory guarantees is that, eventually, the curve corresponding to the optimal will tend to zero faster than for any other value of μ . [Figure 5.9](#) shows the respective trajectories of the successive estimates in the two-dimensional space, together with the isovalue curves; the latter are ellipses, as we can readily deduce if we look carefully at the form of the quadratic cost function written as in [\(5.20\)](#). Observe the zig-zag path, which corresponds to the larger value of $\mu = 1.81$ compared to the smoother one obtained for the smaller step-size $\mu = 0.9$.

**FIGURE 5.9**

The trajectories of the successive estimates (dots) obtained by the gradient descent algorithm for (a) the larger value of $\mu = 1.81$ and (b) for the smaller value of $\mu = 0.9$. In (b), the trajectory toward the minimum is smooth. In contrast, in (a), the trajectory consists of zig-zags.

**FIGURE 5.10**

For the same value of $\mu = 1.81$, the error curves for the case of unequal eigenvalues ($\lambda_1 = 1$ and $\lambda_2 = 0.1$) (red) and for equal eigenvalues ($\lambda_1 = \lambda_2 = 1$). For the latter case, the isovalue curves are circles; if the optimal value $\mu_o = 1$ is used, the algorithm converges in one step. This is demonstrated in Figure 5.11.

For comparison reasons, to demonstrate the dependence of the convergence speed on the eigenvalues spread, Figure 5.10 shows the error curves using the same step size, $\mu = 1.81$, for both cases, Σ_1 and Σ_2 . Observe that large eigenvalues spread of the input covariance matrix slows down the convergence rate. Note that if the eigenvalues of the covariance matrix are equal to, say, λ , the isovalue curves are circles; the optimal step size in this case is $\mu = 1/\lambda$ and convergence is achieved in only one step, Figure 5.11.

Time-varying step-sizes

The previous analysis cannot be carried out for the case of an iteration-dependent step-size. It can be shown (Problem 5.2), that in this case, the gradient descent algorithm converges if

- $\mu_i \rightarrow 0$, as $i \rightarrow \infty$
- $\sum_{i=1}^{\infty} \mu_i = \infty$.

A typical example of sequences, which comply with both conditions, are those that satisfy the following:

$$\sum_{i=1}^{\infty} \mu_i^2 < \infty, \quad \sum_{i=1}^{\infty} \mu_i = \infty, \quad (5.23)$$

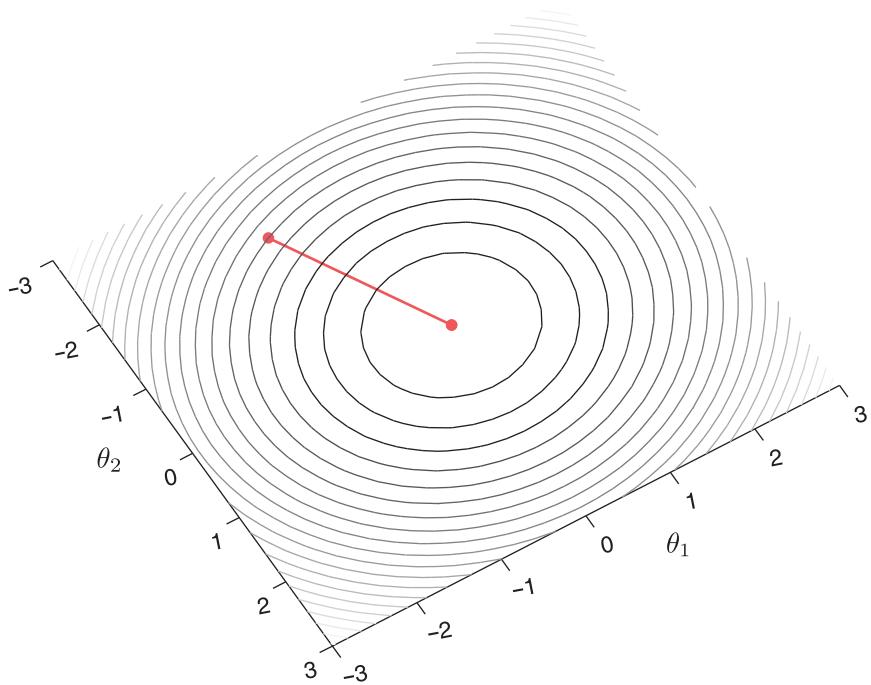


FIGURE 5.11

When the eigenvalues of the covariance matrix are all equal to a value, λ , the use of the optimal $\mu_o = 1/\lambda$ achieves convergence in one step.

as, for example, the sequence,

$$\mu_i = \frac{1}{i}.$$

Note that the two (sufficient) conditions require that the sequence tends to zero, yet its infinite sum diverges. We will meet this pair of conditions in various parts of this book. The previous conditions state that the step-size has to become smaller and smaller as iterations progress, but this should not take place in a very aggressive manner, so that the algorithm is left to be active for a sufficient number of iterations to learn the solution. If the step-size tends to zero very fast, then updates are practically frozen after a few iterations, without the algorithm having acquired enough information to get close to the solution.

5.3.1 THE COMPLEX-VALUED CASE

In Section 4.4.2, we stated that a function $f : \mathbb{C}^l \mapsto \mathbb{R}$ is not differentiable with respect to its complex argument. To deal with such cases, the Wirtinger calculus was introduced. In this section, we use this mathematically convenient tool to derive the corresponding steepest descent direction.

To this end, we again employ a first order Taylor's series approximation [22]. Let

$$\boldsymbol{\theta} = \boldsymbol{\theta}_r + j\boldsymbol{\theta}_i.$$

Then, the cost function

$$J(\boldsymbol{\theta}) : \mathbb{C}^l \mapsto [0, +\infty),$$

is approximated as

$$\begin{aligned} J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) &= J(\boldsymbol{\theta}_r + \Delta\boldsymbol{\theta}_r, \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i) \\ &= J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i) + \Delta\boldsymbol{\theta}_r^T \nabla_r J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i) + \Delta\boldsymbol{\theta}_i^T \nabla_i J(\boldsymbol{\theta}_r, \boldsymbol{\theta}_i), \end{aligned} \quad (5.24)$$

where ∇_r (∇_i) denotes the gradient with respect to $\boldsymbol{\theta}_r$ ($\boldsymbol{\theta}_i$). Taking into account that

$$\Delta\boldsymbol{\theta}_r = \frac{\Delta\boldsymbol{\theta} + \Delta\boldsymbol{\theta}^*}{2}, \quad \Delta\boldsymbol{\theta}_i = \frac{\Delta\boldsymbol{\theta} - \Delta\boldsymbol{\theta}^*}{2j},$$

it is easy to show (Problem 5.3) that

$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \operatorname{Re}\{\Delta\boldsymbol{\theta}^H \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})\}, \quad (5.25)$$

where $\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})$ is the CW-derivative, defined in Section 4.4.2 as

$$\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}) = \frac{1}{2} (\nabla_r J(\boldsymbol{\theta}) + j \nabla_i J(\boldsymbol{\theta})).$$

Looking carefully at (5.25), it is straightforward to observe that the direction

$$\Delta\boldsymbol{\theta} = -\mu \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}),$$

makes the updated cost equal to

$$J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) = J(\boldsymbol{\theta}) - \mu \|\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta})\|^2,$$

which guarantees that $J(\boldsymbol{\theta} + \Delta\boldsymbol{\theta}) < J(\boldsymbol{\theta})$; it is straightforward to see, by taking into account the definition of an inner product, that the above search direction is one of the largest decrease. Thus, the counterpart of (5.3), becomes

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i \nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}^{(i-1)}) : \text{ Complex Gradient Descent Scheme.}$$

(5.26)

For the MSE cost function and for the linear estimation model, we get

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E} \left[(\mathbf{y} - \boldsymbol{\theta}^H \mathbf{x}) (\mathbf{y} - \boldsymbol{\theta}^H \mathbf{x})^* \right] \\ &= \sigma_y^2 + \boldsymbol{\theta}^H \boldsymbol{\Sigma}_x \boldsymbol{\theta} - \boldsymbol{\theta}^H \mathbf{p} - \mathbf{p}^H \boldsymbol{\theta}, \end{aligned}$$

and taking the gradient with respect to $\boldsymbol{\theta}^*$, by treating $\boldsymbol{\theta}$ as a constant (Section 4.4.2), we obtain

$$\nabla_{\boldsymbol{\theta}^*} J(\boldsymbol{\theta}) = \boldsymbol{\Sigma}_x \boldsymbol{\theta} - \mathbf{p}$$

and the respective gradient descent iteration is the same as in (5.6).

5.4 STOCHASTIC APPROXIMATION

Solving for the normal equations as well as using the gradient descent iterative scheme (for the case of the MSE), one has to have access to the second order statistics of the involved variables. However, in most of the cases, this is not known and it has to be approximated using a set of measurements. In this section, we turn our attention to algorithms that can learn the statistics iteratively via the training set. The origins of such techniques are traced back to 1951, when Robbins and Monro introduced the method of *stochastic approximation* [79] or the *Robbins-Monro algorithm*.

Let us consider the case of a function that is defined in terms of the expected value of another one, namely

$$f(\boldsymbol{\theta}) = \mathbb{E}[\phi(\boldsymbol{\theta}, \boldsymbol{\eta})], \quad \boldsymbol{\theta} \in \mathbb{R}^l,$$

where $\boldsymbol{\eta}$ is a random vector of unknown statistics. The goal is to compute a root of $f(\boldsymbol{\theta})$. If the statistics were known, the expectation could be computed, at least in principle, and one could use any root-finding algorithm to compute the roots. The problem emerges when the statistics are unknown, hence the exact form of $f(\boldsymbol{\theta})$ is not known. All one has at her/his disposal is a sequence of i.i.d. observations $\boldsymbol{\eta}_0, \boldsymbol{\eta}_1, \dots$. Robbins and Monro proved that the following algorithm,²

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \phi(\boldsymbol{\theta}_{n-1}, \boldsymbol{\eta}_n) : \quad \text{Robbins-Monro Scheme,} \quad (5.27)$$

starting from an arbitrary initial condition, $\boldsymbol{\theta}_{-1}$, converges³ to a root of $f(\boldsymbol{\theta})$, under some general conditions and provided that (Problem 5.4)

$$\sum_n \mu_n^2 < \infty, \quad \sum_n \mu_n \rightarrow \infty : \quad \text{Convergence Conditions.} \quad (5.28)$$

In other words, in the iteration (5.27), we get rid of the expectation operation and use the value of $\phi(\cdot, \cdot)$, which is computed using the current observations/measurements and the currently available estimate. That is, the algorithm learns both the statistics as well as the root; two into one! The same comments made for the convergence conditions, met in the iteration-dependent step-size case in Section 5.3, are valid here, too.

In the context of optimizing a general differentiable cost function of the form,

$$J(\boldsymbol{\theta}) = \mathbb{E}[\mathcal{L}(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})], \quad (5.29)$$

Robbins-Monro scheme can be mobilized to find a root of the respected gradient, i.e.,

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}[\nabla \mathcal{L}(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})],$$

where the expectation is w.r.t. the pair (\mathbf{y}, \mathbf{x}) . As we have seen in Chapter 3, such cost functions in the machine learning terminology are also known as the *expected risk* or the *expected loss*. Given the sequence of observations $(\mathbf{y}_n, \mathbf{x}_n)$, $n = 0, 1, \dots$, the recursion in (5.27) now becomes

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \nabla \mathcal{L}(\boldsymbol{\theta}_{n-1}, \mathbf{y}_n, \mathbf{x}_n). \quad (5.30)$$

² The original paper dealt with scalar variables only and the method was later extended to more general cases; see [96] for related discussion.

³ Convergence here is meant to be in probability; see Section 2.6.

Let us now assume, for simplicity, that the expected risk has a unique minimum, θ_* . Then, according to Robbins-Monro theorem and using an appropriate sequence μ_n , θ_n will converge to θ_* . However, although this information is important, it is not by itself enough. In practice, one has to seize iterations after a *finite* number of steps. Hence, one has to know something more concerning the rate of convergence of such a scheme. To this end, two quantities are of interest, namely the mean and the covariance matrix of the estimator at iteration n , or

$$\mathbb{E}[\theta_n], \text{Cov}(\theta_n).$$

It can be shown (see [67]), that if $\mu_n = \mathcal{O}(1/n)$ and assuming that iterations have brought the estimate close to the optimal value, then

$$\boxed{\mathbb{E}[\theta_n] = \theta_* + \frac{1}{n}c,} \quad (5.31)$$

and

$$\boxed{\text{Cov}(\theta_n) = \frac{1}{n}V + \mathcal{O}(1/n^2),} \quad (5.32)$$

where c and V are constants that depend on the form of the expected risk. The above formulae have also been derived under some further assumptions concerning the eigenvalues of the Hessian matrix of the expected risk.⁴ As we will also see, the convergence analysis of even simple algorithms is a tough task, and it is common to carry it under a number of assumptions. What is important from (5.31) and (5.32) is that both the mean as well as the standard deviations of the components follow a $\mathcal{O}(1/n)$ pattern. Furthermore, these formulae indicate that the parameter vector estimate *fluctuates* around the optimal value. This fluctuation depends on the choice of the sequence μ_n , being smaller for smaller values of the step-size sequence. However, μ_n cannot be made to decrease very fast due to the two convergence conditions, as discussed before. This is the price one pays for using the noisy version of the gradient and it is the reason that such schemes suffer from relatively slow convergence rates. However, this does not mean that such schemes are, necessarily, the poor relatives of other more “elaborate” algorithms. As we will discuss in Chapter 8, their low complexity requirements makes this algorithmic family to be the one that is selected in a number of practical applications.

Application to the MSE linear estimation

Let us apply the Robbins-Monro algorithm to solve for the optimal MSE linear estimator if the covariance matrix and the cross-correlation vector are unknown. We know that the solution corresponds to the root of the gradient of the cost function, which can be written in the form (recall the orthogonality theorem from Chapter 3),

$$\Sigma_x \theta - p = \mathbb{E}[\mathbf{x}(\mathbf{x}^T \theta - y)] = \mathbf{0}.$$

Given the training sequence of observations, (y_n, \mathbf{x}_n) , which are assumed to be i.i.d. drawn from the joint distribution of (y, \mathbf{x}) , the Robbins-Monro algorithm becomes,

$$\theta_n = \theta_{n-1} + \mu_n \mathbf{x}_n (y_n - \mathbf{x}_n^T \theta_{n-1}), \quad (5.33)$$

⁴ The proof is a bit technical and the interested reader can look at the provided reference.

which converges to the optimal MSE solution provided that the two conditions in (5.28) are satisfied. Compare (5.33) with (5.6). Taking into account the definitions, $\Sigma_x = \mathbb{E}[\mathbf{x}\mathbf{x}^T]$, $\mathbf{p} = \mathbb{E}[\mathbf{xy}]$, the former equation results from the latter one by dropping out the expectation operations and using an iteration-dependent step size. Observe that the iterations in (5.33) coincide with *time updates*; time has now explicitly entered into the scene. This prompts us to start thinking about modifying such schemes appropriately to track time-varying environments. Algorithms such as the one in (5.33), which result from the generic gradient descent formulation by replacing the expectation by the respective instantaneous observations, are also known as *stochastic gradient descent schemes*.

Remarks 5.1.

- All the algorithms to be derived next can also be applied to nonlinear estimation/filtering tasks of the form,

$$\hat{y} = \sum_{k=1}^l \theta_k \phi_k(\mathbf{x}) = \boldsymbol{\theta}^T \boldsymbol{\phi},$$

and the place of \mathbf{x} is taken by $\boldsymbol{\phi}$, where

$$\boldsymbol{\phi} = [\phi_1(\mathbf{x}), \dots, \phi_l(\mathbf{x})]^T.$$

Example 5.2. The aim of this example is to demonstrate the pair of equations (5.31) and (5.32), which characterize the convergence properties of the stochastic gradient scheme.

Data samples were first generated according to the regression model

$$y_n = \boldsymbol{\theta}^T \mathbf{x}_n + \eta_n$$

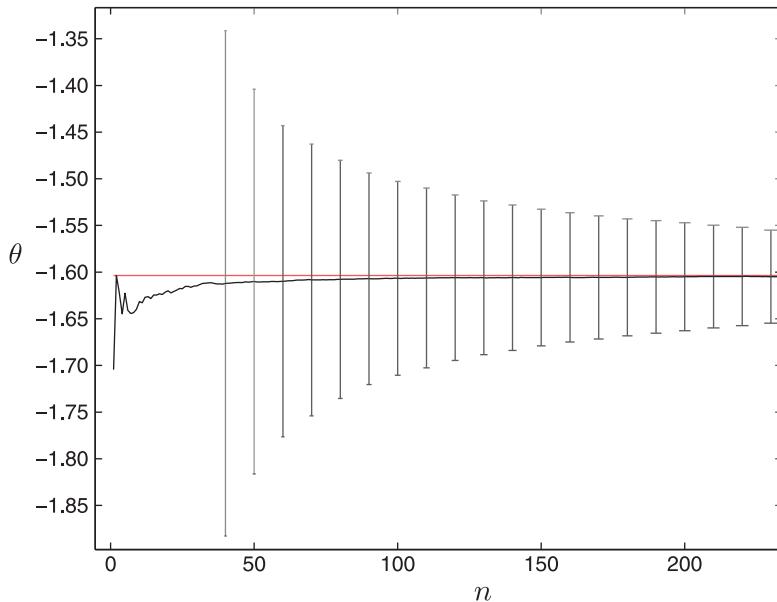
where, $\boldsymbol{\theta} \in \mathbb{R}^2$ was randomly chosen and then fixed. The elements of \mathbf{x}_n were i.i.d. generated via a normal distribution $\mathcal{N}(0, 1)$ and η_n are samples of a white noise sequence with variance equal to $\sigma^2 = 0.1$. Then, the observations (y_n, \mathbf{x}_n) were used in the recursive scheme in (5.33) to obtain an estimate of $\boldsymbol{\theta}$. The experiment was repeated 200 times and the mean and variance of the obtained estimates were computed, for each iteration step. Figure 5.12 shows the resulting curve for one of the parameters (the trend for the other one being similar). Observe that the mean values of the estimates tend to the true value, corresponding to the red line and the standard deviation keeps decreasing as n grows. The step size was chosen equal to $\mu_n = 1/n$.

5.5 THE LEAST-MEAN-SQUARES ADAPTIVE ALGORITHM

The stochastic gradient algorithm in (5.33) converges to the optimal mean-square error solution provided that μ_n satisfies the two convergence conditions. Once the algorithm has converged, it “locks” at the obtained solution. In a case where the statistics of the involved variables/processes and/or the unknown parameters starts changing, the algorithm cannot track the changes. Note that if such changes occur, the error term

$$e_n = y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n$$

will get larger values; however, because μ_n is very small, the increased value of the error will not lead to corresponding changes of the estimate at time n . This can be overcome if one sets the value of μ_n

**FIGURE 5.12**

The red line corresponds to the true value of the unknown parameter. The black curve corresponds to the average over 200 realizations of the experiment. Observe that the mean value converges to the true value. The bars correspond to the respective standard deviation, which keeps decreasing as n grows.

to a preselected *fixed* value, μ . The resulting algorithm is the celebrated least-mean-squares (LMS) algorithm [102].

Algorithm 5.1 (The LMS algorithm).

- Initialize
 - $\theta_{-1} = \mathbf{0} \in \mathbb{R}^l$; other values can also be used.
 - Select the value of μ .
- **For** $n = 0, 1, \dots$, **Do**
 - $e_n = y_n - \theta_{n-1}^T \mathbf{x}_n$
 - $\theta_n = \theta_{n-1} + \mu e_n \mathbf{x}_n$
- **End For**

In case the input is a time series,⁵ u_n , the initialization also involves the samples, $u_{-1}, \dots, u_{-l+1} = 0$, to form the input vectors, $\mathbf{u}_n, n = 0, 1, \dots, l-2$. The complexity of the algorithm amounts to $2l$

⁵ Recall our adopted notation from Chapter 2, that in this case we use \mathbf{u}_n in place of \mathbf{x}_n .

multiplications/additions (MADs) per time update. We have assumed that observations start arriving at time instant $n = 0$, to be in line with most references treating the LMS.

Let us now comment on this simple structure. Assume that the algorithm has converged close to the solution; then the error term is expected to take small values and thus the updates will remain close to the solution. If the statistics and/or the system parameters now start changing, the error values are expected to increase. Given that μ has a constant value, the algorithm has now the “agility” to update the estimates in an attempt to “push” the error to lower values. This small variation of the iterative scheme has important implications. The resulting algorithm is no more a member of the Robbins-Monro stochastic approximation family. Thus, one has to study its convergence conditions as well as its performance properties. Moreover, since the algorithm now has the potential to track changes in the values of the underlying parameters, as well as the statistics of the involved processes/variables, one has to study its performance in nonstationary environments; this is associated to what is known as the *tracking* performance of the algorithm, and it will be treated at the end of the chapter.

5.5.1 CONVERGENCE AND STEADY-STATE PERFORMANCE OF THE LMS IN STATIONARY ENVIRONMENTS

The goal of this subsection is to study the performance of the LMS in stationary environments. That is, to answer the questions: (a) does the scheme converge and under which conditions? and (b) if it converges, where does it converge? Although we introduced the scheme having in mind nonstationary environments, still we have to know how it behaves under stationarity; after all, the environment can change very slowly, and it can be considered “locally” stationary.

The convergence properties of the LMS, as well as of any other online/adaptive algorithm, are related to its *transient* characteristics; that is, the period from the initial estimate until the algorithm reaches a “steady-state” mode of operation. In general, analyzing the transient performance of an online algorithm is a formidable task indeed. This is also true even for the very simple structure of the LMS summarized in [Algorithm 5.1](#). The LMS update recursions are equivalent to a time-varying, nonlinear ([Problem 5.5](#)) and stochastic in nature estimator. Many papers, some of them of high scientific insight and mathematical skill, have been produced. However, with the exception of a few rare and special cases, the analysis involves approximations. Our goal in this book is not to treat this topic in detail. Our focus will be restricted on the most “primitive” of the techniques, which is easier for the reader to follow compared to more advanced and mathematically elegant theories; after all, even this primitive approach provides results that turn out to be in agreement to what one experiences in practice.

Convergence of the parameter error vector

Define

$$c_n := \theta_n - \theta_*,$$

where θ_* is the optimal solution resulting from the normal equations. The LMS update recursion can now be written as

$$c_n = c_{n-1} + \mu x_n (y_n - \theta_{n-1}^T x_n + \theta_*^T x_n - \theta_*^T x_n).$$

Because we are going to study the statistical properties of the obtained estimates, we have to switch our notation from that referring to observations to the one involving the respective random variables. Then we can write that,

$$\begin{aligned}\mathbf{c}_n &= \mathbf{c}_{n-1} + \mu \mathbf{x}(y - \boldsymbol{\theta}_{n-1}^T \mathbf{x} + \boldsymbol{\theta}_*^T \mathbf{x} - \boldsymbol{\theta}_*^T \mathbf{x}) \\ &= \mathbf{c}_{n-1} - \mu \mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1} + \mu \mathbf{x} \mathbf{e}_* \\ &= (I - \mu \mathbf{x} \mathbf{x}^T) \mathbf{c}_{n-1} + \mu \mathbf{x} \mathbf{e}_*,\end{aligned}\quad (5.34)$$

where

$$\mathbf{e}_* = y - \boldsymbol{\theta}_*^T \mathbf{x} \quad (5.35)$$

is the error random variable associated with the optimal $\boldsymbol{\theta}_*$. Compare (5.34) with (5.8). They look similar, yet they are very different. First, the latter of the two, involves the expected value, Σ_x , in place of the respective variables. Moreover in (5.34), there is a second term that acts as an external input to the difference stochastic equation. From (5.34), we obtain

$$\mathbb{E}[\mathbf{c}_n] = \mathbb{E}[(I - \mu \mathbf{x} \mathbf{x}^T) \mathbf{c}_{n-1}] + \mu \mathbb{E}[\mathbf{x} \mathbf{e}_*]. \quad (5.36)$$

To proceed, it is time to introduce assumptions.

Assumption 1. The involved random variables are jointly linked via the regression model,

$$y = \boldsymbol{\theta}_o^T \mathbf{x} + \eta, \quad (5.37)$$

where η is the noise variable with variance σ_η^2 and it is assumed to be independent of \mathbf{x} . Moreover, successive samples η_n , that generate the data, are assumed to be i.i.d. We have seen in Remarks 4.2 and Problem 4.4 that in this case, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$, and $\sigma_{\mathbf{e}_*}^2 = \sigma_\eta^2$. Also, due to the orthogonality condition, $\mathbb{E}[\mathbf{x} \mathbf{e}_*] = \mathbf{0}$. In addition, a stronger condition will be adopted, and \mathbf{e}_* and \mathbf{x} will be assumed to be *statistically independent*. This is justified by the fact that under the above model, $e_{*,n} = \eta_n$, and the noise sequence has been assumed to be independent of the input.

Assumption 2. (Independence Assumption) Assume that \mathbf{c}_{n-1} is statistically independent of both \mathbf{x} and \mathbf{e}_* . No doubt this is a strong assumption, but one we will adopt to simplify computations. Sometimes there is a tendency to “justify” this assumption by resorting to some special cases, which we will not do. If one is not happy with the assumption, he/she has to look for more recent methods, based on more rigorous mathematical analysis; of course, this does not mean that such methods are free of assumptions.

I. *Convergence in the mean:* Having adopted the previous assumptions, (5.36) becomes

$$\begin{aligned}\mathbb{E}[\mathbf{c}_n] &= \mathbb{E}[(I - \mu \mathbf{x} \mathbf{x}^T) \mathbf{c}_{n-1}] \\ &= (I - \mu \Sigma_x) \mathbb{E}[\mathbf{c}_{n-1}].\end{aligned}\quad (5.38)$$

Following similar arguments as in Section 5.3, we obtain

$$\mathbb{E}[\mathbf{v}_n] = (I - \mu \Lambda) \mathbb{E}[\mathbf{v}_{n-1}],$$

where, $\Sigma_x = Q \Lambda Q^T$ and $\mathbf{v}_n = Q^T \mathbf{c}_n$. The last equation leads to

$$\mathbb{E}[\boldsymbol{\theta}_n] \rightarrow \boldsymbol{\theta}_*, \quad \text{as } n \rightarrow \infty,$$

provided that

$$0 < \mu < \frac{2}{\lambda_{\max}}.$$

In other words, in a stationary environment, the LMS converges to the optimal MSE solution *in the mean*. Thus, by fixing the value of the step-size to be a constant, we lose something; the obtained estimates, even after convergence, hover around the optimal solution. The obvious task to be considered next is to study the respective covariance matrix.

II. Error vector covariance matrix: From (5.38), applying it recursively and assuming for the initial condition that $\mathbb{E}[\mathbf{c}_{-1}] = \mathbf{0}$, we have that $\mathbb{E}[\mathbf{c}_n] = \mathbf{0}$. In any case, borrowing the same arguments used in establishing the convergence in the mean, it turns out that the latter is approximately true for large enough values of n , irrespective of the initialization. Thus, from (5.34) we get,

$$\begin{aligned}\Sigma_{c,n} := \mathbb{E}[\mathbf{c}_n \mathbf{c}_n^T] &= \Sigma_{c,n-1} - \mu \mathbb{E}[\mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1} \mathbf{c}_{n-1}^T] \\ &\quad - \mu \mathbb{E}[\mathbf{c}_{n-1} \mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T] + \mu^2 \mathbb{E}[\mathbf{e}_*^2 \mathbf{x} \mathbf{x}^T] \\ &\quad + \mu^2 \mathbb{E}[\mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1} \mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T],\end{aligned}\tag{5.39}$$

where we have used the independence of \mathbf{e}_* with \mathbf{c}_{n-1} and the fact that \mathbf{e}_* is orthogonal to \mathbf{x} in order to set to zero some of the terms. Taking into consideration the adopted independence assumptions and assuming that the input vector follows a *Gaussian distribution*, (5.39) becomes

$$\begin{aligned}\Sigma_{c,n} &= \Sigma_{c,n-1} - \mu \Sigma_x \Sigma_{c,n-1} - \mu \Sigma_{c,n-1} \Sigma_x \\ &\quad + 2\mu^2 \Sigma_x \Sigma_{c,n-1} \Sigma_x + \mu^2 \Sigma_x \text{trace}\{\Sigma_x \Sigma_{c,n-1}\} \\ &\quad + \mu^2 \sigma_\eta^2 \Sigma_x,\end{aligned}\tag{5.40}$$

where the Gaussian assumption has been exploited to express the term involving fourth order moments (e.g., [74]) as

$$\mathbb{E}[\mathbf{x} \mathbf{x}^T \Sigma_{c,n-1} \mathbf{x} \mathbf{x}^T] = 2 \Sigma_x \Sigma_{c,n-1} \Sigma_x + \Sigma_x \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}.$$

Mobilizing the definition of $\mathbf{v}_n = Q^T \mathbf{c}_n$, (5.40) results in (Problem 5.6)

$$\begin{aligned}\Sigma_{v,n} &= Q^T \Sigma_{c,n} Q = \Sigma_{v,n-1} - \mu \Lambda \Sigma_{v,n-1} - \mu \Sigma_{v,n-1} \Lambda \\ &\quad + 2\mu^2 \Lambda \Sigma_{v,n-1} \Lambda + \mu^2 \Lambda \text{trace}\{\Lambda \Sigma_{v,n-1}\} + \mu^2 \sigma_\eta^2 \Lambda.\end{aligned}\tag{5.41}$$

Note that our interest lies at the diagonal elements of $\Sigma_{v,n}$, since these correspond to the variances of the respective elements of $\boldsymbol{\theta}_n - \boldsymbol{\theta}_*$, and correspondingly to $\mathbf{v}_n - \mathbf{v}_*$. Collecting all the *diagonal* elements in a vector, \mathbf{s}_n , a close inspection of the diagonal elements of $\Sigma_{v,n}$ in (5.41) can persuade the reader that the following difference equation is true,

$$s_n = (I - 2\mu \Lambda + 2\mu^2 \Lambda^2 + \mu^2 \lambda \lambda^T) s_{n-1} + \mu^2 \sigma_\eta^2 \lambda,\tag{5.42}$$

where

$$\lambda := [\lambda_1, \lambda_2, \dots, \lambda_l]^T.$$

It is well known from the linear system theory that, the difference equation in (5.42) is stable if the eigenvalues of the matrix,

$$\begin{aligned} A &:= I - 2\mu\Lambda + 2\mu^2\Lambda^2 + \mu^2\lambda\lambda^T \\ &= (I - \mu\Lambda)^2 + \mu^2\Lambda^2 + \mu^2\lambda\lambda^T, \end{aligned} \quad (5.43)$$

have magnitude less than one. This can be guaranteed if the step size μ is chosen such as (Problem 5.7)

$$0 < \mu < \frac{2}{\sum_{i=1}^l \lambda_i},$$

or

$$\boxed{\mu < \frac{2}{\text{trace}\{\Sigma_x\}}}. \quad (5.44)$$

The last condition guarantees that the variances remain *bounded*. Recall the number of assumptions made. Thus, to be on the safe side, μ must be selected so that it is not close to this upper bound.

III. Excess Mean-Square Error: We know that the minimum MSE is achieved at θ_* . Any other weight vector results in higher values of the mean-square error. We have already said that in the steady-state, the estimates obtained via the LMS fluctuate randomly around θ_* ; thus, the mean-square error will be larger than the minimum J_{\min} . This “extra” error power, denoted as J_{exc} , is known as the *excess* mean-square error. Also, the ratio

$$\mathcal{M} := \frac{J_{\text{exc}}}{J_{\min}},$$

is known as the *misadjustment*. No doubt, we should seek for the relationship of \mathcal{M} with μ and in practice we would like to adjust μ accordingly, in order to get a value of \mathcal{M} as small as possible. Unfortunately, we will soon see that there is a trade-off in achieving that. Making \mathcal{M} small, the convergence speed becomes slower and vice versa; there is no free lunch!

By the respective definitions, we have⁶

$$\begin{aligned} \mathbf{e}_n &= \mathbf{y}_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x} \\ &= \mathbf{e}_{*,n} - \mathbf{c}_{n-1}^T \mathbf{x}, \end{aligned}$$

or

$$\mathbf{e}_n^2 = \mathbf{e}_{*,n}^2 + \mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1} - 2\mathbf{e}_{*,n} \mathbf{c}_{n-1}^T \mathbf{x}. \quad (5.45)$$

Taking the expectation on both sides and exploiting the assumed independence between \mathbf{c}_{n-1} and \mathbf{x} and $\mathbf{e}_{*,n}$, as well as the orthogonality between $\mathbf{e}_{*,n}$ and \mathbf{x} , we get

$$\begin{aligned} J_n &:= \mathbb{E}[\mathbf{e}_n^2] = J_{\min} + \mathbb{E}[\mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1}] \\ &= J_{\min} + \mathbb{E}[\text{trace}\{\mathbf{c}_{n-1}^T \mathbf{x} \mathbf{x}^T \mathbf{c}_{n-1}\}] \\ &= J_{\min} + \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}, \end{aligned} \quad (5.46)$$

⁶ The time index n is explicitly used for \mathbf{e} , \mathbf{e}_* , \mathbf{y} , since the formula to be derived is also valid for time-varying environments, and it is going to be used later on for the time-varying statistics case.

where the property $\text{trace}\{AB\} = \text{trace}\{BA\}$ has been used. Thus, we can finally write that,

$$J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\} : \text{Excess MSE at Time Instant } n. \quad (5.47)$$

Let us now elaborate on it a bit more. Taking into account that $QQ^T = I$, we get

$$\begin{aligned} J_{\text{exc},n} &= \text{trace}\{QQ^T \Sigma_x QQ^T \Sigma_{c,n-1} QQ^T\} \\ &= \text{trace}\{Q\Lambda \Sigma_{v,n-1} Q^T\} = \text{trace}\{\Lambda \Sigma_{v,n-1}\} \\ &= \sum_{i=1}^l \lambda_i [\Sigma_{v,n-1}]_{ii} = \lambda^T s_{n-1}, \end{aligned} \quad (5.48)$$

where s_n is the vector of the diagonal elements of $\Sigma_{v,n}$ and obeys the difference equation in (5.42). Assuming that μ has been chosen so that convergence is guaranteed, then for large values of n the *steady-state* has been reached. In a more formal way, an online algorithm has reached the steady state if

$$\mathbb{E}[\theta_n] = \mathbb{E}[\theta_{n-1}] = \text{Constant}, \quad (5.49)$$

$$\Sigma_{\theta,n} = \Sigma_{\theta,n-1} = \text{Constant}. \quad (5.50)$$

Thus, in steady-state, we assume in Eq. (5.42) that $s_n = s_{n-1}$; if this is exploited in (5.48) leads to (Problem 5.10),

$$J_{\text{exc},\infty} := \lim_{n \rightarrow \infty} J_{\text{exc},n} \simeq \frac{\mu \sigma_\eta^2 \text{trace}\{\Sigma_x\}}{2 - \mu \text{trace}\{\Sigma_x\}}, \quad (5.51)$$

and for the misadjustment (since under our assumptions $J_{\min} = \sigma_\eta^2$),

$$\mathcal{M} \simeq \frac{\mu \text{trace}\{\Sigma_x\}}{2 - \mu \text{trace}\{\Sigma_x\}},$$

which, for small values of μ , leads to

$$J_{\text{exc},\infty} \simeq \frac{1}{2} \mu \sigma_\eta^2 \text{trace}\{\Sigma_x\} : \text{Excess MSE}, \quad (5.52)$$

and

$$\mathcal{M} \simeq \frac{1}{2} \mu \text{trace}\{\Sigma_x\} : \text{Misadjustment}. \quad (5.53)$$

That is, the smaller the value of μ the smaller the excess mean-square error.

IV. *Time constant*. Note that the transient behavior of the LMS is described by the difference equation in (5.42) and its convergence rate (the speed with which it forgets the initial conditions till it settles to its steady-state operation) depends on the eigenvalues of A in (5.43). To simplify the formulas, assume μ to be small enough so that A is approximated as $(I - \mu\Lambda)^2$. Following similar arguments as those used for the gradient descent in Section 5.3, we can write that

$$\tau_j^{LMS} \simeq \frac{1}{2\mu\lambda_j}.$$

That is, the time constant for each one of the modes is inversely proportional to μ . Hence, the slower the rate of convergence (small values of μ) the lower the misadjustment and vice versa. Viewing it differently, the more time the algorithm spends on learning, prior to reaching the steady-state, the smaller is its deviation from the optimal.

5.5.2 CUMULATIVE LOSS BOUNDS

In the previously reported analysis method for the LMS performance, there is an underlying assumption that the training samples are generated by a linear model. The focus of the analysis was to investigate how well our algorithm estimates the unknown model, once steady-state has been reached. This path of analysis is very popular and well suited for a number of tasks such as system identification.

An alternative route for studying the performance of an algorithm, shedding light from a different angle, is via the so-called *cumulative loss*. Recall that the main goal in machine learning is *prediction*; hence, measuring the prediction accuracy of an algorithm, given a set of observations, becomes the main goal. However, this performance index should be measured against the generalization ability of the algorithm, as pointed out in [Chapter 3](#). In practice, this can be done in different ways, such as via the Leave-One-Out method, [Section 3.13](#).

For the case of the squared error loss function, the cumulative loss over N observation samples is defined as

$$\mathcal{L}_{\text{cum}} = \sum_{n=0}^{N-1} (y_n - \hat{y}_n)^2 = \sum_{n=0}^{N-1} (y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n)^2 : \quad \text{Cumulative Loss.} \quad (5.54)$$

Note that $\boldsymbol{\theta}_{n-1}$ has been estimated based on observations up to and including the time instant $n - 1$. So, the training pair (y_n, \mathbf{x}_n) , can be considered as a test sample, not involved in the training, for measuring the error. It must be pointed out, however, that the cumulative loss is *not* a direct measure of the generalization performance associated with the finally obtained parameter vector. Such a measure should involve, $\boldsymbol{\theta}_{N-1}$, tested against a number of test samples.

The goal of the family of methods, which build around the cumulative loss, is to derive corresponding upper bounds. Our aim here is to outline the essence behind such approaches, without resorting to proofs; these comprise a series of bounds and can become a bit technical. The interested reader can consult the related references. We will return to the cumulative loss and related bounds in the context of the so-called *regret analysis* in [Chapter 8](#).

In the context of the LMS, the following theorem has been proved in [\[21\]](#).

Theorem 5.1. *Let $C = \max_n \|\mathbf{x}_n\|$, $\mu = \beta/C^2$, $0 < \beta < 2$. Then, the set of predictions, $\hat{y}_0, \dots, \hat{y}_{N-1}$, generated by the Algorithm 5.1, satisfies the following bound*

$$\sum_{n=0}^{N-1} (y_n - \hat{y}_n)^2 \leq \inf_{\boldsymbol{\theta}} \left\{ \frac{C^2 \|\boldsymbol{\theta}\|^2}{2\beta(1-\beta)c} + \frac{\mathcal{L}(\boldsymbol{\theta}, S)}{(2-\beta)^2 c(1-c)} \right\}, \quad (5.55)$$

where, $0 < c < 1$ and

$$\mathcal{L}(\boldsymbol{\theta}, S) = \sum_{n=0}^{N-1} (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2, \quad (5.56)$$

where $S = \{(y_n, \mathbf{x}_n), n = 0, 1, \dots, N - 1\}$.

One can then tune β optimally to minimize the upper bound. Note that this is a *worst-case* scenario. The tuning is achieved by restricting the set of linear functions, so that $\|\theta\| \leq \Theta$. Let

$$L_\Theta(S) = \min_{\|\theta\| \leq \Theta} \mathcal{L}(\theta, S), \quad (5.57)$$

and also assume that there is an upper bound L , such as

$$|L_\Theta(S)| \leq L. \quad (5.58)$$

Then, it can be shown [21] that

$$\sum_{n=0}^{N-1} (y_n - \hat{y}_n)^2 \leq L_\Theta(S) + 2\Theta C \sqrt{L} + (\Theta C)^2. \quad (5.59)$$

Note that the previous analysis has been carried out without invoking any probabilistic arguments. Alternative bounds are derived by mobilizing different assumptions [21]. Bounds of this kind, involving similar assumptions, are frequently encountered for the analysis of various algorithms. We will meet such examples later in this book.

Remarks 5.2.

- The analysis method presented in this section, concerning the transient and steady-state performance of the LMS, can be considered as the most primitive and goes back to the early work of Widrow and Hopf, [102]. Another popular path for analyzing stochastic algorithms in an averaging sense is the so-called *averaging method* that operates under the assumption of small values of the step-size, μ [56]. Another approach that builds around the assumption of small step sizes, $\mu \simeq 0$, is the so-called ordinary-differential-equation approach (ODE) [57]. The difference update equation is “transformed” to a differential equation, which paves the way for using arguments from the Lyapunov stability theory. An alternative elegant theoretical tool, which can be used as a vehicle for analyzing the transient, the steady-state as well as the tracking performance of adaptive schemes is the so-called *energy conservation* method, developed by Sayed and Rupp [82] and later on extended in Refs. [3, 105]. More on the performance analysis of the LMS as well as of other online schemes can be found in more specialized books and papers [4, 13, 47, 62, 83, 91, 92, 96, 103].
- *H^∞ optimality of the LMS.* It may come as a surprise that the LMS algorithm, a very simple structure that was obtained via an approximation, has survived the time and is one of the most popular and widely used schemes in practical applications. The reason is that, besides its low complexity, it enjoys the luxury of robustness. An alternative optimization flavor of the LMS algorithm has been given via the theory of H^∞ optimization for estimation.

Assume that our data obey the regression model of (5.37), where now no assumption is made on the nature of η . Given the sequence of output observation samples, y_0, y_1, \dots, y_{N-1} , the goal is to obtain estimates, $\hat{s}_{n|n-1}$, of s_n , generated as $s_n = \theta^T x_n$, based on the training set up to and including time $n-1$ (causality), such that

$$\frac{\sum_{n=0}^{N-1} |\hat{s}_{n|n-1} - s_n|^2}{\mu^{-1} \|\theta\|^2 + \sum_{n=0}^{N-1} |\eta_n|^2} < \gamma^2. \quad (5.60)$$

The numerator is the total squared estimation error. The denominator involves two terms. One is the noise/disturbance energy, and the other is the norm of the unknown parameter vector.

Assuming that one starts iterations from $\boldsymbol{\theta}_{-1} = \mathbf{0}$, this term measures the energy of the disturbance from the initial guess. It turns out that the LMS scheme is the one that minimizes the following cost

$$\gamma_{\text{opt}}^2 = \inf_{\{\hat{s}_{n|n-1}\}} \sup_{\{\boldsymbol{\theta}, \eta_n\}} \left(\frac{\sum_{n=0}^{\infty} |\hat{s}_{n|n-1} - s_n|^2}{\mu^{-1} \|\boldsymbol{\theta}\|^2 + \sum_{n=0}^{\infty} |\eta_n|^2} \right).$$

Moreover, it turns out that the optimum corresponds to $\gamma_{\text{opt}}^2 = 1$ [46, 83]. Note that, basically, the LMS optimizes a worst-case scenario. It makes the estimation error minimum under the worst (maximum) disturbance circumstances. This type of optimality explains the robust performance of the LMS under “nonideal” environments, which are often met in practice, where a number of modeling assumptions are not valid. Such deviations from the model can be accommodated in η_n , which then “loses” its i.i.d., white, Gaussian, or any other mathematically attractive property. Finally, it is interesting to point out the similarity of the bound in (5.60) with that in (5.55). Indeed, in the former make the following substitutions,

$$\hat{s}_{n|n-1} = \hat{y}_n, \quad s_n = y_n, \quad \eta_n = y_n - \boldsymbol{\theta}^T \mathbf{x}_n.$$

Ignoring the values of the constants, the involved quantities are the same. After all, H^∞ is about maximizing the worst-case scenario [55].

5.6 THE AFFINE PROJECTION ALGORITHM

As it will soon be verified in the simulations section, a major drawback of the basic LMS scheme is its fairly slow convergence speed. In an attempt to improve upon it, a number of variants have been proposed over the years. The *affine projection algorithm* (APA) belongs to the so-called *data-reusing* family, where, at each time instant, past data are reused. Such a rationale helps the algorithm to “learn faster” and improve the convergence speed. However, besides the increased complexity, the faster convergence speed is achieved at the *expense of an increased misadjustment level*.

The APA was proposed originally in Ref. [48] and later on in Ref. [72]. Let the currently available estimate be $\boldsymbol{\theta}_{n-1}$. According to APA, the updated estimate, $\boldsymbol{\theta}$, must satisfy the following constraints:

$$\mathbf{x}_{n-i}^T \boldsymbol{\theta} = y_{n-i}, \quad i = 0, 1, \dots, q-1.$$

In other words, we *force* the parameter vector, $\boldsymbol{\theta}$, to provide at its output the desired response samples, for the q most recent time instants, where q is a user-defined parameter. At the same time, APA requires $\boldsymbol{\theta}$ to be as close as possible, in the Euclidean norm sense, to the current estimate, $\boldsymbol{\theta}_{n-1}$. Thus, APA, at each time instant, solves the following constrained optimization task,

$$\begin{aligned} \boldsymbol{\theta}_n &= \arg \min_{\boldsymbol{\theta}} \|\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}\|^2 \\ \text{s.t.} \quad \mathbf{x}_{n-i}^T \boldsymbol{\theta} &= y_{n-i}, \quad i = 0, 1, \dots, q-1. \end{aligned} \tag{5.61}$$

If one defines the $q \times l$ matrix

$$X_n = \begin{bmatrix} \mathbf{x}_n^T \\ \vdots \\ \mathbf{x}_{n-q+1}^T \end{bmatrix},$$

then the set of constraints can be compactly written as,

$$X_n \boldsymbol{\theta} = \mathbf{y}_n,$$

where

$$\mathbf{y}_n = [y_n \dots y_{n-q+1}]^T.$$

Using Lagrange multipliers in (5.61) results in (Problem 5.11),

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + X_n^T (X_n X_n^T)^{-1} \mathbf{e}_n, \quad (5.62)$$

$$\mathbf{e}_n = \mathbf{y}_n - X_n \boldsymbol{\theta}_{n-1}, \quad (5.63)$$

provided that $X_n X_n^T$ is invertible. The resulting scheme is summarized in Algorithm 5.2.

Algorithm 5.2 (The affine projection algorithm).

- Initialize
 - $\mathbf{x}_{-1} = \dots = \mathbf{x}_{-q+1} = \mathbf{0}$, $y_{-1} \dots y_{-q+1} = 0$
 - $\boldsymbol{\theta}_{-1} = \mathbf{0} \in \mathbb{R}^l$ (or any other value).
 - Choose $0 < \mu < 2$ and δ to be small.
- **For** $n = 0, 1, \dots$, **Do**
 - $\mathbf{e}_n = \mathbf{y}_n - X_n \boldsymbol{\theta}_{n-1}$
 - $\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu X_n^T (\delta I + X_n X_n^T)^{-1} \mathbf{e}_n$
- **End For**

When the input is a time series, the corresponding input vector, denoted as \mathbf{u}_n , is initialized by setting to zero all required samples with negative time index, u_{-1}, u_{-2}, \dots . Note that in the algorithm, a parameter, δ , of a small value has also been used to prevent numerical problems in the associated matrix inversion. Also, a step-size μ has been introduced, that controls the size of the update and whose presence will be justified soon. The complexity of APA is increased, compared to that of the LMS, due to the involved matrix inversion and matrix operations, requiring $O(q^3)$ MADs. Fast versions of the APA, which exploit the special structure of $X_n X_n^T$, for the case where the involved input-output variables are realizations of stochastic processes, have been developed, see [42, 43].

The convergence analysis of the APA is more involved than that of the LMS. It turns out that provided that $0 < \mu < 2$, stability of the algorithm is guaranteed. The misadjustment is approximately given by [2, 34, 83]

$$\mathcal{M} \simeq \frac{\mu q \sigma_\eta^2}{2 - \mu} \mathbb{E} \left[\frac{1}{\|\mathbf{x}_n\|^2} \right] \text{trace}\{\Sigma_x\} : \quad \text{Misadjustment for the APA.}$$

In words, the misadjustment increases as the parameter q increases; that is, as the number of the reused past data samples increases.

Geometric interpretation of APA

Let us look at the optimization task in (5.61), associated with APA. Each one of the q constraints defines a hyperplane in the l -dimensional space. Hence, since $\boldsymbol{\theta}_n$ is constrained to lie on all these hyperplanes, it will lie in their *intersection*. Provided that \mathbf{x}_{n-i} , $i = 0, \dots, q - 1$, are linearly independent, these

hyperplanes share a nonempty intersection, which is an *affine set* of dimension $l - q$. An affine set is the translation of a linear subspace (i.e., a plane crossing the origin) by a constant vector; that is, it defines a plane in a general position. Thus, θ_n can lie anywhere in this affine set. From the infinite number of points lying in this set, APA selects the one that lies closest, in the Euclidean distance sense, to θ_{n-1} . In other words, θ_n is the *projection* of θ_{n-1} on the affine set defined by the intersection of the q hyperplanes. Recall from geometry that, the projection $P_H(\mathbf{a})$ of a point \mathbf{a} on a linear subspace/affine set, H , is the point in H whose distance from \mathbf{a} is minimum. [Figure 5.13](#) illustrates the geometry for the case of $q = 2$; this special case of APA is also known as the binormalized data reusing LMS [7].

In the ideal noiseless case, the unknown parameter vector would lie in the intersection of all the hyperplanes defined by (y_n, \mathbf{x}_n) , $n = 0, 1, \dots, q - 1$, and this is the information that APA tries to exploit to speed up convergence. However, this is also its drawback. In any practical system, noise is present;

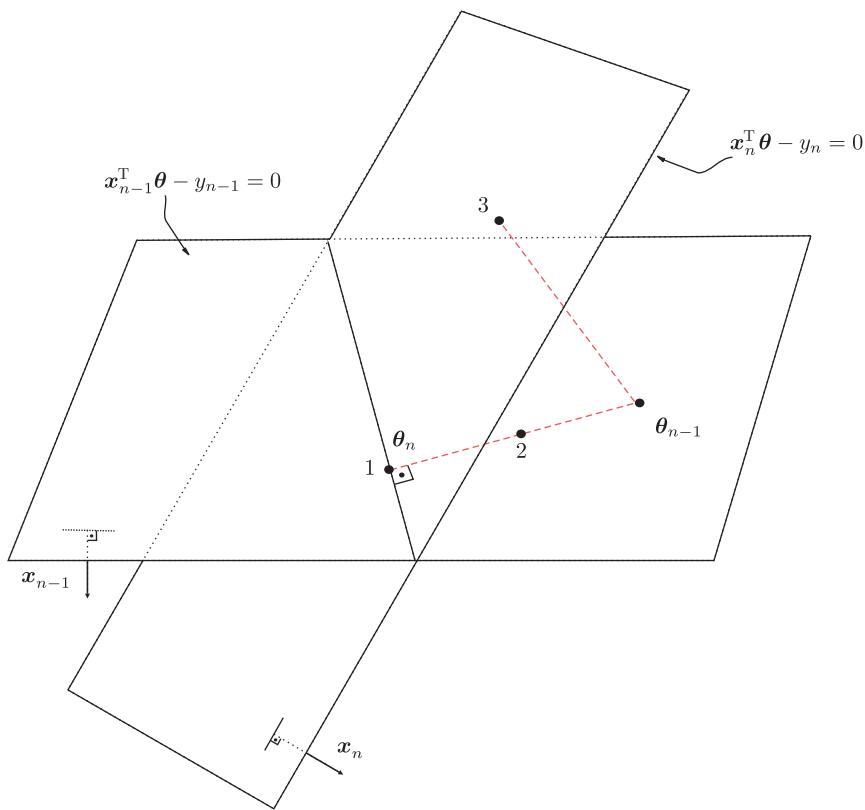


FIGURE 5.13

The geometry associated with the APA algorithm, for $q = 2$ and $l = 3$. The intersection of the two hyperplanes is a straight line (affine set of dimension $3 - 2 = 1$). θ_n is the projection of θ_{n-1} on this line (point 1) for $\mu = 1$ and $\delta = 0$. Point 2 corresponds to the case $\mu < 1$. Point 3 is the projection of θ_n on the hyperplane defined by (y_n, \mathbf{x}_n) . This is the case for $q = 1$. The latter case corresponds to the normalized LMS of [Section 5.6.1](#).

thus forcing the updates to lie in the intersection of these hyperplanes is not necessarily good, since their position in space is also determined by the noise. As a matter of fact, the reason that μ is introduced is to account for such cases. An alternative technique, which exploits projections, and at the same time replaces hyperplanes by hyperslabs (whose width depends on the noise variance) to account for the noise, will be treated in [Chapter 8](#). In addition there, no matrix inversion will be required.

Orthogonal projections

Projections and projection matrices/operators play a crucial part in machine learning, signal processing and optimization in general; after all, a projection corresponds to a minimization task, when the loss is interpreted as a “distance.” Given an $l \times k$, $k < l$ matrix, A , with column vectors, \mathbf{a}_i , $i = 1, \dots, k$, and an l -dimensional vector \mathbf{x} , then the orthogonal projection of \mathbf{x} on the subspace spanned by the columns of A (assumed to be linearly independent) is given by

$$P_{\{\mathbf{a}_i\}}(\mathbf{x}) = A(A^T A)^{-1} A^T \mathbf{x}, \quad (5.64)$$

where in complex spaces the transpose operation is replaced by the Hermitian one. One can easily check that $P_{\{\mathbf{a}_i\}}^\perp(\mathbf{x}) := (I - P_{\{\mathbf{a}_i\}})\mathbf{x}$ is orthogonal to $P_{\{\mathbf{a}_i\}}(\mathbf{x})$ and

$$\mathbf{x} = P_{\{\mathbf{a}_i\}}(\mathbf{x}) + P_{\{\mathbf{a}_i\}}^\perp(\mathbf{x}).$$

When A has orthonormal columns, we obtain our familiar from geometry expansion

$$P_{\{\mathbf{a}_i\}}(\mathbf{x}) = AA^T \mathbf{x} = \sum_{i=1}^k (\mathbf{a}_i^T \mathbf{x}) \mathbf{a}_i.$$

Thus, the factor $(A^T A)^{-1}$, for the general case, accounts for the lack of orthonormality of the columns. The matrix $A(A^T A)^{-1} A^T$ is known as the respective *projection matrix* and $I - A(A^T A)^{-1} A^T$ as the projection matrix on the respective *orthogonal complement* space.

The simplest case occurs when $k = 1$; then the projection of \mathbf{x} onto \mathbf{a}_1 is equal to

$$P_{\{\mathbf{a}_1\}}(\mathbf{x}) = \frac{\mathbf{a}_1 \mathbf{a}_1^T}{\|\mathbf{a}_1\|^2} \mathbf{x},$$

and the corresponding projection matrices are given by,

$$P_{\{\mathbf{a}_1\}} = \frac{\mathbf{a}_1 \mathbf{a}_1^T}{\|\mathbf{a}_1\|^2}, \quad P_{\{\mathbf{a}_1\}}^\perp = I - \frac{\mathbf{a}_1 \mathbf{a}_1^T}{\|\mathbf{a}_1\|^2}.$$

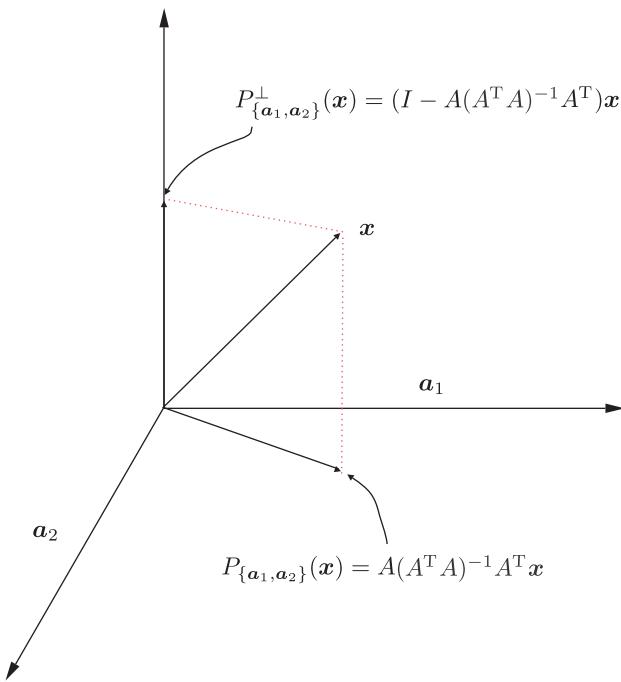
[Figure 5.14](#) illustrates the geometry.

Let us apply the previously reported linear algebra results in the case of the APA algorithm of [\(5.62\)](#) and [\(5.63\)](#), and rewrite them as

$$\begin{aligned} \boldsymbol{\theta}_n &= \left(I - X_n^T (X_n X_n^T)^{-1} X_n \right) \boldsymbol{\theta}_{n-1} \\ &\quad + X_n^T (X_n X_n^T)^{-1} \mathbf{y}_n. \end{aligned}$$

The first term on the right-hand side is the projection $P_{\{\mathbf{x}_n, \dots, \mathbf{x}_{n-q+1}\}}^\perp(\boldsymbol{\theta}_{n-1})$. This is the most natural. By the definition of the respective affine set, as the intersection of the hyperplanes

$$\mathbf{x}_{n-i}^T \boldsymbol{\theta} - y_{n-i} = 0, \quad i = 0, \dots, q-1,$$

**FIGURE 5.14**

Geometry indicating the orthogonal projection operation on the subspace spanned by $\mathbf{a}_1, \mathbf{a}_2$, using the projection matrix. Note that $A = [\mathbf{a}_1, \mathbf{a}_2]$.

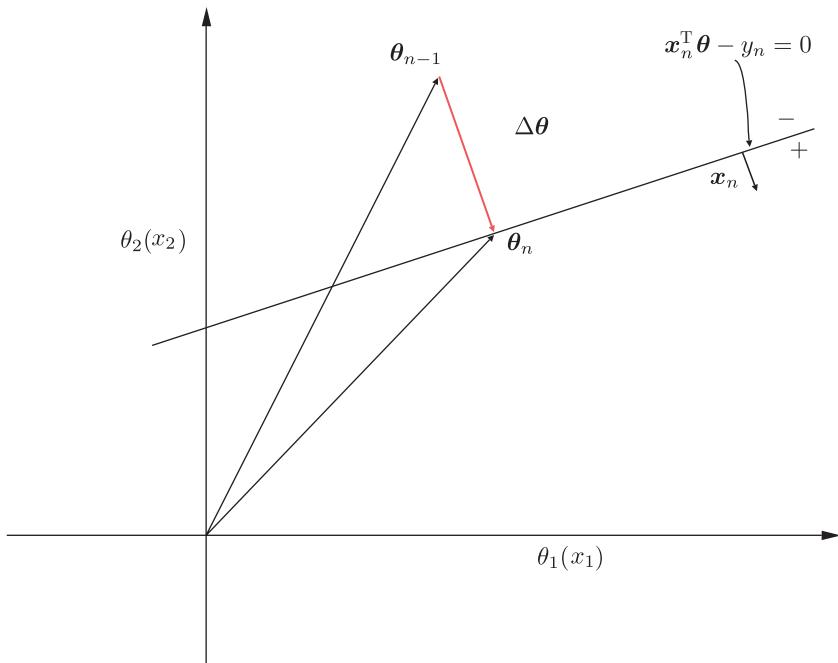
each vector \mathbf{x}_{n-i} is *orthogonal* to the respective hyperplane (Figure 5.13) (Problem 5.12). Hence, projecting $\boldsymbol{\theta}_{n-1}$ on the intersection of all these hyperplanes is equivalent to projecting on an affine set, which is *orthogonal* to all $\mathbf{x}_n, \dots, \mathbf{x}_{n-q+1}$. Note that the matrix $X_n^T(X_n X_n^T)^{-1}X_n$ is the projection matrix that projects on the subspace spanned by $\mathbf{x}_n, \dots, \mathbf{x}_{n-q+1}$. The second term accounts for the fact that the affine set, on which we project, does not include the origin, but it is translated to another point in the space, whose direction is determined by \mathbf{y}_n . Figure 5.15 illustrates the case for $l = 2$ and $q = 1$. Because $\boldsymbol{\theta}_{n-1}$ does not lie on the line (plane) $\mathbf{x}_n^T \boldsymbol{\theta} - y_n = 0$, whose direction is defined by \mathbf{x}_n , we know from geometry (and it is easily checked out) that its distance from this line is $s = \frac{|\mathbf{x}_n^T \boldsymbol{\theta}_{n-1} - y_n|}{\|\mathbf{x}_n\|}$. Also, $\boldsymbol{\theta}_{n-1}$, lies on the negative side of the straight line, so that $\mathbf{x}_n^T \boldsymbol{\theta}_{n-1} - y_n < 0$. Hence, taking into account the directions of the involved vectors, it turns out that

$$\Delta \boldsymbol{\theta} = \frac{y_n - \mathbf{x}_n^T \boldsymbol{\theta}_{n-1}}{\|\mathbf{x}_n\|} \frac{\mathbf{x}_n}{\|\mathbf{x}_n\|}.$$

Thus, for this specific case, the correction

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \Delta \boldsymbol{\theta}$$

coincides with the recursion of the APA.

**FIGURE 5.15**

$\Delta\theta$ is equal to the (signed) distance of θ_{n-1} from the plane times the unit vector $\frac{x_n}{\|x_n\|}$.

5.6.1 THE NORMALIZED LMS

The normalized LMS is a special case of the APA and it corresponds to $q = 1$, see [Figure 5.13](#). We treat it separately due to its popularity and it is summarized in [Algorithm 5.3](#).

Algorithm 5.3 (The normalized LMS).

- Initialization
 - $\theta_{-1} = \mathbf{0} \in \mathbb{R}^l$, or any other value.
 - Choose $0 < \mu < 2$, and δ a small value.
- **For** $n = 0, 1, 2, \dots$, **Do**
 - $e_n = y_n - \theta_{n-1}^T x_n$
 - $\theta_n = \theta_{n-1} + \frac{\mu}{\delta + x_n^T x_n} x_n e_n$
- **End For**

The complexity of the normalized LMS, is $3l$ MADs. Stability of the normalized LMS is guaranteed if

$$0 < \mu < 2.$$

One can look at the normalized LMS as an LMS whose step size is left to vary with the iterations, as represented by

$$\mu_n = \frac{\mu}{\delta + \mathbf{x}_n^T \mathbf{x}_n},$$

which turns out to have a beneficial effect on the convergence speed, compared to the LMS. More on the performance analysis of the normalized LMS, the interested reader can obtain from Refs. [15, 83, 90].

Remarks 5.3.

- To deal with sparse models, the so-called *proportionate* NLMS and related versions of the other algorithms have been derived [11, 35]. The idea is to use a separate step size for each one of the parameters. This gives the freedom to the coefficients, which correspond to small (or zero) values of the model, to adapt at a different rate than the rest, and this has a significant effect on the convergence performance of the algorithm. Such schemes can be considered as the ancestors of the more theoretically elegant sparsity-promoting online algorithms to be treated in Chapter 10.
- Another trend that has received attention more recently is to appropriately (convexly) combine the outputs of two (or more) learning structures. This has the effect of decreasing the sensitivity of the learning algorithms to choices of parameters such as the step-size, or to the dimensionality of the problem (size of the filter). The two (or more) algorithms run independently and the mixing parameters of the outputs are learned during the training. In general, this approach turns out to be more robust in the choice of the involved user-defined parameters [8, 81].
- Sometimes in bibliography, the user-defined parameter, δ , in the NLMS and the APA algorithm is suggested to be given a very small positive value. Often, the explanation for this option is that the use of δ is to avoid division by zero. However, in practice, there are cases, such as the echo cancellation task, where δ need to be set to quite large values (even larger than 1) to attain good performance. It is fairly recently that the importance of δ was emphasized and a formula for its proper tuning was proposed both for the case of NLMS and APA and their proportionate counterparts [12, 73]. There, it is indicated that without the proper setup of this parameter, the performance of these algorithms may be significantly affected, and they may not even converge.

5.7 THE COMPLEX-VALUED CASE

In Section 4.4, when

$$\mathbf{y}_n \in \mathbb{C} \quad \text{and} \quad \mathbf{x}_n \in \mathbb{C}^l,$$

the *widely linear* formulation of the estimation task was introduced to deal with complex-valued data that do not obey the circularity conditions. The output of a widely linear estimator is given by,

$$\hat{\mathbf{y}}_n = \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n,$$

where

$$\boldsymbol{\varphi} := \begin{bmatrix} \boldsymbol{\theta} \\ \mathbf{v} \end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{x}}_n = \begin{bmatrix} \mathbf{x}_n \\ \mathbf{x}_n^* \end{bmatrix},$$

with $\boldsymbol{\theta}, \mathbf{v} \in \mathbb{C}^l$. The MSE cost function is

$$J(\boldsymbol{\varphi}) = \mathbb{E} \left[|\mathbf{y}_n - \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n|^2 \right],$$

and following standard arguments analogous to [Section 5.3.1](#), the minimum w.r.t. φ is given by the root of

$$\Sigma_{\tilde{x}} \varphi - \begin{bmatrix} p \\ q^* \end{bmatrix} = \mathbf{0} \text{ or } \mathbb{E} \left[\tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^H \right] \varphi = \mathbb{E} \left[\tilde{\mathbf{x}}_n y_n^* \right].$$

The widely linear LMS

Employing the Robbins-Monro scheme and fixing the value of μ_n to be a constant, we obtain

$$\varphi_n = \varphi_{n-1} + \mu \tilde{\mathbf{x}}_n e_n^*,$$

$$e_n = y_n - \varphi_{n-1}^H \tilde{\mathbf{x}}_n.$$

Breaking φ_n and $\tilde{\mathbf{x}}_n$ into their components, the widely linear LMS results.

Algorithm 5.4 (The widely linear LMS).

- Initialize
 - $\theta_{-1} = \mathbf{0}$, $v_{-1} = \mathbf{0}$
 - Choose μ .
- **For** $n = 0, 1, \dots$, **Do**

$$e_n = y_n - \theta_{n-1}^H \mathbf{x}_n - v_{n-1}^H \mathbf{x}_n^* \quad (5.65)$$

$$\theta_n = \theta_{n-1} + \mu \mathbf{x}_n e_n^* \quad (5.66)$$

$$v_n = v_{n-1} + \mu \mathbf{x}_n^* e_n^* \quad (5.67)$$

- **End For**

Note that whatever has been said for the stability conditions concerning the LMS is applied here as well, if Σ_x is replaced by $\Sigma_{\tilde{x}}$. For circularly symmetric variables, we set $v_n = \mathbf{0}$ and the *complex linear LMS* results.

The widely linear APA

Let φ_n and $\tilde{\mathbf{x}}_n$ be defined as before. The widely linear APA results and it is given in [Algorithm 5.5 \(Problem 5.13\)](#).

Algorithm 5.5 (The widely linear APA).

- Initialize
 - $\varphi_{-1} = \mathbf{0}$
 - Choose μ .
- **For** $n = 0, 1, 2, \dots$, **Do**

$$\begin{aligned} e_n^* &= y_n^* - \tilde{X}_n \varphi_{n-1} \\ \varphi_n &= \varphi_{n-1} + \mu \tilde{X}_n^H (\delta I + \tilde{X}_n \tilde{X}_n^H)^{-1} e_n^* \end{aligned}$$

- **End For**

Note that,

$$\tilde{X}_n = \begin{bmatrix} \tilde{\mathbf{x}}_n^H \\ \vdots \\ \tilde{\mathbf{x}}_{n-q+1}^H \end{bmatrix} = \begin{bmatrix} \mathbf{x}_n^H, \mathbf{x}_n^T \\ \vdots \\ \mathbf{x}_{n-q+1}^H, \mathbf{x}_{n-q+1}^T \end{bmatrix},$$

and $\varphi_n \in \mathbb{C}^{2l}$. For circular variables/processes, the complex linear APA results by setting

$$\tilde{X}_n = X_n = \begin{bmatrix} x_n^H \\ \vdots \\ x_{n-q+1}^H \end{bmatrix}$$

and

$$\varphi_n = \theta_n \in \mathbb{C}^l.$$

5.8 RELATIVES OF THE LMS

In addition to the three basic stochastic gradient descent schemes that were previously reviewed, a number of variants have been proposed over the years, in an effort either to improve performance or to reduce complexity. Some notable examples are described below.

The sign-error LMS

The update recursion for this algorithm becomes (e.g., [17, 30, 63])

$$\theta_n = \theta_{n-1} + \mu \text{csgn}[e_n^*] x_n,$$

where the complex sign of a complex number, $z = x + jy$, is defined as

$$\text{csgn}(z) = \text{sgn}(x) + j \text{sgn}(y).$$

If in addition, μ is chosen to be a power of two, then the recursion becomes *multiplication free*, and l multiplications are only needed for the computation of the error. It turns out that the algorithm minimizes, in the stochastic approximation sense, the following cost function,

$$J(\theta) = \mathbb{E} [|y - \theta^H x|],$$

and stability is guaranteed for sufficiently small values of μ [83].

The least-mean-fourth (LMF) algorithm

The scheme minimizes the following cost function,

$$J(\theta) = \mathbb{E} [|y - \theta^H x|^4],$$

and the corresponding update recursion is given by,

$$\theta_n = \theta_{n-1} + \mu |e_n|^2 x_n e_n^*.$$

It has been shown [101] that minimization of the fourth power of the error may lead to an adaptive scheme with better compromise between convergence rate and excess mean-square error than the LMS, if the noise source is sub-Gaussian. In a sub-Gaussian distribution, the tails of the pdf graph are decaying at a faster rate compared to the Gaussian one. The LMF could be seen as a version of the LMS with time-varying step-size, equal to $\mu |e_n|^2$. Hence, when the error is large, the step size increases, which helps the LMF to converge faster. On the other hand, when the error is small, the equivalent step size is reduced, leading the excess mean-square error to lower values. This idea turns out to work well when the noise is sub-Gaussian. However, the LMF tends to become unstable in the presence of outliers. This is also

understood, because for very large error values the equivalent step size becomes very large leading to instability. This is, for example, the situation when the noise follows a distribution with high-tails; that is, when the corresponding pdf curve decays at slower rates compared to the Gaussian, such as in the case of super-Gaussian distributions. Results concerning the analysis of the LMF can be found in Refs. [69, 70, 83]. Robustness and stability of the algorithm are guaranteed for sufficiently small values of μ [83].

Transform-domain LMS

We have already commented that the convergence speed of the LMS heavily depends on the condition number $\left(\frac{\lambda_{\max}}{\lambda_{\min}}\right)$ of the covariance matrix; this will soon be demonstrated in the examples below.

Transform-domain techniques exploit the decorrelation properties of certain transforms, such as the DFT and DCT, in order to decorrelate the input variables. When the input comprises a stochastic process, we say that such transforms “*prewhiten*” the input process. Moreover, in the case where the involved variables are part of a random process, one can exploit the time-shifting property and employ *block processing techniques*; by processing a block of data samples per time instant, one can exploit efficient implementations of certain transforms, such as the fast Fourier transform (FFT), to reduce the overall complexity. Such schemes are appropriate for applications where long filters are involved. For example, in some applications, such as echo cancellation, filter orders of a few hundred taps are commonly encountered [10, 39, 53, 66, 68].

Let T be a unitary transform in the complex domain, represented by $TT^H = T^HT = I$. Define

$$\hat{x}_n = T^H x_n, \quad (5.68)$$

and apply the transform matrix T^H on both sides of the complex LMS recursion in (5.66) to obtain

$$\hat{\theta}_n = \hat{\theta}_{n-1} + \mu \hat{x}_n e_n^*, \quad (5.69)$$

where

$$\hat{\theta}_n = T^H \theta_n. \quad (5.70)$$

Note that

$$\begin{aligned} e_n &= y_n - \theta_{n-1}^H x_n = y_n - \theta_{n-1}^H T T^H x_n \\ &= y_n - \hat{\theta}_{n-1}^H \hat{x}_n. \end{aligned}$$

Hence, the error term is not affected. Note that until now, we have not achieved much. Indeed,

$$\Sigma_{\hat{x}} = T^H \Sigma_x T,$$

is a similarity transformation that does not affect the condition number of the matrix, Σ_x ; it is known from linear algebra that both matrices share the same eigenvalues (Problem 5.14). Let us now choose $T = Q$, where Q is the unitary matrix comprising the orthonormal eigenvectors of Σ_x . For this case, $\Sigma_{\hat{x}} = \Lambda$, which is the diagonal matrix with entries the eigenvalues of Σ_x . In the sequel, we modify the transform-domain LMS in (5.69), so that to use a different step size per component, according to the following scenario,

$$\hat{\theta}_n = \hat{\theta}_{n-1} + \mu \Lambda^{-1} \hat{x}_n e_n^*, \quad (5.71)$$

or

$$\bar{\theta}_n = \bar{\theta}_{n-1} + \mu \bar{x}_n e_n^*, \quad (5.72)$$

where

$$\bar{\theta}_n := \Lambda^{1/2} \hat{\theta}_n,$$

and

$$\bar{x}_n := \Lambda^{-1/2} \hat{x}_n.$$

Note that the error is not affected, because $\bar{\theta}_n^H \bar{x}_n = \hat{\theta}_n^H \hat{x}_n$. We have now achieved our original goal, since

$$\Sigma_{\bar{x}} = \Lambda^{-1/2} \Sigma_{\hat{x}} \Lambda^{-1/2} = \Lambda^{-1/2} \Lambda \Lambda^{-1/2} = I.$$

That is, the condition number of $\Sigma_{\bar{x}}$ is equal to 1. In practice, this technique is difficult to apply, due to the complexity associated with the eigendecomposition task; more importantly, Σ_x must be known, which in adaptive implementations is not the case. In practice, we resort to unitary transforms, T , that approximately whiten the input, such as the DFT and DCT. Then, (5.71) is replaced by

$$\hat{\theta}_n = \hat{\theta}_{n-1} + \mu D^{-1} \hat{x}_n e_n^*,$$

where D is the diagonal matrix with entries the variances of the respective components of \hat{x}_n , or

$$[D]_{ii} = \mathbb{E} \left[(\hat{x}_n(i))^2 \right] = \sigma_i^2, \quad i = 1, 2, \dots, l,$$

where $\hat{x}_n(i)$ is the i th entry of \hat{x}_n , which has been assumed to be a zero mean vector; this is justified from the fact that if $\Sigma_{\hat{x}}$ is truly diagonal, and equal to Λ , the eigenvalues correspond to the variances of the respective elements. The entries σ_i^2 are estimated in a time-adaptive fashion, and the scheme given in [Algorithm 5.6](#) results.

Algorithm 5.6 (The transform-domain LMS).

- Initialization
 - $\hat{\theta}_{-1} = \mathbf{0}$; or any other value.
 - $\sigma_{-1}^2(i) = \delta, i = 1, 2, \dots, l; \delta$ a small value.
 - Choose μ , and $0 \ll \beta < 1$.
- **For** $n = 0, 1, 2, \dots$, **Do**
 - $\hat{x}_n = T^H x_n$
 - $e_n = y_n - \hat{\theta}_{n-1}^H \hat{x}_n$
 - $\hat{\theta}_n = \hat{\theta}_{n-1} + \mu D^{-1} \hat{x}_n e_n^*$
 - **For** $i = 1, 2, \dots, l$, **Do**
 - $\sigma_i^2(n) = \beta \sigma_i^2(n-1) + (1-\beta) |\hat{x}_n(i)|^2$
 - **End For**
 - $D = \text{diag}\{\sigma_i^2(n)\}$
- **End For**

Subband adaptive filters is a related family of algorithms where whitening is achieved via block data processing and the use of a multirate filter band. Such schemes were first proposed in Refs. [41, 53, 54] and have successfully been used in applications such as in echo cancellation [45, 53].

Another approach that has been adopted to improve the convergence rate, for the case of system identification applications, is to select the input excitation signal to be of a specific type. For example, in Ref. [5], it is pointed out that the excitation signal that optimizes the convergence speed of the NLMS algorithm is a deterministic perfect periodic sequence (PPSEQ) with period equal to the impulse response of the system. Such sequences have been used in Refs. [24, 25] to derive versions of the LMS,

which not only converge fast but they are also of very low computational complexity, requiring only one multiplication, one addition, and one subtraction per update.

5.9 SIMULATION EXAMPLES

Example 5.3. The goal of this example is to demonstrate the sensitivity of the convergence rate of the LMS on the eigenvalues spread of the input covariance matrix. To this end, two experiments were conducted, in the context of regression/system identification task. Data were generated according to our familiar model

$$y_n = \theta_o^T \mathbf{x}_n + \eta_n.$$

The (unknown) parameters, $\theta_o \in \mathbb{R}^{10}$, were randomly chosen from a $\mathcal{N}(0, 1)$ and then frozen. In the first experiment, the input vectors were formed by a white noise sequence with samples i.i.d. drawn from a $\mathcal{N}(0, 1)$. Thus, the input covariance matrix was diagonal with all the elements being equal to the corresponding noise variance (Section 2.4.3). The noise samples, η_n , were also i.i.d. drawn from a Gaussian with zero mean and variance $\sigma^2 = 0.01$. In the second experiment, the input vectors were formed by an AR(1) process with coefficient equal to $a_1 = 0.85$ and the corresponding white noise excitation was of variance equal to 1 (Section 2.4.4). Thus, the input covariance matrix is no more diagonal and the eigenvalues are not equal. The LMS was run on both cases with the same step-size $\mu = 0.01$. Figure 5.16 summarizes the results. The vertical axis (denoted as MSE) shows the squared

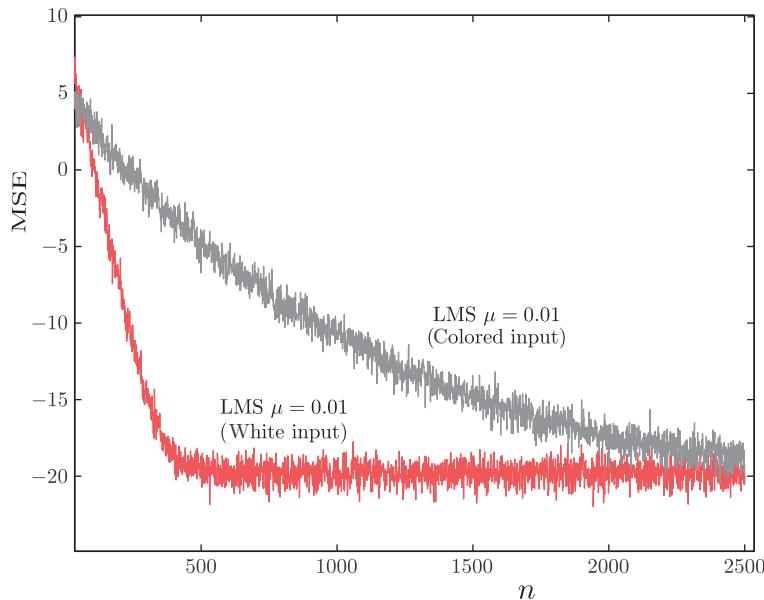


FIGURE 5.16

Observe that for the same step-size, the convergence of the LMS is faster when the input is white. The two curves are the result of averaging 100 independent experimental realizations.

error, e_n^2 , and the horizontal axis the time instants (iterations) n . Note that both curves level out at the same error floor. However, the convergence rate for the case of the white noise input is significantly higher. The curves shown in the figure are the result of averaging 100 independent experimental realizations.

It must be emphasized that, when comparing convergence performance of different algorithms, either all algorithms should converge to the same error floor and compare the respective convergence rates, or all algorithms should have the same convergence rate and compare respective error floors.

Example 5.4. In this example, the dependence of the LMS on the choice of the step-size is demonstrated. The unknown parameters, $\theta_o \in \mathbb{R}^{10}$, as well as the data were exactly the same with the white noise case of [Example 5.3](#).

The LMS was run using the generated samples, with two different step-sizes, namely, $\mu = 0.01$ and $\mu = 0.075$. The obtained averaged (over 100 realizations) curves are shown in [Figure 5.17](#). Observe that the larger the step-size, for the same set of observation samples, the faster the convergence becomes albeit at the expense of a higher error floor (misadjustment), in accordance to what has been discussed in [Section 5.5.1](#).

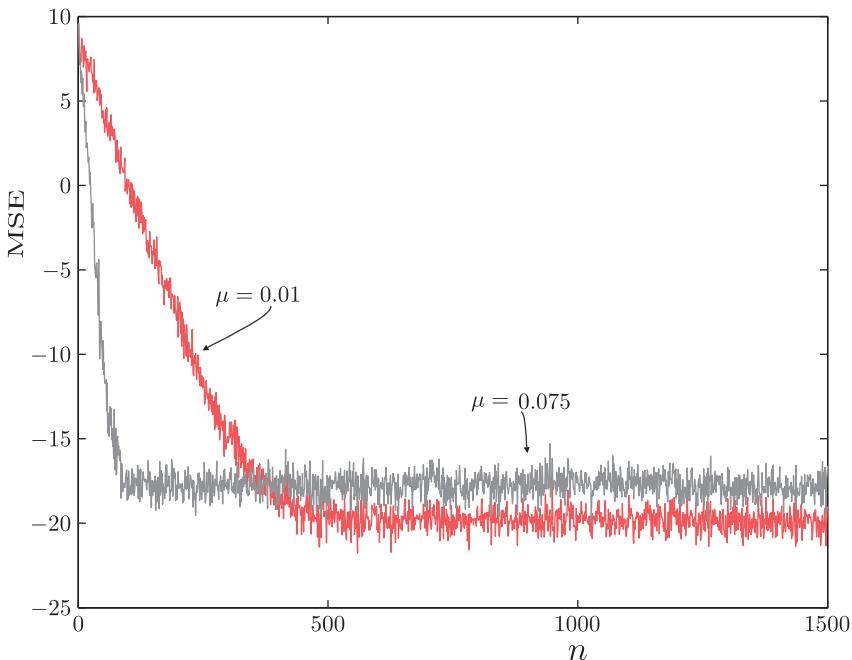


FIGURE 5.17

For the same input, the larger the step-size for the LMS the faster the convergence becomes, albeit at the expense of higher error floor.

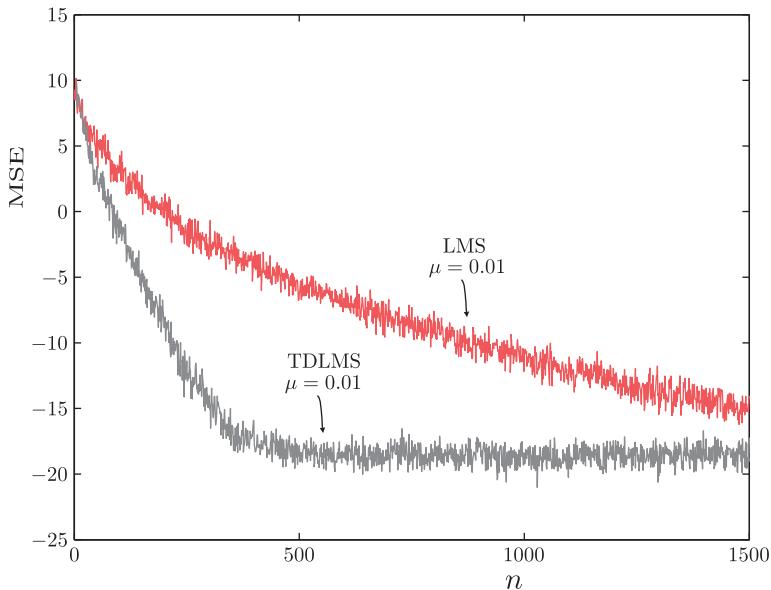


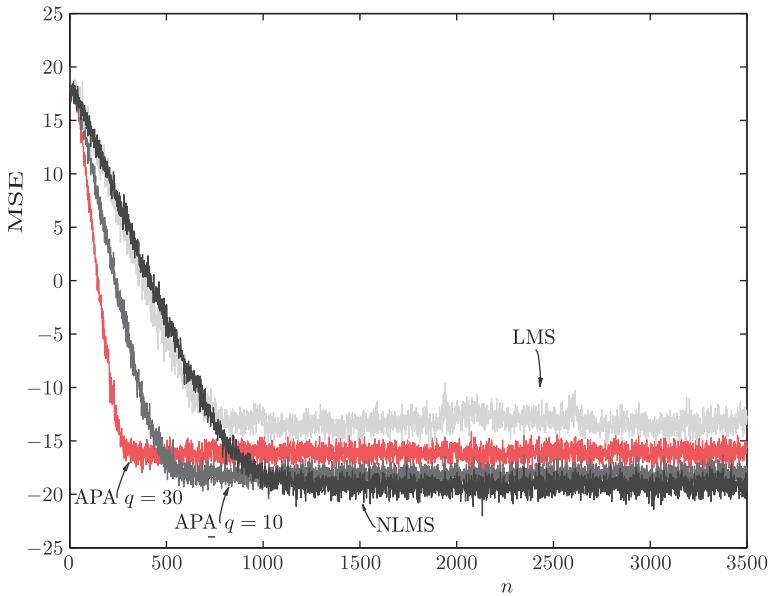
FIGURE 5.18

Observe that for the same step-size, the convergence of the transform-domain LMS is significantly faster compared to the LMS, for similar error floors. The higher the eigenvalues spread of the input covariance matrix is, the more the obtained performance improvement becomes.

Example 5.5 (LMS versus transform-domain LMS). In this example, the stage of the experimental setup is exactly the same as that considered in [Example 5.3](#) for the AR(1) case. The goal is to compare the LMS and the transform-domain LMS. [Figure 5.18](#) shows the obtained averaged error curves. The step-size was the same as the one used in [Example 5.3](#): $\mu = 0.01$; hence the curve for the LMS is the same as the corresponding one appearing in [Figure 5.16](#). Observe the significantly faster convergence achieved by the transform-domain LMS, due to its (approximate) whitening effect on the input.

Example 5.6. The experimental setup is similar to that of [Example 5.4](#), with the only exception that the unknown parameter vector was of higher dimension, $\theta_o \in \mathbb{R}^{60}$, so that the differences in the performance of the algorithms is more clear. The goal is to compare the LMS, the normalized LMS (NLMS) and the affine projection algorithm (APA). The step size of the LMS was chosen equal to $\mu = 0.025$ and for the NLMS $\mu = 0.35$ and $\delta = 0.001$, so that both algorithms have similar convergence rates. The step size for the APA was chosen equal to $\mu = 0.1$, so that $q = 10$ will settle at the same error floor as that of the NLMS. For the APA, we also chose $\delta = 0.001$. The results are shown in [Figure 5.19](#).

Observe the lower error floor, for the same convergence rate, obtained by the NLMS compared to the LMS and the improved performance obtained by the APA for $q = 10$. Increasing the number of past data samples (re)used in APA to $q = 30$, we can see the improved convergence rate that is obtained, although at the expense of higher error floor, as predicted by the theoretical results reported in [Section 5.6](#).

**FIGURE 5.19**

For the same step-size, the NLMS converges at the same rate to a lower error floor compared to the LMS. For the APA, increasing q , improves the convergence, at the expense of higher error floors.

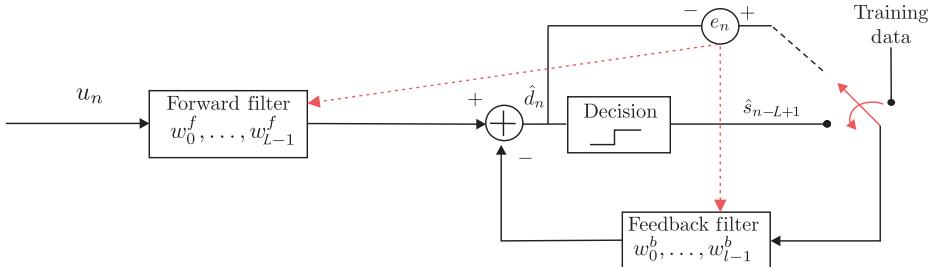
5.10 ADAPTIVE DECISION FEEDBACK EQUALIZATION

The task of channel equalization was introduced in [Figure 4.12](#). The input to the equalizer is a stochastic process (random signal), which, according to the notational convention introduced in [Section 2.4](#), will be denoted as u_n . Note that upon receiving the noisy and distorted by the (communications) channel sample, u_n , one has to obtain an estimate of the originally transmitted information sequence, s_n , delayed by L time lags, which accounts for the various delays imposed by the overall transmission system involved. Thus, at time n , the equalizer decides for \hat{s}_{n-L+1} . Ideally, if one knew the true values of the initially transmitted information sequence up to and including time instant $n-L$, represented by $s_{n-L}, s_{n-L-1}, s_{n-L-2} \dots$, it could only be beneficial to use this information, together with the received sequence, u_n , to recover an estimate of \hat{s}_{n-L+1} . This idea is explored in the *decision feedback equalizer* (DEE). The equalizer's output, for the complex-valued data case, is now written as

$$\begin{aligned}\hat{d}_n &= \sum_{i=0}^{L-1} w_i^f u_{n-i} + \sum_{i=0}^{l-1} w_i^b s_{n-L-i} \\ &= \mathbf{w}^H \mathbf{u}_{e,n}\end{aligned}\tag{5.73}$$

where

$$\mathbf{w} := \begin{bmatrix} \mathbf{w}^f \\ \mathbf{w}^b \end{bmatrix} \in \mathbb{C}^{L+l}, \quad \mathbf{u}_{e,n} := \begin{bmatrix} \mathbf{u}_n \\ s_n \end{bmatrix} \in \mathbb{C}^{L+l},$$

**FIGURE 5.20**

The forward part of the DFE acts on the received samples, while the backward acts on the training data/decisions, depending on the mode of operation.

and $s_n := [s_{n-L}, \dots, s_{n-L-l+1}]^T$. The desired response at time n is

$$d_n = s_{n-L+1}.$$

In practice, after the initial training period, the information samples, s_{n-L-i} are replaced by their computed estimates, \hat{s}_{n-L-i} , $i = 0, 1, \dots, l-1$, which are available from decisions taken in previous time instants. It is said that the equalizer operates in the *decision directed* mode. The basic DFE structure is shown in [Figure 5.20](#). Note that during the training period, the parameter vector, w , is trained so that to minimize the power of the error,

$$e_n = d_n - \hat{d}_n = s_{n-L+1} - \hat{d}_n.$$

Once all the available training samples have been used, training carries on using the estimates \hat{s}_{n-L+1} . For example, for a binary information sequence $s_n \in \{1, -1\}$, the decision concerning the estimate at time n is obtained by passing \hat{d}_n through a threshold device and \hat{s}_{n-L+1} is obtained. Note that DFE is one of the early examples of *semisupervised* learning, where training data is not enough, and then the estimates are used for training [\[94\]](#). In this way, assuming that at the end of the training phase, $\hat{s}_{n-L+1} = s_{n-L+1}$, and also that time variations are slow, so that to guarantee that $\hat{d}_n \simeq d_n$, then we expect, with good enough probability, that \hat{s}_{n-L+1} will remain equal to s_{n-L+1} , so that the equalizer can track the changes. More on DFEs and their error performance can be obtained from Ref. [\[77\]](#).

Any of the adaptive schemes treated so far can be used in a DFE scenario by replacing in the input vector, \mathbf{u}_e , the term s_n by \hat{s}_n , when operating in the decision directed mode. Note that adaptive algorithms in the context of the equalization task were employed first in Refs. [\[44, 78\]](#). A version of the DFE, operating in the frequency domain, has been proposed for the first time in [\[14\]](#).

Thus the LMS recursion for the linear DFE, in its complex-valued formulation, becomes,

$$\hat{d}_n = \mathbf{w}_{n-1}^H \mathbf{u}_{e,n}$$

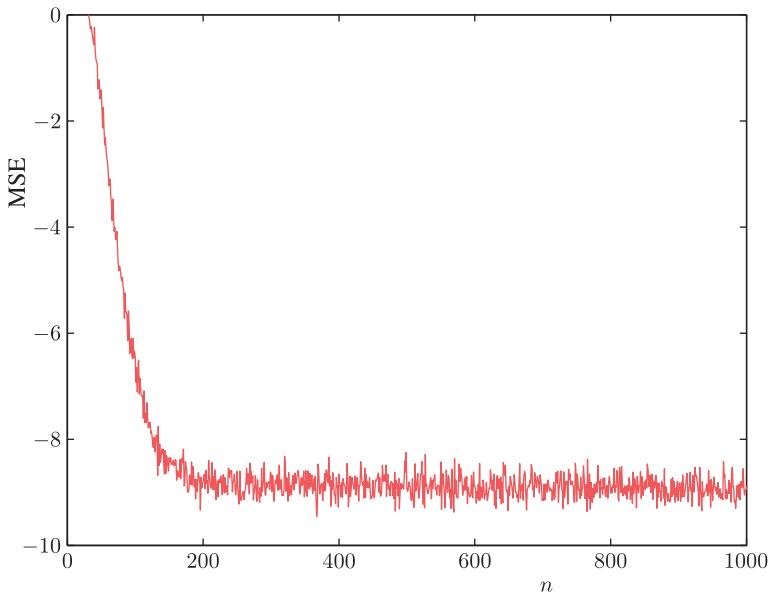
$d_n = s_{n-L+1}$; in the training mode, or

$$d_n = T[\hat{d}_n]; \text{ in the decision directed mode,}$$

$$e_n = d_n - \hat{d}_n$$

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mu \mathbf{u}_{e,n} e_n^*,$$

where $T[\cdot]$ denotes the thresholding operation.

**FIGURE 5.21**

The MSE in dBs for the DFE of [Example 5.7](#). After the time instant $n = 250$, the LMS is trained with the decisions \hat{s}_{n-L+1} .

Example 5.7. Let us consider a communication system where the input information sequence comprises a stream of randomly generated symbols $s_n = \pm 1$, with equal probability. This sequence is sent to a channel with impulse response,

$$\mathbf{h} = [0.04, -0.05, 0.07, -0.21, 0.72, 0.36, 0.21, 0.03, 0.07]^T.$$

The output of the channel is contaminated by white Gaussian noise at the SNR = 11dB level. A DFE is used with the length of the feedforward section being equal to $L = 21$ and the length of the feedback part equal to $l = 10$. The DFE was trained with 250 symbols; then, it was switched on to the decision directed mode and it was run for 10,000 iterations. At each iteration, the decision ($\text{sgn}(\hat{d}_n)$) was compared with the true transmitted symbol s_{n-L+1} . The error rate (total number of errors over the corresponding number of transmitted symbols), was approximately 1%. [Figure 5.21](#) shows the averaged MSE curve as a function of the number of iterations. For the LMS, we used $\mu = 0.025$.

5.11 THE LINEARLY CONSTRAINED LMS

The task of linearly constrained MSE estimation was introduced in [Section 4.9.2](#). Here, we turn our attention into its online stochastic gradient counterpart. The discussion will be carried out within the notational convention used for linear filtering involving processes, since a typical application is that of beamforming; this is also the reason that we will involve the complex-valued formulation. However, everything to be said carries on to the more general linear estimation task.

In [Section 4.9.2](#), the goal was to minimize, subject to a set of constraints, either the noise variance or the output of the filter (beamformer), leading to the costs $\mathbf{w}^H \Sigma_\eta \mathbf{w}$ or $\mathbf{w}^H \Sigma_u \mathbf{w}$, respectively. In a more general setting, we will require the output to be “close” to a desired response random signal d_n . For the beamforming setting, this corresponds to a desired (training) signal; that is, besides the desired direction, which is provided via the constraints, we are also given a desired signal sequence. Moreover, we will assume that our solution has to satisfy more than one constraints. The task now becomes,

$$\begin{aligned}\mathbf{w}_* &= \arg \min_{\mathbf{w}} \mathbb{E} \left[|d_n - \mathbf{w}^H \mathbf{u}_n|^2 \right] \\ \text{s.t. } \quad \mathbf{w}^H \mathbf{c}_i &= g_i, \quad i = 1, 2, \dots, m.\end{aligned}\tag{5.74}$$

for some $g_i \in \mathbb{R}$. The corresponding Lagrangian is,

$$\begin{aligned}L(\mathbf{w}) &= \sigma_d^2 + \mathbf{w}^H \mathbb{E}[\mathbf{u}_n \mathbf{u}_n^H] \mathbf{w} - \mathbf{w}^H \mathbb{E}[\mathbf{u}_n d_n^*] - \mathbb{E}[\mathbf{u}_n^H d_n] \mathbf{w} \\ &\quad + \sum_{i=1}^m \lambda_i (\mathbf{w}^H \mathbf{c}_i - g_i),\end{aligned}$$

where λ_i , $i = 1, 2, \dots, m$, are the corresponding Lagrange multipliers. Taking the gradient w.r.t. \mathbf{w}^* and considering \mathbf{w} to be a constant, we get

$$\nabla_{\mathbf{w}^*} L(\mathbf{w}) = \mathbb{E}[\mathbf{u}_n \mathbf{u}_n^H] \mathbf{w} - \mathbb{E}[\mathbf{u}_n d_n^*] + \sum_{i=1}^m \lambda_i \mathbf{c}_i.$$

Applying the Robbins-Monro scheme to find the root, we get,

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \mu \mathbf{u}_n e_n^* - \mu C \boldsymbol{\lambda}_n,\tag{5.75}$$

where we have used a constant step-size, μ , and

$$\hat{d}_n = \mathbf{w}_{n-1}^H \mathbf{u}_n,$$

and

$$e_n = d_n - \hat{d}_n,$$

and have allowed $\boldsymbol{\lambda}$ to be time-dependent. C is defined as the matrix having as columns the vectors \mathbf{c}_i , $i = 1, 2, \dots, m$.

Plugging (5.75) into the constraints in (5.74), which can be compactly written as $C^H \mathbf{w} = \mathbf{g}$ (recall, $g_i \in \mathbb{R}$), we readily obtain

$$-\mu \boldsymbol{\lambda}_n = (C^H C)^{-1} \mathbf{g} - (C^H C)^{-1} C^H (\mathbf{w}_{n-1} + \mu \mathbf{u}_n e_n^*),$$

and the update recursion becomes

$$\mathbf{w}_n = \left(I - C(C^H C)^{-1} C^H \right) (\mathbf{w}_{n-1} + \mu \mathbf{u}_n e_n^*) + C(C^H C)^{-1} \mathbf{g}.$$

Note that $(I - C(C^H C)^{-1} C^H)$ is the orthogonal projection matrix on the intersection (affine set) of the hyperplanes defined by the constraints (recall that, $C(C^H C)^{-1} C^H$ is the respective projection on the subspace spanned by \mathbf{c}_i , $i = 1, 2, \dots, m$). In case the goal is to minimize the output, then one has to set in e_n the desired response $d_n = 0$.

The constrained LMS was first treated in [40]. Besides the previously used constraints, other constraints can also be used, for example, for the constrained normalized LMS, one additionally demands

$$\mathbf{w}_n^H \mathbf{u}_n = d_n$$

see (e.g., [6]).

5.12 TRACKING PERFORMANCE OF THE LMS IN NONSTATIONARY ENVIRONMENTS

We have already considered the convergence behavior of the LMS and made related comments for the other algorithms that have been discussed. As it has been stated before, convergence is a *transient* phenomenon; that is, it concerns the period from the initial kick-off point, to the steady-state. The steady-state was also discussed for stationary environments; that is, environments in which the unknown parameter vector as well as the underlying statistics of the involved random variables/processes remain unchanged.

We now turn our focus to cases where the true (yet unknown) parameter vector/system undergoes changes. Thus, this affects the output observations and consequently their statistics. Note that the statistics of the input can also change. However, we are not going to consider such cases, as the analysis can become quite involved. Our goal is to study the *tracking* performance of the LMS; that is, the ability of the algorithm to track changes of the unknown parameter vector. Note that tracking is a *steady-state* phenomenon. In other words, we assume that enough time has elapsed so that the influence of the initial conditions has been forgotten and has no effect, any more, on the algorithm. Tracking agility and convergence speed are two *different* properties of an algorithm. An algorithm may converge fast, but it *may not* necessarily have a good tracking performance and vice versa. We will see such cases later on.

The setting of our discussion will be similar to that of Section 5.5.1. In conformity with the discussion there, we consider the real-data case; similar results hold true for the complex-valued linear estimation scenario. However, in contrast to the adopted model in (5.37), a *time-varying* model is adopted here using the following assumptions.

Assumption 1. The output observations are generated according to the model,

$$\mathbf{y}_n = \boldsymbol{\theta}_{o,n-1}^T \mathbf{x} + \eta, \quad (5.76)$$

which is in line with the prediction model used in LMS, at time n . That is, the unknown set of parameters is a time-varying one. The statistical properties of the input vector, \mathbf{x} , as well as the noise variable η are assumed to be time-independent, and this is the reason we have not used the time index; equivalently, in the case where the input is a random process, \mathbf{u}_n , it is assumed to be *stationary*. Moreover, the input variables are assumed to be independent of the zero mean noise variable, η . Furthermore, successive samples of η are i.i.d. (white noise sequence) of variance σ_η^2 . So far, we have not gone much beyond the Assumption 1, stated in Section 5.5.1.

Assumption 2. The time varying model follows a random walk variation, represented by

$$\boldsymbol{\theta}_{o,n} = \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}. \quad (5.77)$$

The random vector $\boldsymbol{\omega}$ is assumed to be zero mean with covariance matrix

$$\mathbb{E} [\boldsymbol{\omega} \boldsymbol{\omega}^T] = \Sigma_\omega.$$

Note that the variance of a random walk grows unbounded with time; this is readily shown by applying (5.77) recursively.

A variant of this model would be more sensible to use,

$$\theta_{o,n} = a\theta_{o,n-1} + \omega,$$

with $|a| < 1$ [106]. However, the analysis gets more involved, so we will stick with the model in (5.77). After all, our goal here is to highlight and have a first touch on the notion of tracking and get an idea of its effects on the misadjustment in steady-state.

Assumption 3. As in Section 5.5.1, we assume that $\mathbf{c}_{n-1} := \theta_{n-1} - \theta_{o,n-1}$ is independent of \mathbf{x} and η . This time, we will also assume independence of \mathbf{c}_n and ω .

Recall from (5.47) that, the excess mean-square error, at time n , is given by

$$J_{\text{exc},n} = \text{trace}\{\Sigma_x \Sigma_{c,n-1}\}.$$

Thus, our goal now is to compute $\Sigma_{c,n-1}$ for the time-varying model case. It is straightforward to see that the counterpart of (5.34) now becomes,

$$\mathbf{c}_n = (I - \mu \mathbf{x} \mathbf{x}^T) \mathbf{c}_{n-1} + \mu \mathbf{x} \eta - \omega. \quad (5.78)$$

Adopting the previously stated three assumptions, as well as the *Gaussian* assumption for \mathbf{x} , and following exactly the same steps used for (5.40), we end up with,

$$\begin{aligned} \Sigma_{c,n} &= \Sigma_{c,n-1} - \mu \Sigma_x \Sigma_{c,n-1} - \mu \Sigma_{c,n-1} \Sigma_x + 2\mu^2 \Sigma_x \Sigma_{c,n-1} \Sigma_x \\ &\quad + \mu^2 \Sigma_x \text{trace}\{\Sigma_x \Sigma_{c,n-1}\} + \mu^2 \sigma_\eta^2 \Sigma_x + \Sigma_\omega. \end{aligned} \quad (5.79)$$

Note that if complex data are involved, the only difference is that the fourth term on the right-hand side is not multiplied by 2 (Problem 5.15). This equation governs the propagation of $\Sigma_{c,n}$, which in turn can provide the excess MSE error.

A more convenient form results for small values of μ , where the fourth and fifth terms on the right-hand side can be neglected, being small w.r.t. $\mu \Sigma_x \Sigma_{c,n-1}$. Moreover, at the steady-state, $\Sigma_{c,n} = \Sigma_{c,n-1} := \Sigma_c$, and taking the trace on both sides, we end up with (recall $\text{trace}\{A + B\} = \text{trace}\{A\} + \text{trace}\{B\}$ and $\text{trace}\{AB\} = \text{trace}\{BA\}$),

$$J_{\text{exc}} = \text{trace}\{\Sigma_x \Sigma_c\} = \frac{1}{2} \left(\mu \sigma_\eta^2 \text{trace}\{\Sigma_x\} + \frac{1}{\mu} \text{trace}\{\Sigma_\omega\} \right). \quad (5.80)$$

Note that this is exactly the same approximation as the one resulting from the more sound theory of energy conservation [83].

Compare (5.80) with (5.52); in the current setting, there is another term associated with the noise, which drifts the model around its mean. Thus, the excess MSE is contributed (a) by the inability of the LMS to obtain the optimum value exactly, and (b) there is an extra term measuring its “inertia” to track the changes of the model fast enough. This is the most important outcome of the current discussion. In time-varying environments, the misadjustment increases. Moreover, looking at (5.80) and at the effect of μ , it is observed that small values of μ have a beneficial effect on the first term, but they increase the contribution of the second one. The opposite is true for relatively big values of μ . This is natural. Small step-sizes give the algorithm the chance to learn better under stationary environments, but the

algorithm cannot track the changes fast enough. Thus, the choice of μ should be the outcome of a trade-off. Minimizing the excess error in (5.80), it is easily shown to result in

$$\mu_{\text{opt}} = \sqrt{\frac{\text{trace}\{\Sigma_{\omega}\}}{\sigma_{\eta}^2 \text{trace}\{\Sigma_x\}}}.$$

Note, however, that such choices for μ are of theoretical importance only. In practice, the time variation of the system can hardly correspond to that of the adopted model; the latter, due to the complexity of the analysis, was chosen to be a simple one in an effort to simplify the mathematical manipulations. Moreover, for the sake of simplifying the analysis, a number of assumptions were adopted. In practice, μ is chosen more according to the user's practical experience, after experimentation, than based on the theory. The theory, however, has pointed out the trade-off between the speed of convergence and that of tracking.

More mathematically rigorous analysis of the performance of online/adaptive schemes in non-stationary environments can be obtained from Refs. [16, 38, 47, 61, 70, 83]. Simulation results, demonstrating the tracking performance of the LMS compared to other algorithms are given in Example 6.3.

5.13 DISTRIBUTED LEARNING: THE DISTRIBUTED LMS

The focus of our attention is now turned toward a problem that has gained increasing importance over the last decade or so. There is a growing number of applications where data are received/reside in different sensors/databases, which are spatially distributed. However, all this spatially distributed information has to be exploited towards achieving a common goal; that is, to perform a *common estimation/inference* task. We refer to such tasks as *distributed* or *decentralized* learning. At the heart of this problem lies the concept of *cooperation*, which is another name for the process of exchanging learning experience/information in order to reach a common goal/decision. Human societies have survived because of cooperation (and have disappeared due to lack of cooperation).

Distributed learning is common in many biological systems, where no individual/agent is in charge, yet the group exhibits a high degree of intelligence (we humans refer to it as instinct). Look at the way birds fly in formation and bees swarm in a new hive.

Besides sociology and biology, science and engineering have used the concept of distributed learning; *wireless sensor networks* (WSNs) are a typical example. WSNs were originally suggested as spatially distributed autonomous sensors to monitor physical and environmental conditions, such as pressure, temperature, sound and to cooperatively pass their data to a central unit. Although WSNs were originally motivated for military applications, today are targeted at a diverse number of applications, such as traffic control, homeland security and surveillance, health care and environmental modeling. Each sensor node is equipped with an onboard processor, in order to perform locally some simple processing and transmit the required and *partially* processed data. Sensors/nodes are characterized by low processing, memory, and communication capabilities due to low energy and bandwidth constraints [1, 104].

Other typical examples of distributed learning applications are the modeling and study of the way individuals are linked in social networks, modeling pathways defined over complex power grids, cognitive radio systems, and pattern recognition; the common characteristic of all these applications is

that data are partially processed in each individual node/agent, and the processed information is passed over to the network under a *certain protocol*.

The obvious question that the unfamiliar reader may ask is why not use only one node/agent and the locally residing information to perform the inference task. The answer, of course, is that we can come with better estimates/results by exploiting the available data/information across the whole network. This brings us to the notion of *consensus*.

According to the American Heritage dictionary, “consensus” is defined as “an opinion or position reached by a group as a whole”; that is, consensus is the process that guarantees an “accepted agreement” within the group. The term “accepted agreement” is not uniquely defined. In some cases, this may refer to a *unanimous* decision, in other cases it refers to a *majority* rule. In some cases, all the opinions of all agents are equally weighted whereas in others, different weights are imposed, based on some relative significance measures. However, in all cases, the essence of any consensus-based process is the trust that a “better” decision is reached when compared to the process of each agent/person acting individually.

In this section, we focus on the task of *parameter estimation*. Each individual agent has access to partial information via a “local” acquisition process of data. Although each agent has access to a different set of data, they all share a common goal; that is, to estimate the *same* unknown set of parameters. This task will be achieved in a collaborative manner. However, different cooperation scenarios can be adopted.

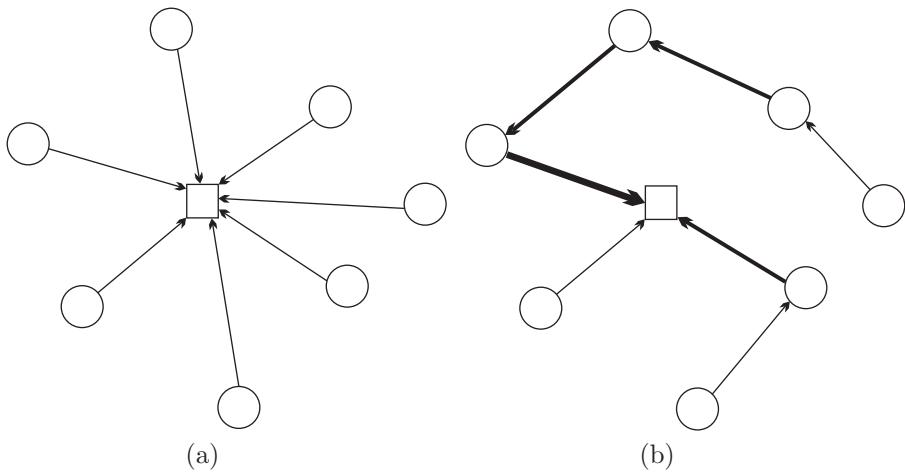
5.13.1 COOPERATION STRATEGIES

In distributed learning, each individual agent is represented as a node in a graph. Edges between nodes indicate that the respective agents can exchange information. Undirected edges indicate that information can be exchanged in both directions, while directed edges indicate the allowed direction of information flow.⁷

Centralized networks

Under this scenario of cooperation, nodes communicate their measurements to a central *fusion* unit for processing. The obtained estimate can be communicated back to each one of the nodes. [Figure 5.22](#) illustrates the topology. In [Figure 5.22a](#) all nodes are connected directly to the fusion center, indicated by a square. In [Figure 5.22b](#), some of the nodes can be linked directly to the fusion center, while others communicate their measurements to a linked neighbor, which then passes the received as well as the locally available observations/measurements either to a neighboring node or to the fusion center. The major advantage in this cooperation strategy is that the fusion center can compute optimal estimates, since it has access to all the available information. However, the optimality is obtained under a number of drawbacks, such as: demand for increased communication costs and delays, especially for large networks. In addition, when the fusion center breaks down, the whole network collapses. Moreover, in certain applications, privacy issues are involved. For example, when data concern medical records, the nodes do not wish to send the available (training) data, but it is preferably to communicate certain locally obtained processed information. To overcome the drawbacks of the centralized processing scenario, different distributed processing schemes have been proposed.

⁷ More rigorous definitions on graphs are given in [Chapter 15](#).

**FIGURE 5.22**

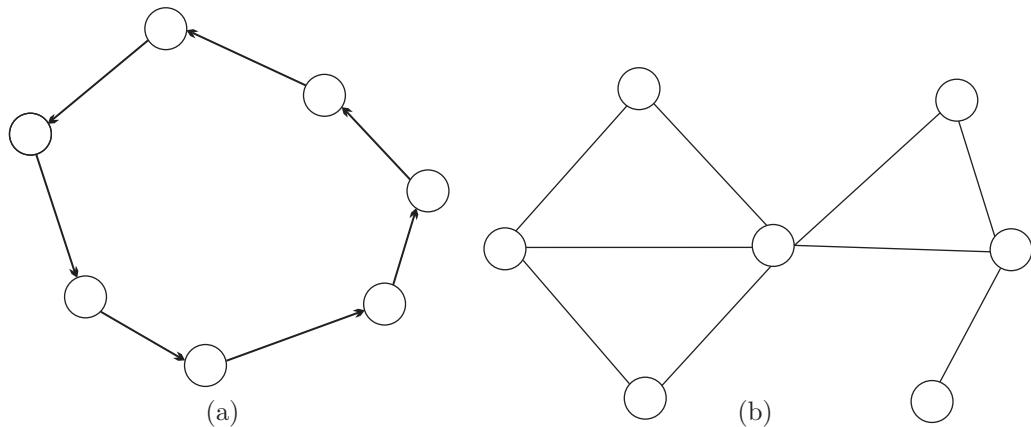
The square indicates the fusion center. (a) All nodes communicate directly to the fusion center. (b) Some nodes are connected directly to the fusion center. Others, communicate their own data to a neighboring node and so on, until the information reaches the fusion center. The bolder a connection is drawn, the higher the amount of data transmitted via the corresponding link.

Decentralized networks

Under this scenario, there is no central fusion center. Processing is performed *locally* at each node, employing the locally received measurements, and in the sequel, each node communicates the locally obtained estimates to its neighbors; that is, to the nodes it is linked with. These links are denoted as edges in the respective graph. There are different decentralized schemes.

- *Incremental/ring networks*: These require the existence of a *cyclic path*⁸ following the edges through the network. Starting from a node, such a cycle has to visit every node, at least once, and then return to the first one. Such a topology implements an *iterative* computational scheme. At each iteration, every node performs its data acquisition and processing locally and communicates the required information to its neighbor in the cyclic path. It has been shown that incremental schemes achieve global performance (e.g., [58]). The main disadvantage of this mode of cooperation is that, cycling information around at each iteration is a problem in large networks. It is also important to stress at this point that the construction and maintenance of a cyclic graph, visiting each node, is an NP-hard task [52]. Moreover, the whole network collapses if one node malfunctions. The corresponding graph topology is shown in Figure 5.23a.
- *Ad hoc networks*: According to this philosophy of cooperation, nodes perform locally data acquisition as well as processing, at each iteration. However, the constraint of a cyclic path is removed. Each node communicates information to its neighboring nodes with which it shares an

⁸ Consider a set of nodes x_1, \dots, x_k in a graph such that there is an edge connecting (x_{i-1}, x_i) , $i = 2, \dots, k$. The set of edges connecting the k nodes is a path.

**FIGURE 5.23**

(a) The incremental or ring topology. The information flow follows a cyclic path. (b) Topology corresponding to a diffusion strategy. Each node communicates information to the nodes with which it shares an edge.

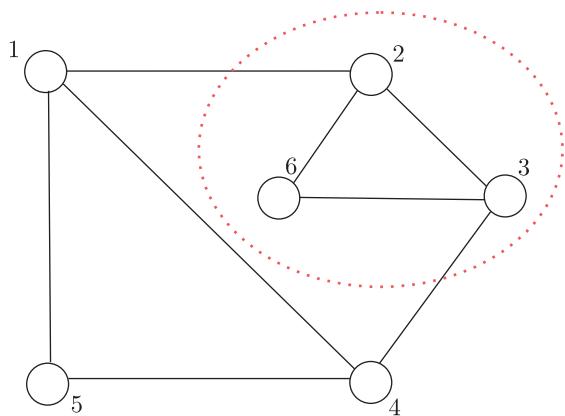
edge; in this way, information is *diffused* across the whole network. An advantage of such schemes is that operation is not seized if some nodes are malfunctioning. Also, the topology of the network may not be fixed. The price one pays for such “extras” is that the final obtained performance, after convergence, is inferior to those obtained by its incremental counterpart and by the centralized processing. This is natural, since at each iteration every node has access to only a limited amount of information. Figure 5.23b illustrates an example of the topology for ad hoc networks.

Besides the previous schemes, a number of variants exists. For example, the neighbors of each node may change probabilistically, which introduces randomness in the way information is diffused at each iteration [33, 60].

In this section, our focus will be on diffusion schemes. Our aim is to provide the reader with a sample of basic techniques around the LMS scheme and not to cover distributed learning in general, which is a field on its own with a long history; see [9, 19, 29, 51, 76, 95, 107] for sample references from classical to more recent contributions to the field. Besides distributed inference, a number of related aspects concerning the topology of the network and learning over graphs are attracting a lot of attention in the context of the emerging field of *complex networks*; see, [18, 37, 87, 89, 100] and the references therein.

5.13.2 THE DIFFUSION LMS

Let us consider a network of K agents/nodes. Each node exchanges information with the nodes in its neighborhood. Given a node, k , in a graph, let \mathcal{N}_k be the set of nodes with which this node shares an edge; moreover, node k is also included in \mathcal{N}_k . This comprises the neighborhood set of k . We will denote the cardinality of this set as n_k . Figure 5.24 shows a graph with six nodes. For example, the neighborhood of node $k = 6$ is $\mathcal{N}_6 = \{2, 3, 6\}$, with cardinality $n_6 = 3$. The cardinality of \mathcal{N}_k is also

**FIGURE 5.24**

A graph corresponding to a network operating under a diffusion strategy. The red dotted line encircles the nodes comprising the neighborhood of node $k = 6$.

known as the *degree* of node k . On the contrary, nodes $k = 6$ and $k = 5$ are not neighbors, because they are not directly linked via an edge. For the needs of the section, we assume that the graph is a *strongly connected* one; that is, there is at least one path of edges that connects any pair of nodes.

Each node in the network has access to a local data acquisition process, that provides the pair of training data⁹ $(y_k(n), \mathbf{x}_k(n))$, $k = 1, 2, \dots, K$, $n = 0, 1, \dots$, which are i.i.d. observations drawn from the respected stochastic zero-mean jointly distributed variables, y_k , \mathbf{x}_k , $k = 1, \dots, K$. We further assume that, in all cases, the pairs of the input-output variables are associated with a common to all (unknown) parameter vector $\boldsymbol{\theta}_o$. For example, in every node, the data are assumed to be generated by a corresponding regression model

$$y_k = \boldsymbol{\theta}_o^T \mathbf{x}_k + \eta_k, \quad k = 1, 2, \dots, K, \quad (5.81)$$

where \mathbf{x}_k , as well as the zero mean noise variable, η_k , obey, in general, *different* statistical properties in each node. We will discuss such applications soon.

Treating each node individually, the MSE optimal solution, which minimizes the *local* cost function,

$$J_k(\boldsymbol{\theta}) = \mathbb{E} [|y_k - \boldsymbol{\theta}^T \mathbf{x}_k|^2],$$

will be given by the respective normal equations, involving the respective covariance matrix and cross-correlation vector; in other words,

$$\Sigma_{x_k} \boldsymbol{\theta}_* = \mathbf{p}_k. \quad (5.82)$$

Recall that for the case of a regression model, $\boldsymbol{\theta}_* = \boldsymbol{\theta}_o$ and the same solution results from all nodes. No doubt, if the statistics Σ_{x_k} , \mathbf{p}_k , $k = 1, 2, \dots, K$, were known we could stop here. However, we already know that this is not the case and in practice they have to be estimated. Alternatively, one has to resort

⁹ The time index is used in parentheses, to unclutter notation due to the presence of the node index k .

to iterative techniques to learn the statistics as well as the unknown parameters. This is where one has to consider all nodes, to benefit from all the measurements/observations, which are distributed across the network. Thus, a more natural criterion to adopt is

$$J(\boldsymbol{\theta}) = \sum_{k=1}^K J_k(\boldsymbol{\theta}) = \sum_{k=1}^K \mathbb{E} [|y_k - \boldsymbol{\theta}^T \mathbf{x}_k|^2]. \quad (5.83)$$

Using the standard arguments, which we have employed a number of times so far, it is readily seen that the (common) estimate of the unknown $\boldsymbol{\theta}_o$ will be provided as a solution of

$$\left(\sum_{k=1}^K \boldsymbol{\Sigma}_{x_k} \right) \boldsymbol{\theta}_* = \sum_{k=1}^K \mathbf{p}_k.$$

Let us use the global cost in (5.83) as our kick-off point to apply a gradient descent optimization scheme

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu \sum_{k=1}^K \left(\mathbf{p}_k - \boldsymbol{\Sigma}_{x_k} \boldsymbol{\theta}^{(i-1)} \right), \quad (5.84)$$

from which a corresponding stochastic gradient scheme results, by replacing expectations with instantaneous observations and associating iteration steps with time updates, so that

$$\begin{aligned} \boldsymbol{\theta}_n &= \boldsymbol{\theta}_{n-1} + \mu \sum_{k=1}^K \mathbf{x}_k(n) e_k(n), \\ e_k(n) &= y_k(n) - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_k(n), \end{aligned}$$

and a constant step size has been used, adopting the rationale behind the classical LMS formulation. Such an LMS-type recursion is perfect for a centralized scenario, where all data are transmitted to a fusion center. This is one of the extremes, having at its opposite end the scenario with nodes acting individually without cooperation. However, there is an intermediate path, which will lead us to the distributed diffusion mode of operation.

Instead of trying to minimize (5.83), let us select a specific node k , and construct a local cost as the weighted aggregate in \mathcal{N}_k , represented by

$$J_k^{loc}(\boldsymbol{\theta}) = \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}), \quad k = 1, 2, \dots, K, \quad (5.85)$$

so that

$$\sum_{k=1}^K c_{mk} = 1, \quad c_{mk} \geq 0, \quad \text{and } c_{mk} = 0 \quad \text{if } m \notin \mathcal{N}_k, \quad m = 1, 2, \dots, K. \quad (5.86)$$

Let C be the $K \times K$ matrix with entries $[C]_{mk} = c_{mk}$, then the summation condition in (5.86) can be written as

$$C\mathbf{1} = \mathbf{1}, \quad (5.87)$$

where $\mathbf{1}$ is the vector with all its entries being equal to 1. That is, all the entries across a row are summing to 1. Such matrices are known as *right stochastic* matrices. In contrast, a matrix is said to be left stochastic if

$$C^T \mathbf{1} = \mathbf{1}.$$

Also, a matrix that is both left and right stochastic is known as *doubly stochastic* (Problem 5.16). Note that due to this matrix constraint, we still have that

$$\begin{aligned} \sum_{k=1}^K J_k^{loc}(\boldsymbol{\theta}) &= \sum_{k=1}^K \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) = \sum_{k=1}^K \sum_{m=1}^K c_{mk} J_m(\boldsymbol{\theta}) \\ &= \sum_{m=1}^K J_m(\boldsymbol{\theta}) = J(\boldsymbol{\theta}). \end{aligned}$$

That is, summing all local costs, the global one results.

Let us focus on minimizing (5.85). The gradient descent scheme results in

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} (\mathbf{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)}).$$

However, since nodes in the neighborhood exchange information, they could also share their current estimates. This is justified by the fact that the ultimate goal is to reach a common estimate; thus, exchanging current estimates could be used for the benefit of the algorithmic process to achieve this goal. To this end, we will modify the cost in (5.85) by regularizing it, leading to

$$\tilde{J}_k^{loc}(\boldsymbol{\theta}) = \sum_{m \in \mathcal{N}_k} c_{mk} J_m(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|^2, \quad (5.88)$$

where $\tilde{\boldsymbol{\theta}}$ encodes information with respect to the unknown vector, which is obtained by the neighboring nodes and $\lambda > 0$. Applying the gradient descent on (5.88) (and absorbing the factor “2”, which comes from the exponents, into the step-size), we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} (\mathbf{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)}) + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}_k^{(i-1)}), \quad (5.89)$$

which can be broken into the following two steps:

$$\text{Step 1: } \boldsymbol{\psi}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} (\mathbf{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)}),$$

$$\text{Step 2: } \boldsymbol{\theta}_k^{(i)} = \boldsymbol{\psi}_k^{(i)} + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\psi}_k^{(i)}).$$

Step 2 can slightly be modified and replace $\boldsymbol{\theta}_k^{(i-1)}$ by $\boldsymbol{\psi}_k^{(i)}$, since this encodes more recent information, and we obtain

$$\boldsymbol{\theta}_k^{(i)} = \boldsymbol{\psi}_k^{(i)} + \mu_k \lambda (\tilde{\boldsymbol{\theta}} - \boldsymbol{\psi}_k^{(i)}).$$

Furthermore, a reasonable choice of $\tilde{\boldsymbol{\theta}}$, at each iteration step, would be

$$\tilde{\boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}}^{(i)} := \sum_{m \in \mathcal{N}_{k \setminus k}} b_{mk} \boldsymbol{\psi}_m^{(i)},$$

where

$$\sum_{m \in \mathcal{N}_{k \setminus k}} b_{mk} = 1, \quad b_{mk} \geq 0,$$

and $\mathcal{N}_{k \setminus k}$ denotes the elements in \mathcal{N}_k excluding k . In other words, at each iteration, we update $\boldsymbol{\theta}_k$ so that to move it toward the descent direction of the local cost and at the same time we constrain it to stay close to the convex combination of the rest of the updates, which are obtained during the computations in step 1 from *all* the nodes in its neighborhood. Thus, we end up with the following recursions:

Diffusion gradient descent

$$\text{Step 1: } \boldsymbol{\psi}_k^{(i)} = \boldsymbol{\theta}_k^{(i-1)} + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} (\mathbf{p}_m - \Sigma_{x_m} \boldsymbol{\theta}_k^{(i-1)}), \quad (5.90)$$

$$\text{Step 2: } \boldsymbol{\theta}_k^{(i)} = \sum_{m \in \mathcal{N}_k} a_{mk} \boldsymbol{\psi}_m^{(i)}, \quad (5.91)$$

where we set

$$a_{kk} = 1 - \mu_k \lambda \quad \text{and} \quad a_{mk} = \mu_k \lambda b_{mk}, \quad (5.92)$$

which leads to

$$\sum_{m \in \mathcal{N}_k} a_{mk} = 1, \quad a_{mk} \geq 0, \quad (5.93)$$

for small enough values of $\mu_k \lambda$. Note that by setting $a_{mk} = 0$, $m \notin \mathcal{N}_k$ and defining A to be the matrix with entries $[A]_{mk} = a_{mk}$, we can write

$$\sum_{m=1}^K a_{mk} = 1 \Rightarrow A^T \mathbf{1} = 1, \quad (5.94)$$

that is, A is a left stochastic matrix. It is important to stress here that, irrespective of our derivation before, *any* left stochastic matrix A in (5.91) can be used.

A slightly different path to arrive at (5.89) is via the interpretation of the gradient descent scheme as a minimizer of a regularized linearization of the cost function around the currently available estimate. The regularizer used is $\|\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)}\|^2$ and it tries to keep the new update as close as possible to the currently available estimate. In the context of the distributed learning, instead of $\boldsymbol{\theta}^{(i-1)}$ we can use a convex combination of the available estimates obtained in the neighborhood [84], [26, 27].

We are now ready to state the first version of the diffusion LMS, by replacing in (5.90) and (5.91) expectations with instantaneous observations and interpreting iterations as time updates.

Algorithm 5.7 (The adapt-then-combine diffusion LMS).

- Initialize
 - **For** $k = 1, 2, \dots, K$, **Do**
 - $\boldsymbol{\theta}_k^{(-1)} = \mathbf{0} \in \mathbb{R}^l$; or any other value.
 - **End For**
 - Select μ_k , $k = 1, 2, \dots, K$; a small positive number.
 - Select C : $C\mathbf{1} = \mathbf{1}$
 - Select A : $A^T \mathbf{1} = \mathbf{1}$

- **For** $n = 0, 1, \dots$, **Do**
 - **For** $k = 1, 2, \dots, K$, **Do**
 - **For** $m \in \mathcal{N}_k$, **Do**
 - $e_{k,m}(n) = y_m(n) - \theta_k^T(n-1)x_m(n)$; For complex-valued data, change $T \rightarrow H$.
 - **End Do**
 - $\psi_k(n) = \theta_k(n-1) + \mu_k \sum_{m \in \mathcal{N}_k} c_{mk} x_m(n) e_{k,m}(n)$; For complex-valued data, $e_{k,m}(n) \rightarrow e_{k,m}^*(n)$.
 - **End For**
 - **For** $k = 1, 2, \dots, K$
 - $\theta_k(n) = \sum_{m \in \mathcal{N}_k} a_{mk} \psi_m(n)$
 - **End For**
- **End For**

The following comments are in order:

- This form of diffusion LMS (DiLMS) is known as *Adapt-then-Combine DiLMS* (ATC) since the first step refers to the update and the combination step follows.
- In the special case of $C = I$, then the adaptation step becomes

$$\psi_k(n) = \theta_k(n-1) + \mu x_k(n) e_k(n),$$

and nodes *need not exchange* their observations/measurements.

- The adaptation rationale is illustrated in Figure 5.25. At time n , all three neighbors exchange the received data. In case the input vector corresponds to a realization of a random signal, $u_k(n)$, the exchange of information comprises two values $(y_k(n), u_k(n))$ in each direction for each one of the links. In the more general case, where the input is a random vector of jointly distributed variables, then all l variables have to be exchanged. After this message passing, adaptation takes place as shown in Figure 5.25a. Then, the nodes exchange their obtained estimates, $\psi_k(n)$, $k = 1, 2, 3$, across the links, 5.25b.

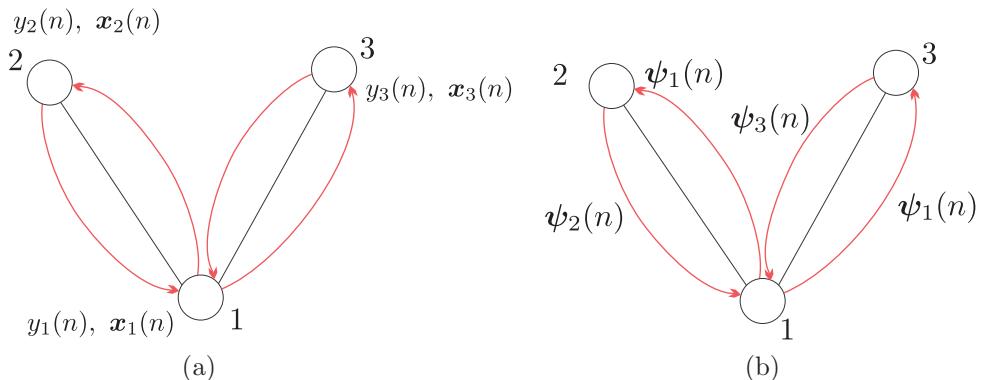


FIGURE 5.25

-
- (a) In step 1, adaptation is carried out after the exchange of the received observations.
 - (b) In step 2, the nodes exchange their locally computed estimates to obtain the updated one.

A different scheme results if one reverses the order of the two steps and performs first the combination and then the adaptation.

Algorithm 5.8 (The combine-then-adapt diffusion LMS).

- Initialization
 - **For** $k = 1, 2, \dots, K$, **Do**
 - $\theta_k(-1) = \mathbf{0} \in \mathbb{R}^l$; or any other value.
 - **End For**
 - Select C : $C\mathbf{1} = \mathbf{1}$
 - Select A : $A^T\mathbf{1} = \mathbf{1}$
 - Select μ_k , $k = 1, 2, \dots, K$; a small value.
- **For** $n = 0, 1, 2, \dots$, **Do**
 - **For** $k = 1, 2, \dots, K$, **Do**
 - $\psi_k(n-1) = \sum_{m \in \mathcal{N}_k} a_{mk} \theta_m(n-1)$
 - **End For**
 - **For** $k = 1, 2, \dots, K$, **Do**
 - **For** $m \in \mathcal{N}_k$, **Do**
 - $e_{k,m}(n) = y_m(n) - \psi_k^T(n-1)x_m(n)$; For complex-valued data, change $T \rightarrow H$.
 - **End For**
 - $\theta_k(n) = \psi_k(n-1) + \mu_k \sum_{m \in \mathcal{N}_k} x_m(n)e_{k,m}(n)$; For complex-valued data, $e_{k,m}(n) \rightarrow e_{k,m}^*(n)$.
 - **End For**
- **End For**

The rationale of this adaptation scheme is the reverse of that illustrated in [Figure 5.25](#), where the phase in [5.25b](#) precedes that of [5.25a](#). In case $C = I$, then there is no input-output data information exchange and the parameter update for the k node becomes

$$\theta_k(n) = \psi_k(n-1) + \mu_k x_k(n) e_k(n).$$

Remarks 5.4.

- One of the early reports on the diffusion LMS can be found in [59]. In [80, 93], versions of the algorithm for diminishing step sizes are presented and its convergence properties are analyzed. Besides the DiLMS, a version for incremental distributed cooperation has been proposed in [58]. For a related review, see [84, 86, 87].
- So far, nothing has been said about the choice of the matrices C (A). There are a number of possibilities. Two popular choices are:

Averaging Rule:

$$c_{mk} = \begin{cases} \frac{1}{n_k}, & \text{if } k = m, \text{ or if nodes } k \text{ and } m \text{ are neighbors,} \\ 0, & \text{otherwise,} \end{cases}$$

and the respective matrix is left stochastic.

Metropolis Rule:

$$c_{mk} = \begin{cases} \frac{1}{\max\{n_k, n_m\}}, & \text{if } k \neq m \text{ and } k, m \text{ are neighbors,} \\ 1 - \sum_{i \in \mathcal{N}_k \setminus k} c_{ik}, & m = k, \\ 0, & \text{otherwise,} \end{cases}$$

which makes the respective matrix to be doubly stochastic.

- Distributed LMS-based algorithms for the case where different nodes estimate different, yet overlapping, parameter vectors, have also been derived, [20, 75].

5.13.3 CONVERGENCE AND STEADY-STATE PERFORMANCE: SOME HIGHLIGHTS

In this subsection, we will summarize some findings concerning the performance analysis of the DiLMS. We will not give proofs. The proofs follow similar lines as for the standard LMS, with a slightly more involved algebra. The interested reader can obtain proofs by looking at the original papers as well as in [84].

- The gradient descent scheme in (5.90), (5.91) is guaranteed to converge, meaning

$$\boldsymbol{\theta}_k^{(i)} \xrightarrow[i \rightarrow \infty]{} \boldsymbol{\theta}_*,$$

provided that

$$\mu_k \leq \frac{2}{\lambda_{\max}\{\Sigma_k^{loc}\}},$$

where

$$\Sigma_k^{loc} = \sum_{m \in \mathcal{N}_k} c_{mk} \Sigma_{x_m}. \quad (5.95)$$

This corresponds to the condition in (5.15).

- If one assumes that C is doubly stochastic, it can be shown that the convergence rate to the solution for the distributed case is higher than that corresponding to the noncooperative scenario, when each node operates individually, using the same step size, $\mu_k = \mu$, for all cases and provided this common value guarantees convergence. In other words, cooperation *improves the convergence speed*. This is in line with the general comments made in the beginning of the section.
- Assume that in the model in (5.81), the involved noise sequences are both spatially and temporally white, as represented by

$$\mathbb{E}[\eta_k(n)\eta_k(n-r)] = \sigma_k^2 \delta_r,$$

$$\delta_r = \begin{cases} 1, & r = 0 \\ 0, & r \neq 0 \end{cases}$$

$$\mathbb{E}[\eta_k(n)\eta_m(r)] = \sigma_k^2 \delta_{km} \delta_{nr},$$

$$\delta_{km}, \delta_{nr} = \begin{cases} 1, & k = m, n = r \\ 0, & \text{otherwise.} \end{cases}$$

Also, the noise sequences are independent of the input vectors,

$$\mathbb{E}[\mathbf{x}_m(n)\eta_k(n-r)] = \mathbf{0}, \quad k, m = 1, 2, \dots, K, \forall r,$$

and finally, the independence assumption is mobilized among the input vectors, spatially as well as temporally, namely

$$\mathbb{E}[\mathbf{x}_k(n)\mathbf{x}_m^T(n-r)] = O, \text{ if } k \neq m, \text{ and } \forall r.$$

Under the previous assumptions, which correspond to the assumptions adopted when studying the performance of the LMS, the following hold true for the diffusion LMS.

Convergence in the mean: Provided that

$$\mu_k < \frac{2}{\lambda_{\max}\{\Sigma_k^{\text{loc}}\}}, \quad (5.96)$$

then

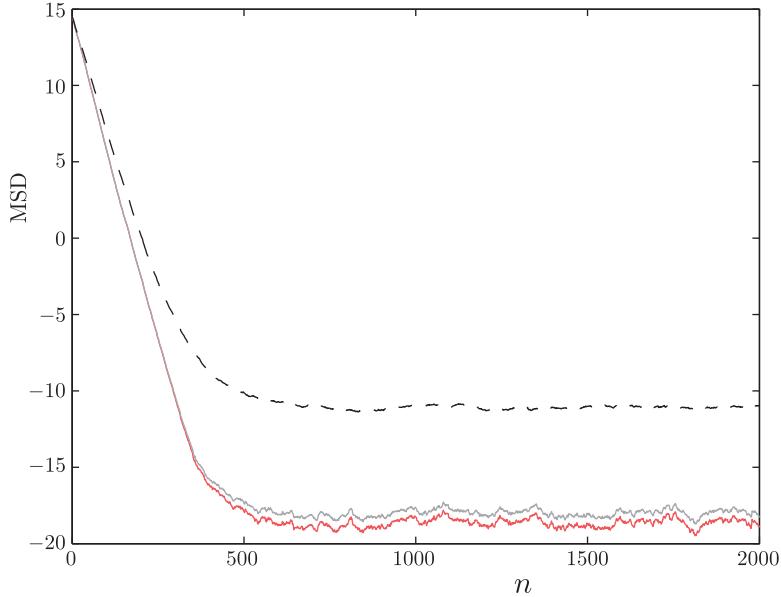
$$\mathbb{E}[\theta_k(n)] \xrightarrow{n \rightarrow \infty} \theta_*, \quad k = 1, 2, \dots, K.$$

It is important to state here that the stability condition in (5.96) depends on C and not on A .

- If in addition to the previous assumption, C is chosen to be doubly stochastic, then the convergence in the mean, in any node under the distributed scenario, is *faster* than that obtained if the node is operating individually without cooperation, provided $\mu_k = \mu$ is the same and it is chosen so as to guarantee convergence.
- *Misadjustment:* under the assumptions of C and A being doubly stochastic, the following are true:
 - The average misadjustment over all nodes in the steady-state for the adapt-then-combine strategy is always smaller than that of the combine-then-adapt one.
 - The average misadjustment over all the nodes of the network in the distributed operation is always lower than that obtained if nodes are adapted individually, without cooperation, by using the same $\mu_k = \mu$ in all cases. That is, cooperation does not only improve convergence speed but it also *improves the steady-state performance*.

Example 5.8. In this example, a network of $L = 10$ nodes is considered. The nodes were randomly connected with a total number of 32 connections; the resulting network was checked out that it was strongly connected. In each node, data are generated according to a regression model, using the same vector $\theta_o \in \mathbb{R}^{30}$. The latter was randomly generated by a $\mathcal{N}(0, 1)$. The input vectors, \mathbf{x}_k in (5.81), were i.i.d. generated according to a $\mathcal{N}(0, 1)$ and the noise level was different for each node, varying from 20 to 25 dBs.

Three experiments were carried out. The first involved the distributed LMS in its adapt-then-combine (ATC) form and the second one the combine-then-adapt (CTA) version. In the third experiment, the LMS algorithm was run independently for each node, without cooperation. In all cases, the step size was chosen equal to $\mu = 0.01$. Figure 5.26 shows the average (over all nodes) $\text{MSD}(n) : \frac{1}{K} \sum_{k=1}^K \|\theta_k(n) - \theta_o\|^2$ obtained for each one of the experiments. It is readily seen that cooperation improves the performance significantly, both in terms of convergence as well as in steady-state error floor. Moreover, as stated in Section 5.13.3, the ATC performs slightly better than the CTA version.

**FIGURE 5.26**

Average (over all the nodes) error convergence curves (MSD) for the LMS in noncooperative mode of operation (dotted line) and for the case of the diffusion LMS, in the ATC mode (red line) and the CTA mode (gray line). The step-size μ was the same in all three cases. Cooperation among nodes significantly improves performance. For the case of the diffusion LMS, the ATC version results in slightly better performance compared to that of the CTA.

5.13.4 CONSENSUS-BASED DISTRIBUTED SCHEMES

An alternative path for deriving an LMS version for distributed networks was followed in [64, 88]. Recall that, so far, in our discussion in deriving the DiLMS, we required the update at each node to be close to a convex combination of the available estimates in the respective neighborhood. Now we will demand such a requirement to become very strict. Although we are not going to get involved with details, since this would require to divert quite a lot from the material and the algorithmic tools which have been presented so far, let us state the task in the context of the linear MSE estimation.

To bring (5.83) in a distributed learning context, let us modify it by allowing different parameter vectors for each node, k , so that

$$J(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K) = \sum_{k=1}^K \mathbb{E} \left[|\mathbf{y}_k - \boldsymbol{\theta}_k^T \mathbf{x}_k|^2 \right].$$

Then, the task is cast according to the following constrained optimization problem,

$$\begin{aligned} \{\hat{\boldsymbol{\theta}}_k, k = 1, \dots, K\} &= \arg \min_{\{\boldsymbol{\theta}_k, k=1, \dots, K\}} J(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K) \\ \text{s.t.} \quad \boldsymbol{\theta}_k &= \boldsymbol{\theta}_m, \quad k = 1, 2, \dots, K, \quad m \in \mathcal{N}_k. \end{aligned}$$

In other words, one demands *equality* of the estimates within a neighborhood. As a consequence, these constraints lead to network-wise equality, since the graph that represents the network has been assumed to be connected. The optimization is carried out iteratively by employing stochastic approximation arguments and building on the alternating direction method of multipliers (ADMM) (Chapter 8), [19]. The algorithm, besides updating the vector estimates, has to update the associated Lagrange multipliers as well.

In addition to the previously reported ADMM-based scheme, a number of variants known as *consensus-based algorithms* have been employed in several studies [19, 33, 49, 50, 71]. A formulation around which a number of stochastic gradient consensus-based algorithms evolve is the following [33, 49, 50]:

$$\boldsymbol{\theta}_k(n) = \boldsymbol{\theta}_k(n-1) + \mu_k(n) \left[\mathbf{x}_k(n) e_k(n) + \lambda \sum_{m \in \mathcal{N}_k \setminus k} (\boldsymbol{\theta}_k(n-1) - \boldsymbol{\theta}_m(n-1)) \right], \quad (5.97)$$

where

$$e_k(n) := y_k(n) - \boldsymbol{\theta}_k^T(n-1) \mathbf{x}_k(n)$$

and for some $\lambda > 0$. Observe the form in (5.97); the term in the bracket on the right-hand side is a regularizer whose goal is to enforce equality among the estimates within the neighborhood of node k . Several alternatives to the formula (5.97) have been proposed. For example, in [49] a different step size is employed for the consensus summation on the right-hand side of (5.97). In [99], the following formulation is provided,

$$\boldsymbol{\theta}_k(n) = \boldsymbol{\theta}_k(n-1) + \mu_k(n) \left[\mathbf{x}_k(n) e_k(n) + \sum_{m \in \mathcal{N}_k \setminus k} b_{m,k} (\boldsymbol{\theta}_k(n-1) - \boldsymbol{\theta}_m(n-1)) \right], \quad (5.98)$$

where $b_{m,k}$ stands for some nonnegative coefficients. If one defines the weights,

$$a_{m,k} := \begin{cases} 1 - \sum_{m \in \mathcal{N}_k \setminus k} \mu_k(n) b_{m,k}, & m = k \\ \mu_k(n) b_{m,k}, & m \in \mathcal{N}_k \setminus k \\ 0, & \text{otherwise,} \end{cases} \quad (5.99)$$

recursion (5.98) can be equivalently written as,

$$\boxed{\boldsymbol{\theta}_k(n) = \sum_{m \in \mathcal{N}_k} a_{m,k} \boldsymbol{\theta}_m(n-1) + \mu_k(n) \mathbf{x}_k(n) e_k(n).} \quad (5.100)$$

The update rule in (5.100) is also referred to as *consensus strategy* (see, e.g., [99]). Note that the step-size is considered to be time-varying. In particular, in Refs. [19, 71], a diminishing step-size is employed, within the stochastic gradient rationale, which has to satisfy the familiar pair of conditions in order to guarantee convergence to a consensus value over all the nodes,

$$\sum_{n=0}^{\infty} \mu_k(n) = \infty, \quad \sum_{n=0}^{\infty} \mu_k^2(n) < \infty. \quad (5.101)$$

Observe the update recursion in (5.100). It is readily seen that the update $\boldsymbol{\theta}_k(n)$ involves only the error $e_k(n)$ of the corresponding node. In contrast, looking carefully at the corresponding update recursions

in both [Algorithms 5.7, 5.8](#), $\theta_k(n)$ is updated according to the average error within the neighborhood. This is an important difference.

The theoretical properties of the consensus recursion [\(5.100\)](#), which employs a *constant step-size*, as well as a comparative analysis against the diffusion schemes has been presented in [\[86, 99\]](#). There, it has been shown that the diffusion schemes outperform the consensus-based ones, in the sense that (a) they converge faster, (b) they reach lower steady-state mean-square deviation error floor, and (c) their mean-square stability is insensitive to the choice of the combination weights.

5.14 A CASE STUDY: TARGET LOCALIZATION

Consider a network consisting of K nodes, whose goal is to estimate and track the location of a specific target. The location of the unknown target, say θ_o , is assumed to belong to the two-dimensional space. The position of each node is denoted by $\theta_k = [\theta_{k1}, \theta_{k2}]^T$, and the true distance between node k and the unknown target equals to

$$r_k = \|\theta_o - \theta_k\|. \quad (5.102)$$

The vector, whose direction points from node k to the unknown source, is given by,

$$\mathbf{g}_k = \frac{\theta_o - \theta_k}{\|\theta_o - \theta_k\|}. \quad (5.103)$$

Obviously, the distance can be rewritten in terms of the direction vector as,

$$r_k = \mathbf{g}_k^T (\theta_o - \theta_k). \quad (5.104)$$

It is reasonable to assume that each node, k , “senses” the distance and the direction vectors via noisy observations. For example, such a noisy information can be inferred from the strength of the received signal or other related information. Following a similar rationale as in [\[84, 98\]](#), the noisy distance can be modeled as,

$$\hat{r}_k(n) = r_k + v_k(n), \quad (5.105)$$

where n stands for the discrete time instance and $v_k(n)$ for the additive noise term. The noise in the direction vector is a consequence of two effects: (a) a deviation occurring along the perpendicular direction to \mathbf{g}_k and (b) a deviation that takes place along the parallel direction of \mathbf{g}_k . All in one, the noisy direction vector (see [Figure 5.27](#)), occurring at time instance n , can be written as,

$$\hat{\mathbf{g}}_k(n) = \mathbf{g}_k + v_k^\perp(n)\mathbf{g}_k^\perp + v_k^\parallel(n)\mathbf{g}_k^\parallel, \quad (5.106)$$

where $v_k^\perp(n)$ is the noise corrupting the unit norm perpendicular direction vector \mathbf{g}_k^\perp and $v_k^\parallel(n)$ is the noise occurring at the parallel direction vector. Taking into consideration the noisy terms, [\(5.105\)](#) is written as

$$\hat{r}_k(n) = \hat{\mathbf{g}}_k^T(n)(\theta_o - \theta_k) + \eta_k(n), \quad (5.107)$$

where

$$\eta_k(n) = v_k(n) - v_k^\perp(n)\mathbf{g}_k^{1T}(\theta_o - \theta_k) - v_k^\parallel(n)\mathbf{g}_k^T(\theta_o - \theta_k). \quad (5.108)$$

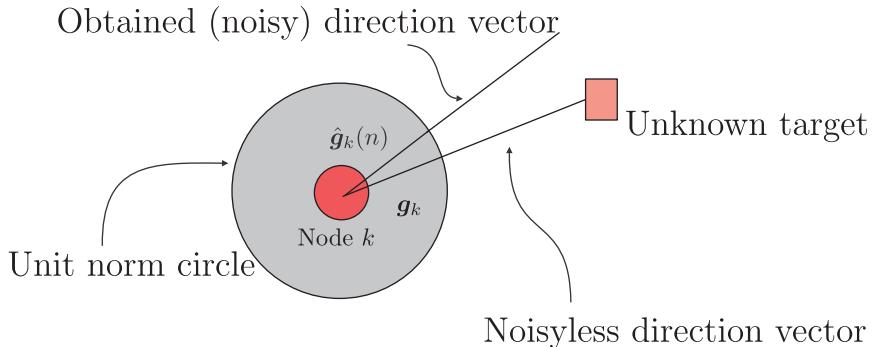
**FIGURE 5.27**

Illustration of a node, the target source, and the direction vectors.

Equation (5.108) can be further simplified if one recalls that by construction $\mathbf{g}_k^{\perp T}(\boldsymbol{\theta}_o - \boldsymbol{\theta}_k) = 0$. Moreover, typically, the contribution of $v_k^{\perp}(n)$ is assumed to be significantly larger than the contribution of $v_k^{\parallel}(n)$. Henceforth, taking into consideration these two arguments, (5.108) can be simplified to

$$\eta_k(n) \approx v_k(n). \quad (5.109)$$

If one defines $y_k(n) := \hat{r}_k(n) + \hat{\mathbf{g}}_k^T(n)\boldsymbol{\theta}_k$ and combines (5.107) with (5.109) the following model results:

$$y_k(n) \approx \boldsymbol{\theta}_o^T \hat{\mathbf{g}}_k(n) + v_k(n). \quad (5.110)$$

Equation (5.110) is a linear regression model. Using the available estimates, for each time instant, one has access to $y_k(n)$, $\hat{\mathbf{g}}_k(n)$ and any form of distributed algorithm can be adopted in order to obtain a better estimate of $\boldsymbol{\theta}_o$.

Indeed, it has been verified that the information exchange and fusion enhances significantly the ability of the nodes to estimate and track the target source. The nodes can possibly represent fish schools, which seek a nutrition source, bee swarms, which search for their hive, or bacteria seeking nutritive sources [28, 84, 85, 97].

Some other typical applications of distributed learning are social networks [36], radio resource allocation [32], and network cartography [65].

5.15 SOME CONCLUDING REMARKS: CONSENSUS MATRIX

In our treatment of the diffusion LMS, we used the combination matrices A (C), which we assumed to be left (right) stochastic. Also, in the performance-related section, we pointed out that some of the reported results hold true if these matrices are, in addition, doubly stochastic. Although it was not needed in this chapter, in the general distributed processing theory, a matrix of significant importance is the so-called *consensus matrix*. A matrix $A \in \mathbb{R}^{K \times K}$ is said to be a consensus matrix, if in addition to being doubly stochastic, as represented by

$$A\mathbf{1} = \mathbf{1}, \quad A^T\mathbf{1} = \mathbf{1},$$

it also satisfies the property,

$$\left| \lambda_i \{A^T - \frac{1}{K} \mathbf{1}\mathbf{1}^T\} \right| < 1, \quad i = 1, 2, \dots, K.$$

In other words, all eigenvalues of the matrix

$$A^T - \frac{1}{K} \mathbf{1}\mathbf{1}^T$$

have magnitude strictly less than one. To demonstrate its usefulness, we will state a fundamental theorem in distributed learning.

Theorem 5.2. *Consider a network consisting of K nodes, and each one of them having access to a state vector \mathbf{x}_k . Consider the recursion,*

$$\boldsymbol{\theta}_k^{(i)} = \sum_{m \in \mathcal{N}_k} a_{mk} \boldsymbol{\theta}_m^{(i-1)}, \quad k = 1, 2, \dots, K, \quad i > 0 : \text{Consensus Iteration},$$

with

$$\boldsymbol{\theta}_k^{(0)} = \mathbf{x}_k, \quad k = 1, 2, \dots, K.$$

Define $A \in \mathbb{R}^{K \times K}$, to be the matrix with entries

$$[A]_{mk} = a_{mk}, \quad m, k = 1, 2, \dots, K,$$

where $a_{mk} \geq 0$, $a_{mk} = 0$ if $m \notin \mathcal{N}_k$. If A is a consensus matrix, then [31],

$$\boldsymbol{\theta}_k^{(i)} \longrightarrow \frac{1}{K} \sum_{k=1}^K \mathbf{x}_k.$$

The opposite is also true. If convergence is always guaranteed, then A is a consensus matrix.

In other words, this theorem states that updating each node by convexly combining, with appropriate weights, the current estimates in its neighborhood, then the network converges to the average value in a consensus rationale (Problem 5.17).

PROBLEMS

- 5.1 Show that the gradient vector is perpendicular to the tangent at a point of an isovalue curve.
 5.2 Prove that if

$$\sum_{i=1}^{\infty} \mu_i^2 < \infty, \quad \sum_{i=1}^{\infty} \mu_i = \infty,$$

the steepest descent scheme, for the MSE loss function and for the iteration-dependent step size case, converges to the optimal solution.

- 5.3 Derive the steepest gradient descent direction for the complex-valued case.
 5.4 Let θ , \mathbf{x} be two jointly distributed random variables. Let also the function (regressor)

$$f(\theta) = \mathbb{E}[\mathbf{x}|\theta].$$

Show that under the conditions in (5.28), the recursion

$$\theta_n = \theta_{n-1} - \mu_n x_n$$

converges in probability to a root of $f(\theta)$.

5.5 Show that the LMS algorithm is a nonlinear estimator.

5.6 Show Eq. (5.41).

5.7 Derive the bound in (5.44).

Hint. Use the well-known property from linear algebra, that the eigenvalues of a matrix, $A \in \mathbb{R}^{l \times l}$, satisfy the following bound,

$$\max_{1 \leq i \leq l} |\lambda_i| \leq \max_{1 \leq i \leq l} \sum_{j=1}^l |a_{ij}| := \|A\|_1.$$

5.8 *Gershgorin circle theorem.* Let A be an $l \times l$ matrix, with entries a_{ij} , $i, j = 1, 2, \dots, l$. Let $R_i := \sum_{\substack{j=1 \\ j \neq i}}^l |a_{ij}|$, be the sum of absolute values of the nondiagonal entries in row i . Then show that if λ is an eigenvalue of A , then there exists at least one row i , such that the following is true,

$$|\lambda - a_{ii}| \leq R_i.$$

The last bound defines a circle, which contains the eigenvalue λ .

5.9 Apply the Gershgorin circle theorem to prove the bound in (5.44).

5.10 Derive the misadjustment formula given in (5.51).

5.11 Derive the APA iteration scheme.

5.12 Given a value x , define the hyperplane comprising all values of θ such as

$$x^T \theta - y = 0.$$

Then x is perpendicular to the hyperplane.

5.13 Derive the recursions for the widely linear APA.

5.14 Show that a similarity transformation of a square matrix, via a unitary matrix does not affect the eigenvalues.

5.15 Show that if $x \in \mathbb{R}^l$ is a Gaussian random vector, then

$$F := \mathbb{E}[xx^T S xx^T] = \Sigma_x \text{trace}\{S\Sigma_x\} + 2\Sigma_x S \Sigma_x,$$

and if $x \in \mathbb{C}^l$,

$$F := \mathbb{E}[xx^H S xx^H] = \Sigma_x \text{trace}\{S\Sigma_x\} + \Sigma_x S \Sigma_x.$$

5.16 Show that if a $l \times l$ matrix C is right stochastic, then all its eigenvalues satisfy

$$|\lambda_i| \leq 1, \quad i = 1, 2, \dots, l.$$

The same holds true for left and doubly stochastic matrices.

5.17 Prove Theorem 5.2.

MATLAB Exercises

- 5.18** Consider the MSE cost function in (5.4). Set the cross-correlation equal to $\mathbf{p} = [0.05, 0.03]^T$. Also, consider two covariance matrices,

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Compute the corresponding optimal solutions, $\boldsymbol{\theta}_{(*,1)} = \Sigma_1^{-1}\mathbf{p}$, $\boldsymbol{\theta}_{(*,2)} = \Sigma_2^{-1}\mathbf{p}$. Apply the gradient descent scheme of (5.6) to estimate $\boldsymbol{\theta}_{(*,2)}$; set the step-size equal to (a) its optimal value μ_o according to (5.16) and (b) equal to $\mu_o/2$. For these two choices for the step-size, plot the error $\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_{(*,2)}\|^2$, at each iteration step i . Compare the convergence speeds of these two curves towards zero. Moreover, in the two-dimensional space, plot the coefficients of the successive estimates, $\boldsymbol{\theta}^{(i)}$, for both step-sizes, together with the isovalue contours of the cost function. What do you observe regarding the trajectory towards the minimum?

Apply (5.6) for the estimation of $\boldsymbol{\theta}_{(*,1)}$ employing Σ_1^{-1} and \mathbf{p} . Use as step size μ_o of the previous experiment. Plot, in the same figure, the previously computed error curve $\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_{(*,2)}\|^2$ together with the error curve $\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_{(*,1)}\|^2$. Compare the convergence speeds. Set now the step size equal to the optimum value associated with Σ_1 . Again, in the two-dimensional space, plot the values of the successive estimates and the isovalue contours of the cost function. Compare the number of steps needed for convergence, with the ones needed in the previous experiment. Play with different covariance matrices and step sizes.

- 5.19** Consider the linear regression model

$$y_n = \mathbf{x}_n^T \boldsymbol{\theta}_o + \eta_n,$$

where $\boldsymbol{\theta}_o \in \mathbb{R}^2$. Generate the coefficients of the unknown vector, $\boldsymbol{\theta}_o$, randomly according to the normalized Gaussian distribution, $\mathcal{N}(0, 1)$. The noise is assumed to be white Gaussian with variance 0.1. The samples of the input vector are i.i.d. generated via the normalized Gaussian. Apply the Robbins-Monro algorithm in (5.33), for the optimal MSE linear estimation, with a step size equal to $\mu_n = 1/n$. Run 1000 independent experiments and plot the mean value of the first coefficient of the 1000 produced estimates, at each iteration step. Also, plot the horizontal line crossing the true value of the first coefficient of the unknown vector. Furthermore, plot the standard deviation of the obtained estimate, every 30 iteration steps. Comment on the results. Play with different rules of diminishing step-sizes and comment on the results.

- 5.20** Generate data according to the regression model

$$y_n = \mathbf{x}_n^T \boldsymbol{\theta}_o + \eta_n,$$

where $\boldsymbol{\theta}_o \in \mathbb{R}^{10}$, and whose elements are randomly obtained using the Gaussian distribution $\mathcal{N}(0, 1)$. The noise samples are also i.i.d. generated from $\mathcal{N}(0, 0.01)$.

Generate the inputs samples as part of two processes: (a) a white noise sequence, i.i.d. generated via $\mathcal{N}(0, 1)$ and (b) an autoregressive AR(1) process with $a_1 = 0.85$ and the corresponding white noise excitation is of variance equal to 1. For these two choices of the input, run the LMS algorithm, on the generated training set (y_n, \mathbf{x}_n) , $n = 0, 1, \dots$, to estimate $\boldsymbol{\theta}_o$. Use a step size equal to $\mu = 0.01$. Run 100 independent experiments and plot the average error per iteration in dBs, using $10 \log_{10}(e_n^2)$, with $e_n^2 = (y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n)^2$. What do you observe regarding the convergence speed of the algorithm for the two cases? Repeat the experiment with different values of the AR coefficient a_1 and different values of the step-size. Observe how

the learning curve changes with the different values of the step-size and/or the value of the AR coefficient. Choose, also, a relatively large value for the step-size and make the LMS algorithm to diverge. Comment and justify theoretically the obtained results concerning convergence speed and the error floor at the steady-state after convergence.

- 5.21** Use the data set generated from the AR(1) process of the previous exercise. Employ the transform-domain LMS ([Algorithm 5.6](#)) with step-size equal to 0.01. Also, set $\delta = 0.01$ and $\beta = 0.5$. Moreover, employ the DCT transform. As in the previous exercise, run 100 independent experiments and plot the average error per iteration. Compare the results with that of the LMS with the same step-size.

Hint. Compute the DCT transformation matrix using the *dctmtx* MATLAB function.

- 5.22** Generate the same experimental setup, as in [Exercise 5.20](#), with the difference that $\theta_o \in \mathbb{R}^{60}$. For the LMS algorithm set $\mu = 0.025$ and for the NLMS ([Algorithm 5.3](#)) $\mu = 0.35$ and $\delta = 0.001$. Employ also the APA ([Algorithm 5.2](#)) algorithm with parameters $\mu = 0.1$, $\delta = 0.001$ and $q = 10, 30$. Plot in the same figure the error learning curves of all these algorithms, as in the previous exercises. How does the choice of q affect the behavior of the APA algorithm, both in terms of convergence speed as well as the error floor at which it settles after convergence? Play with different values of q and of the step-size μ .

- 5.23** Consider the decision feedback equalizer described in [Section 5.10](#).
- (a) Generate a set of 1000 random ± 1 values (BPSK) (i.e., s_n). Direct this sequence into a linear channel with impulse response $\mathbf{h} = [0.04, -0.05, 0.07, -0.21, 0.72, 0.36, 0.21, 0.03, 0.07]^T$ and add to the output 11dB white Gaussian noise. Denote the output as u_n .
 - (b) Design the adaptive decision feedback equalizer (DFE) using $L = 21$, $l = 10$, and $\mu = 0.025$ following the training mode only. Perform a set of 500 experiments feeding the DFE with different random sequences from the ones described in [step 5.23a](#). Plot the mean-square error (averaged over the 500 experiments). Observe that around $n = 250$ the algorithm achieves convergence.
 - (c) Design the adaptive decision feedback equalizer using the parameters of [step 5.23b](#). Feed the equalizer with a series of 10,000 random values generated as in [step 5.23a](#). After the 250th data sample, change the DFE to decision-directed mode. Count the percentage of the errors performed by the equalizer from the 251th to the 10,000th sample.
 - (d) Repeat [steps 5.23a](#) to [5.23c](#) changing the level of the white Gaussian noise added to the BPSK values to 15, 12, 10 dBs. Then, for each case, change the delay to $L = 5$. Comment on the results.
- 5.24** Develop the MATLAB code for the two forms of the diffusion LMS, adapt-then-combine (ATC) and combine-then-adapt (CTA) and reproduce the results of [Example 5.8](#). Play with the choice of the various parameters. Make sure that the resulting network is strongly connected.

REFERENCES

- [1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, A survey on sensor networks, *IEEE Commun. Mag.* 40(8) (2002) 102-114.
- [2] S.J.M. Almeida, J.C.M. Bermudez, N.J. Bershad, M.H. Costa, A statistical analysis of the affine projection algorithm for unity step size and autoregressive inputs, *IEEE Trans. Circuits Syst. I* 52(7) (2005) 1394-1405.
- [3] T.Y. Al-Naffouri, A.H. Sayed, Transient analysis of data-normalized adaptive filters, *IEEE Trans. Signal Process.* 51(3) (2003) 639-652.

- [4] S. Amari, Theory of adaptive pattern classifiers, *IEEE Trans. Electron. Comput.* 16(3) (1967) 299-307.
- [5] C. Antweiler, M. Dörbecker, Perfect sequence excitation of the NLMS algorithm and its application to acoustic echo control, *Ann. Telecommun.* 49(7-8) (1994) 386-397.
- [6] J.A. Appolinario, S. Werner, P.S.R. Diniz, T.I. Laakso, Constrained normalized adaptive filtering for CDMA mobile communications, in: *Proceedings, EUSIPCO*, Rhodes, Greece, 1998.
- [7] J.A. Appolinario, M.L.R. de Campos, P.S.R. Diniz, The binormalized data-reusing LMS algorithm, *IEEE Trans. Signal Process.* 48 (2000) 3235-3242.
- [8] J. Arenas-Garcia, A.R. Figueiras-Vidal, A.H. Sayed, Mean-square performance of a convex combination of two adaptive filters, *IEEE Trans. Signal Process.* 54(3) (2006) 1078-1090.
- [9] S. Barbarossa, G. Scutari, Bio-inspired sensor network design: Distributed decisions through self-synchronization, *IEEE Signal Process. Mag.* 24(3) (2007) 26-35.
- [10] J. Benesty, T. Gänslер, D.R. Morgan, M.M. Sondhi, S.L. Gay, *Advances in Network and Acoustic Echo Cancellation*, Springer Verlag, Berlin, 2001.
- [11] J. Benesty, S.L. Gay, An improved PNLMS algorithm, in: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, vol. 2, 2002.
- [12] J. Benesty, C. Paleologu, S. Ciocchina, On regularization in adaptive filtering, *IEEE Trans. Audio Speech Lang. Process.* 19(6) (2011) 1734-1742.
- [13] A. Benveniste, M. Metivier, P. Piouret, *Adaptive Algorithms and Stochastic Approximations*, Springer-Verlag, NY, 1987.
- [14] K. Berberidis, P. Karaivazoglou, An efficient block adaptive DFE implemented in the frequency domain, *IEEE Trans. Signal Proc.* 50 (9) (2002) 2273-2286.
- [15] N.J. Bershad, Analysis of the normalized LMS with Gaussian inputs, *IEEE Trans. Acoust. Speech Signal Process.* 34(4) (1986) 793-806.
- [16] N.J. Bershad, O.M. Macchi, Adaptive recovery of a chirped sinusoid in noise. Part 2: Performance of the LMS algorithm, *IEEE Trans. Signal Process.* 39 (1991) 595-602.
- [17] J.C.M. Bermudez, N.J. Bershad, A nonlinear analytical model for the quantized LMS algorithm: The arbitrary step size case, *IEEE Trans. Signal Process.* 44 (1996) 1175-1183.
- [18] A. Bertrand, M. Moonen, Seeing the bigger picture, *IEEE Signal Process. Mag.* 30(3) (2013) 71-82.
- [19] D.P. Bertsekas, J.N. Tsitsiklis, *Parallel and Distributed Computations: Numerical Methods*, Athena Scientific, Belmont, MA, 1997.
- [20] N. Bogdanovic, J. Plata-Chaves, K. Berberidis, Distributed incremental-based LMS for node-specific adaptive parameter estimation, *IEEE Trans. Signal Proc.* 62 (20) (2014) 5382-5397.
- [21] N. Cesa-Bianchi, P.M. Long, M.K. Warmuth, Worst case quadratic loss bounds for prediction using linear functions and gradient descent, *IEEE Trans. Neural Networks* 7(3) (1996) 604-619.
- [22] P. Boulopoulis, S. Theodoridis, Extension of Wirtinger's Calculus to Reproducing Kernel Hilbert Spaces and the Complex Kernel LMS, *IEEE Trans. Signal Process.* 53(3) (2011) 964-978.
- [23] S. Boyd, L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
- [24] A. Carini, Efficient NLMS and RLS Algorithms for perfect and imperfect periodic sequences, *IEEE Trans. Signal Process.* 58(4) (2010) 2048-2059.
- [25] A. Carini, G.L. Sicuranza, V.J. Mathews, Efficient adaptive identification of linear-in-the-parameters nonlinear filters using periodic input sequences, *Signal Process.* 93(5) (2013) 1210-1220.
- [26] F.S. Cattivelli, A.H. Sayed, Diffusion LMS strategies for distributed estimation, *IEEE Trans. Signal Process.* 58(3) (2010) 1035-1048.
- [27] F.S. Cattivelli, *Distributed Collaborative Processing over Adaptive Networks*, Ph.D. Thesis, University of California, LA, 2010.
- [28] J. Chen, A.H. Sayed, Bio-inspired cooperative optimization with application to bacteria mobility, in: *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 2011, pp. 5788-5791.
- [29] S. Chouvardas, Y. Kopsinis, S. Theodoridis, Sparsity-aware distributed learning, in: S. Cui, A. Hero, J. Moura, Z.Q. Luo (Eds.), *Big Data over Networks*, Cambridge University Press, 2014.

- [30] T.A.C.M. Claasen, W.F.G. Mecklenbrauker, Comparison of the convergence of two algorithms for adaptive FIR digital filters, *IEEE Trans. Acoust. Speech Signal Process.* 29 (1981) 670-678.
- [31] M.H. DeGroot, Reaching a consensus, *J. Amer. Stat. Assoc.* 69(345) (1974) 118-121.
- [32] P. Di Lorenzo, S. Barbarossa, Swarming algorithms for distributed radio resource allocation, *IEEE Signal Process. Mag.* 30(3) (2013) 144-154.
- [33] A.G. Dimakis, S. Kar, J.M.F. Moura, M.G. Rabbat, A. Scaglione, Gossip algorithms for distributed signal processing, *Proc. IEEE* 98(11) (2010) 1847-1864.
- [34] P.S.R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*, fourth ed., Springer, 2013.
- [35] D.L. Duttweiler, Proportionate NLMS adaptation in echo cancelers, *IEEE Trans. Audio Speech Lang. Process.* 8 (2000) 508-518.
- [36] C. Chamley, A. Scaglione, L. Li, Models for the diffusion of belief in social networks, *IEEE Signal Process. Mag.* 30(3) (2013) 16-28.
- [37] C. Eksin, P. Molavi, A. Ribeiro, A. Jadbabaie, Learning in network games with incomplete information, *IEEE Signal Process. Mag.* 30(3) (2013) 30-42.
- [38] D.C. Farden, Tracking properties of adaptive signal processing algorithms, *IEEE Trans. Acoust. Speech Signal Process.* 29 (1981) 439-446.
- [39] E.R. Ferrara, Fast implementations of LMS adaptive filters, *IEEE Trans. Acoust. Speech Signal Process.* 28 (1980) 474-475.
- [40] O.L. Frost III, An algorithm for linearly constrained adaptive array processing, *Proc. IEEE* 60 (1972) 962-935.
- [41] I. Furukawa, A design of canceller of broadband acoustic echo, in: *Proceedings, International Teleconference Symposium*, 1984, pp. 1-8.
- [42] S.L. Gay, S. Tavathia, The fast affine projection algorithm, in: *Proceedings International Conference on Acoustics, Speech and Signal Processing, ICASSP*, 1995, pp. 3023-3026.
- [43] S.L. Gay, J. Benesty, *Acoustical Signal Processing for Telecommunications*, Kluwer, 2000.
- [44] A. Gershho, Adaptive equalization of highly dispersive channels for data transmission, *Bell Syst. Tech. J.* 48 (1969) 55-70.
- [45] A. Gilloire, M. Vetterli, Adaptive filtering in subbands with critical sampling: analysis, experiments and applications to acoustic echo cancellation, *IEEE Trans. Signal Process.* 40 (1992) 1862-1875.
- [46] B. Hassibi, A.H. Sayed, T. Kailath, H^∞ optimality of the LMS algorithm, *IEEE Trans. Signal Process.* 44(2) (1996) 267-280.
- [47] S. Haykin, *Adaptive Filter Theory*, fourth ed., Prentice Hall, 2002.
- [48] T. Hinamoto, S. Maekawa, Extended theory of learning identification, *IEEE Trans.* 95(10) (1975) 227-234 (in Japanese).
- [49] S. Kar, J. Moura, Convergence rate analysis of distributed gossip (linear parameter) estimation: fundamental limits and tradeoffs, *IEEE J Select. Topics Signal Process.* 5(4) (2011) 674-690.
- [50] S. Kar, J. Moura, K. Ramanan, Distributed parameter estimation in sensor networks: nonlinear observation models and imperfect communication, *IEEE Trans. Informat. Theory* 58(6) (2012) 3575-3605.
- [51] S. Kar, J.M.F. Moura, Consensus + innovations distributed inference over networks, *IEEE Signal Process. Mag.* 30(3) (2013) 99-109.
- [52] R.M. Karp, Reducibility among combinational problems, in: R.E. Miller, J.W. Thatcher (Eds.), *Complexity of Computer Computations*, Plenum Press, NY, 1972, pp. 85-104.
- [53] W. Kellermann, Kompensation akustischer echos in frequenzteilbandern, *Aachener Kolloquium*, Aachen, Germany, 1984, pp. 322-325.
- [54] W. Kellermann, Analysis and design of multirate systems for cancellation of acoustical echos, in: *Proceedings, IEEE International Conference on Acoustics, Speech and Signal Processing*, New York, 1988, pp. 2570-2573.
- [55] J. Kivinen, M.K. Warmuth, B. Hassibi, The p -norm generalization of the LMS algorithms for filtering, *IEEE Trans. Signal Process.* 54(3) (2006) 1782-1793.

- [56] H.J. Kushner, G.G. Yin, Stochastic Approximation Algorithms and Applications, Springer, New York, 1997.
- [57] L. Ljung, System Identification: Theory for the User, Prentice Hall, Englewood Cliffs, NJ, 1987.
- [58] C.G. Lopes, A.H. Sayed, Incremental adaptive strategies over distributed networks, IEEE Trans. Signal Process. 55(8) (2007) 4064-4077.
- [59] C.G. Lopes, A.H. Sayed, Diffusion least-mean-squares over adaptive networks: Formulation and performance analysis, IEEE Transactions on Signal Processing 56(7) (2008) 3122-3136.
- [60] C. Lopes, A.H. Sayed, Diffusion adaptive networks with changing topologies, in: Proceedings International Conference on Acoustics, Speech and Signal processing, CASSP, Las Vegas, April 2008, pp. 3285-3288.
- [61] O.M. Macci, N.J. Bershad, Adaptive recovery of chirped sinusoid in noise. Part 1: Performance of the RLS algorithm, IEEE Trans. Signal Process. 39 (1991) 583-594.
- [62] O. Macchi, Adaptive Processing: The Least-Mean-Squares Approach with Applications in Transmission, Wiley, New York, 1995.
- [63] V.J. Mathews, S.H. Cho, Improved convergence analysis of stochastic gradient adaptive filters using the sign algorithm, IEEE Trans. Acoust. Speech Signal Process. 35 (1987) 450-454.
- [64] G. Mateos, I.D. Schizas, G.B. Giannakis, Performance analysis of the consensus-based distributed LMS algorithm, EURASIP J. Adv. Signal Process. Article: ID981030, doi: 10.1155/2009/981030, 2009.
- [65] G. Mateos, K. Rajawat, Dynamic network cartography, IEEE Signal Process. Mag. 30(3) (2013) 129-143.
- [66] R. Merched, A. Sayed, An embedding approach to frequency-domain and subband filtering, IEEE Trans. Signal Process. 48(9) (2000) 2607-2619.
- [67] N. Murata, A statistical study on online learning, in: D. Saad (Ed.), Online Learning and Neural Networks, Cambridge University Press, UK, 1998, pp. 63-92.
- [68] S.S. Narayan, A.M. Peterson, Frequency domain LMS algorithm, Proc. IEEE 69(1) (1981) 124-126.
- [69] V.H. Nascimento, J.C.M. Bermudez, Probability of divergence for the least mean fourth algorithm, IEEE Trans. Signal Process. 54 (2006) 1376-1385.
- [70] V.H. Nascimento, M.T.M. Silva, Adaptive filters, in: R. Chellappa, S. Theodoridis (Eds.), Signal Process, E-Ref. 1 (2014) 619-747.
- [71] A. Nedic, A. Ozdaglar, Distributed subgradient methods for multi-agent optimization, IEEE Trans. Automat. Control 54(1) (2009) 48-61.
- [72] K. Ozeki, T. Umeda, An adaptive filtering algorithm using an orthogonal projection to an affine subspace and its properties, IEICE Trans. 67-A(5) (1984) 126-132 (in Japanese).
- [73] C. Paleologu, J. Benesty, S. Ciochina, Regularization of the affine projection algorithm, IEEE Trans. Circuits Syst. II: Express Briefs 58(6) (2011) 366-370.
- [74] A. Papoulis, S.U. Pillai, Probability, Random Variables and Stochastic Processes, fourth ed., McGraw Hill, 2002.
- [75] J. Plata-Chaves, N. Bogdanovic, K. Berberidis, Distributed diffusion-based LMS for node-specific adaptive parameter estimation, *arXiv:1408.3354* (2014). <http://arxiv.org/abs/1408.3354>.
- [76] J.B. Predd, S.R. Kulkarni, H.V. Poor, Distributed learning in wireless sensor networks, IEEE Signal Process. Mag. 23(4) (2006) 56-69.
- [77] J. Proakis, Digital Communications, fourth ed., McGraw Hill, New York, 2000.
- [78] J. Proakis, J.H. Miller, Adaptive receiver for digital signalling through channels with intersymbol interference, IEEE Trans. Informat. Theory 15 (1969) 484-497.
- [79] H. Robbins, S. Monro, A stochastic approximation method, Ann. Math. Stat. 22 (1951) 400-407.
- [80] S.S. Ram, A. Nedich, V.V. Veeravalli, Distributed stochastic subgradient projection algorithms for convex optimization, J. Optim. Theory Appl. 147(3) (2010) 516-545.
- [81] M. Martinez-Ramon, J. Arenas-Garcia, A. Navia-Vazquez, A.R. Figueiras-Vidal, An adaptive combination of adaptive filters for plant identification, in: Proceedings the 14th International Conference on Digital Signal Processing (DSP), 2002, pp. 1195-1198.

- [82] A.H. Sayed, M. Rupp, Error energy bounds for adaptive gradient algorithms, *IEEE Trans. Signal Process.* 44(8) (1996) 1982-1989.
- [83] A.H. Sayed, *Fundamentals of Adaptive Filtering*, John Wiley, 2003.
- [84] A.H. Sayed, Diffusion adaptation over networks, in: R. Chellappa, S. Theodoridis (Eds.), *Academic Press Library in Signal Processing*, Vol. 3, Academic Press, 2014, pp. 323-454
- [85] A.H. Sayed, S.-Y. Tu, X. Zhao, Z.J. Towfic, Diffusion strategies for adaptation and learning over networks, *IEEE Signal Process. Mag.* 30(3) (2013) 155-171.
- [86] A.H. Sayed, Adaptive networks, *Proc. IEEE* 102(4) (2014) 460-497.
- [87] A.H. Sayed, Adaptation, learning, and optimization over networks, *Foundat. Trends Machine Learn.* 7(4-5) (2014) 311-801.
- [88] I.D. Schizas, G. Mateos, G.B. Giannakis, Distributed LMS for consensus-based in-network adaptive processing, *IEEE Trans. Signal Process.* 57(6) (2009) 2365-2382.
- [89] D.I. Shuman, S.K. Narang, A. Ortega, P. Vandergheyrst, The emerging field of signal processing on graphs, *IEEE Signal Process. Mag.* 30(3) (2013) 83-98.
- [90] D.T. Slock, On the convergence behavior of the LMS and normalized LMS algorithms, *IEEE Trans. Signal Process.* 40(9) (1993) 2811-2825.
- [91] V. Solo, X. Kong, *Adaptive Signal Processing Algorithms: Stability and Performance*, Prentice Hall, Upper Saddle River, NJ, 1995.
- [92] V. Solo, The stability of LMS, *IEEE Trans. Signal Process.* 45(12) (1997) 3017-3026.
- [93] S.S. Stankovic, M.S. Stankovic, D.M. Stipanovic, Decentralized parameter estimation by consensus based stochastic approximation, *IEEE Trans. Automat. Control* 56(3) (2011) 531-543.
- [94] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, fourth ed., Academic Press, 2009.
- [95] J.N. Tsitsiklis, Problems in Decentralized Decision Making and Computation, Ph.D. Thesis, MIT, 1984.
- [96] Y.Z. Tsypkin, *Adaptation and Learning in Automatic Systems*, Academic Press, New York, 1971.
- [97] S.-Y. Tu, A.H. Sayed, Foraging behavior of fish schools via diffusion adaptation, in: *Proceedings Cognitive Information Processing*, CIP, 2010, pp. 63-68.
- [98] S.-Y. Tu, A.H. Sayed, Mobile adaptive networks, *IEEE J. Selected Topics Signal Process.* 5(4) (2011) 649-664.
- [99] S.-Y. Tu, A.H. Sayed, Diffusion strategies outperform consensus strategies for distributed estimation over adaptive networks, *IEEE Trans. Signal Process.* 60(12) (2012) 6217-6234.
- [100] K. Vikram, V.H. Poor, Social learning and Bayesian games in multiagent signal processing, *IEEE Signal Process. Mag.* 30(3) (2013) 43-57.
- [101] E. Walach, B. Widrow, The least mean fourth (LMF) adaptive algorithm and its family, *IEEE Trans. Informat. Theory* 30(2) (1984) 275-283.
- [102] B. Widrow, M.E. Hoff, Adaptive switching circuits, *IRE Part 4, IRE WESCON Convention Record*, 1960, pp. 96-104.
- [103] B. Widrow, S.D. Stearns, *Adaptive Signal Processing*, Prentice Hall, Englewood Cliffs, 1985.
- [104] J.-J. Xiao, A. Ribeiro, Z.-Q. Luo, G.B. Giannakis, Distributed compression-estimation using wireless networks, *IEEE Signal Process. Mag.* 23 (4) (2006) 27-41.
- [105] N.R. Yousef, A.H. Sayed, A unified approach to the steady-state and tracking analysis of adaptive filters, *IEEE Trans. Signal Process.* 49(2) (2001) 314-324.
- [106] N.R. Yousef, A.H. Sayed, Ability of adaptive filters to track carrier offsets and random channel nonstationarities, *IEEE Trans. Signal Process.* 50(7) (2002) 1533-1544.
- [107] F. Zhao, J. Lin, L. Guibas, J. Reich, Collaborative signal and information processing, *Proc. IEEE* 91(8) (2003) 1199-1209.

This page intentionally left blank

THE LEAST-SQUARES FAMILY

6

CHAPTER OUTLINE

6.1	Introduction	234
6.2	Least-Squares Linear Regression: A Geometric Perspective	234
6.3	Statistical Properties of the LS Estimator..... <i>The LS Estimator is Unbiased</i>	236
	<i>Covariance Matrix of the LS Estimator</i>	236
	<i>The LS Estimator is BLUE in the Presence of White Noise</i>	237
	<i>The LS Estimator Achieves the Cramér-Rao Bound for White Gaussian Noise</i>	238
	<i>Asymptotic Distribution of the LS Estimator.....</i>	238
6.4	Orthogonalizing the Column Space of X : The SVD Method	239
	<i>Pseudo-Inverse Matrix and SVD.....</i>	240
6.5	Ridge Regression	243
	<i>Principal Components Regression</i>	245
6.6	The Recursive Least-Squares Algorithm	245
	<i>Time-Iterative Computations of Φ_n, p_n</i>	247
	<i>Time Updating of θ_n</i>	247
6.7	Newton's Iterative Minimization Method	249
6.7.1	RLS and Newton's Method	252
6.8	Steady-State Performance of the RLS	253
6.9	Complex-Valued Data: The Widely Linear RLS.....	255
6.10	Computational Aspects of the LS Solution	256
	<i>Cholesky Factorization</i>	256
	<i>QR Factorization</i>	256
	<i>Fast RLS Versions</i>	257
6.11	The Coordinate and Cyclic Coordinate Descent Methods.....	259
6.12	Simulation Examples	260
6.13	Total-Least-Squares	262
	<i>Geometric Interpretation of the Total-Least-Squares Method.....</i>	267
Problems		269
	<i>MATLAB Exercises</i>	272
References		273

6.1 INTRODUCTION

The squared error loss function was at the center of our attention in the previous two chapters. The sum of error-squares cost was introduced in [Chapter 3](#), followed by the mean-square error (MSE) version, treated in [Chapter 4](#). The stochastic gradient descent technique was employed in [Chapter 5](#) to help us bypass the need to perform expectations for obtaining the second order statistics of the data, as required by the MSE formulation.

In this chapter, we return to the original formulation of the sum of error squares, and our goal is to look more closely at the resulting family of algorithms and their properties. A major part of the chapter will be dedicated to the recursive least-squares (RLS) algorithm, which is an online scheme that solves the least-squares (LS) optimization task. The spine of the RLS scheme comprises an efficient update of the inverse (sample) covariance matrix of the input data, whose rationale can also be adopted in the context of different learning methods for developing related online schemes; this is one of the reasons we pay special tribute to the RLS algorithm. The other reason is its popularity in a large number of signal processing/machine learning tasks, due to some attractive properties that this scheme enjoys.

Finally, at the end of the chapter, a more general formulation of the LS task, known as the total-least-squares (TLS), is given and reviewed.

6.2 LEAST-SQUARES LINEAR REGRESSION: A GEOMETRIC PERSPECTIVE

We begin with our familiar linear regression model. Given a set of observations,

$$y_n = \boldsymbol{\theta}^T \mathbf{x}_n + \eta_n, \quad n = 1, 2, \dots, N, \quad y_n \in \mathbb{R}, \quad \mathbf{x}_n \in \mathbb{R}^l, \quad \boldsymbol{\theta} \in \mathbb{R}^l,$$

where η_n denotes the (unobserved) values of a *zero* mean noise source, the task is to obtain an estimate of the unknown parameter vector, $\boldsymbol{\theta}$, so that

$$\hat{\boldsymbol{\theta}}_{\text{LS}} = \arg \min_{\boldsymbol{\theta}} \sum_{n=1}^N (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2. \quad (6.1)$$

Our stage of discussion is that of real numbers, and we will point out differences with the complex number case whenever needed. Moreover, we assume that our data have been centered around their sample means; alternatively, the intercept, θ_0 , can be absorbed in $\boldsymbol{\theta}$ with a corresponding increase in the dimensionality of \mathbf{x}_n . Define,

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \mathbb{R}^N, \quad \mathbf{X} := \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix} \in \mathbb{R}^{N \times l}. \quad (6.2)$$

Equation (6.1) can be recast as,

$$\hat{\boldsymbol{\theta}}_{\text{LS}} = \arg \min_{\boldsymbol{\theta}} \|\mathbf{e}\|^2,$$

where

$$\mathbf{e} := \mathbf{y} - \mathbf{X}\boldsymbol{\theta},$$

and $\|\cdot\|$ denotes the Euclidean norm, which measures the “distance” between the respective vectors in \mathbb{R}^N , i.e., \mathbf{y} and $X\boldsymbol{\theta}$. Let us denote as $\mathbf{x}_1^c, \dots, \mathbf{x}_l^c \in \mathbb{R}^N$ the columns of X , i.e.,

$$X = [\mathbf{x}_1^c, \dots, \mathbf{x}_l^c].$$

Then we can write,

$$\hat{\mathbf{y}} := X\boldsymbol{\theta} = \sum_{i=1}^l \theta_i \mathbf{x}_i^c,$$

and

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}.$$

Obviously, $\hat{\mathbf{y}}$ represents a vector that lies in the $\text{span}\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$. Thus, naturally, our task now becomes that of selecting $\boldsymbol{\theta}$ so that the error vector between \mathbf{y} and $\hat{\mathbf{y}}$ has minimum norm. According to the Pythagorean theorem of orthogonality for Euclidean spaces this is achieved if $\hat{\mathbf{y}}$ is chosen as the orthogonal projection of \mathbf{y} onto the $\text{span}\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$. **Figure 6.1** illustrates the geometry. Recalling the concept of orthogonal projections ([Section 5.6](#), Eq. (5.64)), we obtain

$$\boxed{\hat{\mathbf{y}} = X(X^T X)^{-1} X^T \mathbf{y} : \text{ LS Estimate,}} \quad (6.3)$$

assuming that $X^T X$ is invertible. It is common to describe the LS solution in terms of the *Moore-Penrose pseudo-inverse* of X , which for a tall matrix is defined as,

$$\boxed{X^\dagger := (X^T X)^{-1} X^T : \text{ Pseudo-inverse of a Tall Matrix } X,} \quad (6.4)$$

and hence we can write,

$$\hat{\boldsymbol{\theta}}_{\text{LS}} = X^\dagger \mathbf{y}. \quad (6.5)$$

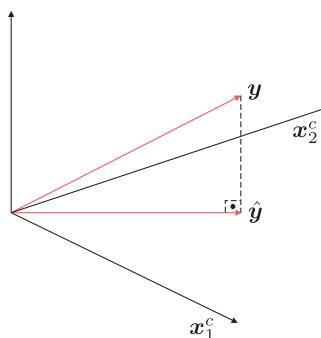


FIGURE 6.1

The LS estimate is chosen so that $\hat{\mathbf{y}}$ is the orthogonal projection of \mathbf{y} onto the $\text{span}\{\mathbf{x}_1^c, \mathbf{x}_2^c\}$, that is the columns of X .

Thus, we have rederived Eq. (3.17) of [Chapter 3](#), this time via geometric arguments. Note that the pseudo-inverse is a generalization of the notion of the inverse of a square matrix. Indeed, if X is square, then it is readily seen that the pseudo inverse coincides with X^{-1} . For complex-valued data, the only difference is that transposition is replaced by the Hermitian one.

6.3 STATISTICAL PROPERTIES OF THE LS ESTIMATOR

Some of the statistical properties of the LS estimator were touched on in [Chapter 3](#), for the special case of a random real parameter. Here, we will look at this issue in a more general setting. Assume that there exists a true (yet unknown) parameter/weight vector, θ_o , that generates the output (dependent) random variables (stacked in a random vector $y \in \mathbb{R}^N$), according to the model,

$$y = X\theta_o + \eta,$$

where η is a zero mean noise vector. Observe that we have assumed that X is fixed and not random; that is, the randomness underlying the output variables, y , is due solely to the noise. Under the previously stated assumptions, the following properties hold.

The LS estimator is unbiased

The LS estimator for the parameters is given by

$$\begin{aligned}\hat{\theta}_{LS} &= (X^T X)^{-1} X^T y, \\ &= (X^T X)^{-1} X^T (X\theta_o + \eta) = \theta_o + (X^T X)^{-1} X^T \eta,\end{aligned}\tag{6.6}$$

or

$$\mathbb{E}[\hat{\theta}_{LS}] = \theta_o + (X^T X)^{-1} X^T \mathbb{E}[\eta] = \theta_o,$$

which proves the claim.

Covariance matrix of the LS estimator

Let in addition to the previously adopted assumptions that

$$\mathbb{E}[\eta \eta^T] = \sigma_\eta^2 I,$$

that is, the source generating the noise samples is white. By the definition of the covariance matrix, we get

$$\Sigma_{\hat{\theta}_{LS}} = \mathbb{E}[(\hat{\theta}_{LS} - \theta_o)(\hat{\theta}_{LS} - \theta_o)^T],$$

and substituting $\hat{\theta}_{LS} - \theta_o$ from (6.6), we obtain

$$\begin{aligned}\Sigma_{\hat{\theta}_{LS}} &= \mathbb{E}[(X^T X)^{-1} X^T \eta \eta^T X (X^T X)^{-1}] \\ &= (X^T X)^{-1} X^T \mathbb{E}[\eta \eta^T] X (X^T X)^{-1} \\ &= \sigma_\eta^2 (X^T X)^{-1}.\end{aligned}\tag{6.7}$$

Note that, for large values of N , we can write

$$X^T X = \sum_{n=1}^N x_n x_n^T \approx N \Sigma_x,$$

where Σ_x is the covariance matrix of our (zero mean) input variables, i.e.,

$$\Sigma_x := \mathbb{E}[\mathbf{x}_n \mathbf{x}_n^T] \approx \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T.$$

Thus, for large values of N , we can write

$$\Sigma_{\hat{\theta}_{LS}} \approx \frac{\sigma_\eta^2}{N} \Sigma_x^{-1}.$$

(6.8)

In other words, under the adopted assumptions, the LS estimator is not only unbiased, but its covariance matrix *tends asymptotically to zero*. That is, with high probability, the estimate $\hat{\theta}_{LS}$, which is obtained via a large number of measurements, will be close to the true value θ_o . Viewing it slightly differently, note that the LS solution tends to the MSE solution, which was discussed in [Chapter 4](#). Indeed, for the case of centered data,

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T = \Sigma_x,$$

and

$$\lim_{N \rightarrow \infty} \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n y_n = \mathbb{E}[\mathbf{x}\mathbf{y}] = \mathbf{p}.$$

Moreover, we know that for the linear regression modeling case, the normal equations, $\Sigma_x \theta = \mathbf{p}$, result in the solution $\theta = \theta_o$ ([Remarks 4.2](#)).

The LS estimator is BLUE in the presence of white noise

The notion of the best linear unbiased estimator (BLUE) was introduced in [Section 4.9.1](#) in the context of the *Gauss Markov* theorem. Let $\hat{\theta}$ denote any other *linear* unbiased estimator, under the assumption that

$$\mathbb{E}[\boldsymbol{\eta}\boldsymbol{\eta}^T] = \sigma_\eta^2 I.$$

Then, by its definition, the estimator will have a linear dependence on the output data, i.e.,

$$\hat{\theta} = H\mathbf{y}, \quad H \in \mathbb{R}^{l \times N}.$$

It will be shown that

$$\mathbb{E}[(\hat{\theta} - \theta_o)^T(\hat{\theta} - \theta_o)] \geq \mathbb{E}[(\hat{\theta}_{LS} - \theta_o)^T(\hat{\theta}_{LS} - \theta_o)]. \quad (6.8a)$$

Indeed, from the respective definitions we have

$$\hat{\theta} = H(X\theta_o + \boldsymbol{\eta}) = HX\theta_o + H\boldsymbol{\eta}. \quad (6.9)$$

However, because $\hat{\theta}$ has been assumed unbiased, then (6.9) implies that, $HX = I$ and

$$\hat{\theta} - \theta_o = H\boldsymbol{\eta}.$$

Thus,

$$\begin{aligned}\Sigma_{\hat{\theta}} &:= \mathbb{E}[(\hat{\theta} - \theta_o)(\hat{\theta} - \theta_o)^T] \\ &= \sigma_\eta^2 HH^T.\end{aligned}$$

However, taking into account that $HX = I$, it is easily checked out (try it) that

$$\sigma_\eta^2 HH^T = \sigma_\eta^2 (H - X^\dagger)(H - X^\dagger)^T + \sigma_\eta^2 (X^T X)^{-1},$$

where X^\dagger is the respective pseudo-inverse matrix, defined in (6.4).

Because $\sigma_n^2(H - X^\dagger)(H - X^\dagger)^T$ is a positive semidefinite matrix, its trace is nonnegative (Problem 6.1) and thus we have that

$$\text{trace}\{\sigma_n^2 HH^T\} \geq \text{trace}\{\sigma_n^2 (X^T X)^{-1}\},$$

and recalling (6.7), Eq. (6.8a) is proved, with equality only if

$$H = X^\dagger = (X^T X)^{-1} X^T.$$

Note that this result could have been obtained directly from (4.102) by setting $\Sigma_\eta = \sigma_\eta^2 I$. This also emphasizes the fact that if the noise is not white, then the LS parameter estimator is *no more BLUE*.

The LS estimator achieves the Cramér-Rao bound for white Gaussian noise

The concept of the Cramér-Rao lower bound was introduced in Chapter 3. There, it was shown that, under the *white Gaussian noise* assumption, the LS estimator of a real number was *efficient*; that is, it achieves the CR bound. Moreover, in Problem 3.8, it was shown that if η is zero mean Gaussian noise with covariance matrix Σ_η , then the efficient estimator is given by

$$\hat{\theta} = (X^T \Sigma_\eta^{-1} X)^{-1} X^T \Sigma_\eta^{-1} \mathbf{y},$$

which for $\Sigma_\eta = \sigma_\eta^2 I$ coincides with the LS estimator. In other words, under the white Gaussian noise assumption, the LS estimator becomes *minimum variance unbiased estimator* (MVUE). This is a strong result. No other unbiased estimator (*not necessarily linear*) will do better than the LS one. Note that this result holds true not asymptotically, but also for finite number of samples N . If one wishes to decrease further the mean-square error, then *biased* estimators, as produced via regularization, have to be considered; this has already been discussed in Chapter 3; see also [16, 50] and the references therein.

Asymptotic distribution of the LS estimator

We have already seen that the LS estimator is unbiased and that its covariance matrix is approximately (for large values of N) given by (6.8). Thus, as $N \rightarrow \infty$, the variance around the true value, θ_o , is becoming increasingly small. Furthermore, there is a stronger result, which provides the distribution of the LS estimator for large values of N . Under some general assumptions, such as independence of successive observation vectors and that the white noise source is independent of the input, and mobilizing the central limit theorem, it can be shown (Problem 6.2) that

$$\boxed{\sqrt{N}(\hat{\theta}_{LS} - \theta_o) \xrightarrow{\text{---}} \mathcal{N}(\mathbf{0}, \sigma^2 \Sigma_x^{-1})}, \quad (6.10)$$

where the limit is meant to be in *distribution* (see [Section 2.6](#)). Alternatively, we can write that

$$\hat{\theta}_{LS} \sim \mathcal{N}\left(\theta_o, \frac{\sigma^2}{N} \Sigma_x^{-1}\right).$$

In other words, the LS parameter estimator is asymptotically distributed according to the normal distribution.

6.4 ORTHOGONALIZING THE COLUMN SPACE OF X : THE SVD METHOD

The *singular value decomposition* (SVD) of a matrix is among the most powerful tools in linear algebra. Due to its importance in machine learning, we present the basic theory here and exploit it to shed light onto our LS estimation task from a different angle. We start by considering the general case, and then we tailor the theory to our specific needs.

Let X be an $m \times l$ matrix and allow its rank, r , not to be necessarily full, i.e.,

$$r \leq \min(m, l).$$

Then, there exist *orthogonal* matrices,¹ U and V , of dimensions $m \times m$ and $l \times l$, respectively, so that

$$X = U \begin{bmatrix} D & O \\ O & O \end{bmatrix} V^T : \quad \text{Singular Value Decomposition of } X,$$

(6.11)

where D is an $r \times r$ *diagonal matrix*² with elements $\sigma_i = \sqrt{\lambda_i}$, known as the *singular values* of X , where λ_i , $i = 1, 2, \dots, r$, are the *nonzero* eigenvalues of XX^T ; matrices denoted as O comprise zero elements and are of appropriate dimensions.

Taking into account the zero elements in the diagonal matrix, (6.11) can be rewritten as

$$X = U_r D V_r^T = \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^T,$$
(6.12)

where

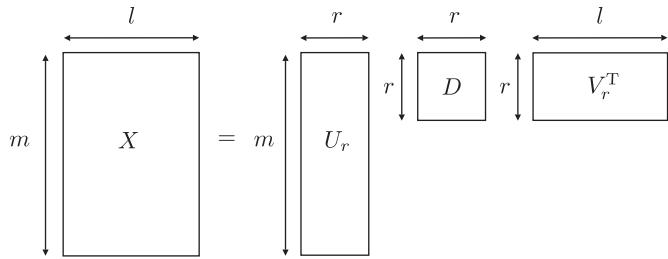
$$U_r := [\mathbf{u}_1, \dots, \mathbf{u}_r] \in \mathbb{R}^{m \times r}, \quad V_r := [\mathbf{v}_1, \dots, \mathbf{v}_r] \in \mathbb{R}^{l \times r}.$$
(6.13)

Equation (6.12) provides a matrix factorization of X in terms of U_r , V_r , and D . We will make use of this factorization in [Chapter 19](#), when dealing with dimensionality reduction techniques. [Figure 6.2](#) offers a schematic illustration of (6.12).

It turns out that $\mathbf{u}_i \in \mathbb{R}^m$, $i = 1, 2, \dots, r$, known as *left singular vectors*, are the normalized eigenvectors corresponding to the nonzero eigenvalues of XX^T , and $\mathbf{v}_i \in \mathbb{R}^l$, $i = 1, 2, \dots, r$, are the normalized eigenvectors associated with the nonzero eigenvalues of X^TX , and they are known as *right singular vectors*. Note that both XX^T and X^TX share the same eigenvalues ([Problem 6.3](#)).

¹ Recall that a square matrix U is called orthogonal if $U^T U = U U^T = I$. For complex-valued square matrices, if $U^H U = U U^H = I$, it is called unitary.

² Usually it is denoted as Σ , but here we avoid the notation so as not to confuse it with the covariance matrix Σ ; D reminds us of its diagonal structure.

**FIGURE 6.2**

The $m \times l$ matrix X , of rank $r \leq \min(m, l)$, factorizes in terms of the matrices $U_r \in \mathbb{R}^{m \times r}$, $V_r \in \mathbb{R}^{l \times r}$ and the $r \times r$ diagonal matrix D .

Proof. By the respective definitions, we have

$$XX^T \mathbf{u}_i = \lambda_i \mathbf{u}_i, \quad i = 1, 2, \dots, r, \quad (6.14)$$

and

$$X^T X \mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad i = 1, 2, \dots, r. \quad (6.15)$$

Moreover, because XX^T and $X^T X$ are symmetric matrices, it is known from linear algebra that their eigenvalues are real³ and the respective eigenvectors are orthogonal, which can then be normalized to unit norm to become orthonormal ([Problem 6.4](#)). It is a matter of simple algebra ([Problem 6.5](#)) to show from (6.14) and (6.15) that,

$$\mathbf{u}_i = \frac{1}{\sigma_i} X \mathbf{v}_i, \quad i = 1, 2, \dots, r. \quad (6.16)$$

Thus, we can write that

$$\sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^T = X \sum_{i=1}^r \mathbf{v}_i \mathbf{v}_i^T = X \sum_{i=1}^l \mathbf{v}_i \mathbf{v}_i^T = X V V^T,$$

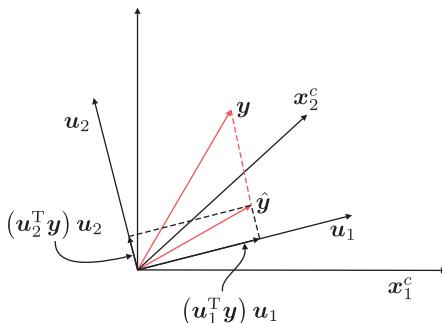
where we used the fact that for eigenvectors corresponding to $\sigma_i = 0$ ($\lambda_i = 0$), $i = r+1, \dots, l$, $X \mathbf{v}_i = \mathbf{0}$. However, due to the orthonormality of \mathbf{v}_i , $i = 1, 2, \dots, l$, $V V^T = I$ and the claim in (6.12) has been proved. \square

Pseudo-inverse matrix and SVD

Let us now elaborate on the SVD expansion. By the definition of the pseudo-inverse, X^\dagger , and assuming the $N \times l$ ($N > l$) data matrix to be full column rank ($r = l$), then employing (6.12) in (6.5) we get ([Problem 6.6](#)),

$$\begin{aligned} \hat{\mathbf{y}} &= X \hat{\boldsymbol{\theta}}_{\text{LS}} = X(X^T X)^{-1} X^T \mathbf{y} \\ &= U_l U_l^T \mathbf{y} = [\mathbf{u}_1, \dots, \mathbf{u}_l] \begin{bmatrix} \mathbf{u}_1^T \mathbf{y} \\ \vdots \\ \mathbf{u}_l^T \mathbf{y} \end{bmatrix}, \end{aligned}$$

³ This is also true for complex matrices, XX^H , $X^H X$.

**FIGURE 6.3**

The eigenvectors $\mathbf{u}_1, \mathbf{u}_2$, from an orthonormal basis, in $\text{span}\{\mathbf{x}_1^c, \mathbf{x}_2^c\}$; that is, the column space of X . $\hat{\mathbf{y}}$ is the projection of \mathbf{y} onto this subspace.

or

$$\hat{\mathbf{y}} = \sum_{i=1}^l (\mathbf{u}_i^T \mathbf{y}) \mathbf{u}_i : \quad \text{LS Estimate in Terms of an Orthonormal Basis.} \quad (6.17)$$

The latter represents the *projection* of \mathbf{y} onto the column space of X , i.e., $\text{span}\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$ using a corresponding *orthonormal* basis, $\{\mathbf{u}_1, \dots, \mathbf{u}_l\}$, to describe the subspace, see Figure 6.3. Note that each \mathbf{u}_i , $i = 1, 2, \dots, l$, lies in the space spanned by the columns of X as it is suggested from Eq. (6.16). Moreover, it is easily shown that we can write

$$X^\dagger = (X^T X)^{-1} X^T = V_l D^{-1} U_l^T = \sum_{i=1}^l \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T.$$

As a matter of fact, this is in line with the more general definition of a pseudo-inverse in linear algebra, including matrices that are not full rank (i.e., $X^T X$ is not invertible), namely

$$X^\dagger := V_r D^{-1} U_r^T = \sum_{i=1}^r \frac{1}{\sigma_i} \mathbf{v}_i \mathbf{u}_i^T : \quad \text{Pseudo-inverse of a Matrix of Rank } r. \quad (6.18)$$

In the case of matrices with $N < l$, and assuming that the rank of X is equal to N , then it is readily verified that the previous generalized definition of the pseudo-inverse is equivalent to

$$X^\dagger = X^T (X X^T)^{-1} : \quad \text{Pseudo-inverse of a Fat Matrix, } X. \quad (6.19)$$

Note that a system with N equations and $l > N$ unknowns,

$$X\boldsymbol{\theta} = \mathbf{y},$$

has infinite solutions. Such systems are known as *underdetermined*, to be contrasted with the *over-determined* systems for which $N > l$. It can be shown that for underdetermined systems, the solution $\boldsymbol{\theta} = X^\dagger \mathbf{y}$ is the one with the minimum Euclidean norm. We will consider the case of such systems of equations in more detail in Chapter 9, in the context of sparse models.

Remarks 6.1.

- Computing the pseudo-inverse using the SVD is numerically more robust than the direct method via the inversion of $(X^T X)^{-1}$.
- *k-rank matrix approximation:* The best rank $k < r \leq \min(m, l)$ approximation matrix, $\hat{X} \in \mathbb{R}^{m \times l}$, of $X \in \mathbb{R}^{m \times l}$ in the Frobenius, $\|\cdot\|_F$, as well as in the spectral, $\|\cdot\|_2$, norms sense is given by (e.g., [26]),

$$\hat{X} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T, \quad (6.20)$$

with the previously stated norms defined as (Problem 6.9),

$$\|X\|_F := \sqrt{\sum_i \sum_j |X(i,j)|^2} = \sqrt{\sum_{i=1}^r \sigma_i^2} : \text{Frobenius Norm of } X, \quad (6.21)$$

and

$$\|X\|_2 := \sigma_1 : \text{Spectral Norm of } X, \quad (6.22)$$

where, $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ are the singular values of X . In other words, \hat{X} in (6.20) minimizes the error matrix norms,

$$\|X - \hat{X}\|_F \quad \text{and} \quad \|X - \hat{X}\|_2.$$

Moreover, it turns out that the approximation error is given by (Problems 6.10 and 6.11),

$$\|X - \hat{X}\|_F = \sqrt{\sum_{i=k+1}^r \sigma_i^2}, \quad \|X - \hat{X}\|_2 = \sigma_{k+1}.$$

This is also known as the *Eckart-Young-Mirsky theorem*.

- *Null and range spaces of X :* Let the rank of an $m \times l$ matrix, X , be equal to $r \leq \min(m, l)$. Then, the following easily shown properties hold (Problem 6.13): The null space of X , $\mathcal{N}(X)$, defined as

$$\mathcal{N}(X) := \{\mathbf{x} \in \mathbb{R}^l : X\mathbf{x} = \mathbf{0}\}, \quad (6.23)$$

is also expressed as

$$\mathcal{N}(X) = \text{span}\{\mathbf{v}_{r+1}, \dots, \mathbf{v}_l\}. \quad (6.24)$$

Furthermore, the range space of X , $\mathcal{R}(X)$, defined as

$$\mathcal{R}(X) := \{\mathbf{x} \in \mathbb{R}^l : \exists \mathbf{a} \text{ such as } X\mathbf{a} = \mathbf{x}\}, \quad (6.25)$$

is expressed as

$$\mathcal{R}(X) = \text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\}. \quad (6.26)$$

- Everything that has been said before transfers to complex-valued data, trivially, by replacing transposition with the Hermitian one.

6.5 RIDGE REGRESSION

Ridge regression was introduced in [Chapter 3](#) as a means to impose bias on the LS solution and also as a major path to cope with overfitting and ill-conditioning problems. In ridge regression, the minimizer results as

$$\hat{\boldsymbol{\theta}}_R = \arg \min_{\boldsymbol{\theta}} \left\{ \|\mathbf{y} - X\boldsymbol{\theta}\|^2 + \lambda \|\boldsymbol{\theta}\|^2 \right\},$$

where $\lambda > 0$ is a user-defined parameter that controls the importance of the regularizing term. Taking the gradient w.r.t. $\boldsymbol{\theta}$ and equating to zero results in

$$\hat{\boldsymbol{\theta}}_R = (X^T X + \lambda I)^{-1} X^T \mathbf{y}. \quad (6.27)$$

Looking at (6.27), we readily observe (a) its “stabilizing” effect from the numerical point of view, when $X^T X$ has large condition number, and (b) its biasing effect on the (unbiased) LS solution. Note that ridge regression provides a solution even if $X^T X$ is not invertible, as is the case when $N < l$. Let us now employ the SVD expansion of (6.12) in (6.27). Assuming a full column rank matrix, X , we obtain ([Problem 6.14](#)),

$$\hat{\mathbf{y}} = X\hat{\boldsymbol{\theta}}_R = U_l D(D^2 + \lambda I)^{-1} D U_l^T \mathbf{y},$$

or

$$\hat{\mathbf{y}} = \sum_{i=1}^l \frac{\sigma_i^2}{\lambda + \sigma_i^2} (\mathbf{u}_i^T \mathbf{y}) \mathbf{u}_i : \quad \text{Ridge Regression Shrinks the Weights.}$$

(6.28)

Comparing (6.28) and (6.17), we observe that the components of the projection of \mathbf{y} onto the $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_l\}$ ($\text{span}\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$) are *shrunk* with respect to their LS counterpart. Moreover, the shrinking level depends on the singular values, σ_i ; the smaller the value of σ_i , the higher the shrinking of the corresponding component. Let us now turn our attention to investigate the geometric interpretation of this algebraic finding. This small diversion will also provide more insight in the interpretation of \mathbf{v}_i and \mathbf{u}_i , $i = 1, 2, \dots, l$, that appear in the SVD method.

Recall that $X^T X$ is a scaled version of the sample covariance matrix for centered regressors. Also, by the definition of the \mathbf{v}_i 's, we have

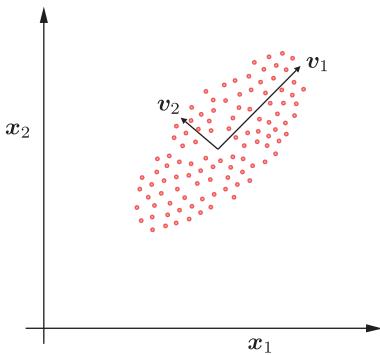
$$(X^T X) \mathbf{v}_i = \sigma_i^2 \mathbf{v}_i, \quad i = 1, 2, \dots, l,$$

and in a compact form,

$$\begin{aligned} (X^T X) V_l &= V_l \text{diag}\{\sigma_1^2, \dots, \sigma_l^2\} \Rightarrow \\ (X^T X) &= V_l D^2 V_l^T = \sum_{i=1}^l \sigma_i^2 \mathbf{v}_i \mathbf{v}_i^T. \end{aligned} \quad (6.29)$$

Note that in (6.29), the (scaled) sample covariance matrix is written as a sum of rank one matrices, $\mathbf{v}_i \mathbf{v}_i^T$, each one weighted by the square of respective singular value, σ_i^2 . We are now close to revealing the physical/geometric meaning of the singular values. To this end, define

$$\mathbf{q}_j := X \mathbf{v}_j = \begin{bmatrix} \mathbf{x}_1^T \mathbf{v}_j \\ \vdots \\ \mathbf{x}_N^T \mathbf{v}_j \end{bmatrix} \in \mathbb{R}^N, \quad j = 1, 2, \dots, l. \quad (6.30)$$

**FIGURE 6.4**

The singular vector \mathbf{v}_1 , which is associated with the singular value $\sigma_1 > \sigma_2$, points to the direction where most of the (variance) activity in the data space takes place. The variance in the direction of \mathbf{v}_2 is smaller.

Note that \mathbf{q}_j is a vector in the column space of X . Moreover, the respective squared norm of \mathbf{q}_j is given by

$$\sum_{n=1}^N q_j^2(n) = \mathbf{q}_j^T \mathbf{q}_j = \mathbf{v}_j^T X^T X \mathbf{v}_j = \mathbf{v}_j^T \left(\sum_{i=1}^l \sigma_i^2 \mathbf{v}_i \mathbf{v}_i^T \right) \mathbf{v}_j = \sigma_j^2,$$

due to the orthonormality of the \mathbf{v}_j 's. That is, σ_j^2 is equal to the (scaled) sample variance of the elements of \mathbf{q}_j . However, by the definition in (6.30), this is the sample variance of the projections of the input vectors (regressors), \mathbf{x}_n , $n = 1, 2, \dots, N$, along the direction \mathbf{v}_j . The larger the value of σ_j , the larger the spread of the (input) data along the respective direction. This is shown in Figure 6.4, where $\sigma_1 \gg \sigma_2$. From the variance point of view, \mathbf{v}_1 is the more *informative* direction, compared to \mathbf{v}_2 . It is the direction where most of the activity takes place. This observation is at the heart of *dimensionality reduction*, which will be treated in more detail in Chapter 19. Moreover, from (6.16), we obtain

$$\mathbf{q}_j = X \mathbf{v}_j = \sigma_j \mathbf{u}_j. \quad (6.31)$$

In other words, \mathbf{u}_j points in the direction of \mathbf{q}_j . Thus, (6.28) suggests that while projecting \mathbf{y} onto the column space of X , the directions, \mathbf{u}_j , associated with larger values of variance are weighted more heavily than the rest. *Ridge regression respects and assigns higher weights to the more informative directions, where most of the data activity takes place.* Alternatively, the less important directions, those associated with small data variance, are shrunk the most.

One final comment concerning ridge regression is that the ridge solutions are not invariant under scaling of the input variables. This becomes obvious by looking at the respective equations. Thus, in practice, often the input variables are standardized to unit variances.

Principal components regression

We have just seen that the effect of the ridge regression is to enforce a shrinking rule on the parameters, which decreases the contribution of the less important of the components, \mathbf{u}_i , in the respective summation. This can be considered as a soft shrinkage rule. An alternative path is to adopt a hard

thresholding rule and keep only the m most significant directions, known as the *principal axes* or *directions*, and forget the rest by setting the respective weights equal to zero. Equivalently, we can write

$$\hat{\mathbf{y}} = \sum_{i=1}^m \hat{\theta}_i \mathbf{u}_i, \quad (6.32)$$

where

$$\hat{\theta}_i = \mathbf{u}_i^T \mathbf{y}, \quad i = 1, 2, \dots, m. \quad (6.33)$$

Furthermore, employing (6.16) we have that

$$\hat{\mathbf{y}} = \sum_{i=1}^m \frac{\hat{\theta}_i}{\sigma_i} X \mathbf{v}_i, \quad (6.34)$$

or equivalently, the weights for the expansion of the solution in terms of the input data can be expressed as

$$\boldsymbol{\theta} = \sum_{i=1}^m \frac{\hat{\theta}_i}{\sigma_i} \mathbf{v}_i. \quad (6.35)$$

In other words, the prediction $\hat{\mathbf{y}}$ is performed in a subspace of the column space of X , which is spanned by the m principal axes; that is, the subspace where most of the data activity takes place.

6.6 THE RECURSIVE LEAST-SQUARES ALGORITHM

In previous chapters, we discussed the need for developing recursive algorithms that update the estimates every time a new pair of input-output training samples is received. Solving the LS problem using a general purpose solver would amount to $\mathcal{O}(l^3)$ MADS, due to the involved matrix inversion. Also, $\mathcal{O}(Nl^2)$ operations are required to compute the (scaled) sample covariance matrix $X^T X$. In this section, the special structure of $X^T X$ will be taken into account in order to obtain a computationally efficient online scheme for the solution of the LS task. Moreover, when dealing with time recursive techniques, one can also care for time variations of the statistical properties of the involved data. In this section, we will allow for such applications, and the LS cost will be slightly modified in order to accommodate time varying environments.

For the purpose of the section, we will slightly “enrich” our notation and we will use explicitly the time index, n . Also, to be consistent with the previously discussed online schemes, we will assume that the time starts at $n = 0$ and the received observations are (y_n, \mathbf{x}_n) , $n = 0, 1, 2, \dots$. To this end, let us denote the input matrix at time n as

$$X_n^T = [\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_n].$$

Moreover, the least-squares cost function in (6.1) is modified to involve a *forgetting factor*, $0 < \beta \leq 1$. The purpose of its presence is to help the cost function slowly forget past data samples by weighting heavier the more recent observations. This will equip the algorithm with the ability to track changes that occur in the underlying data statistics. Moreover, since we are interested in time recursive solutions, starting from time $n = 0$, we are forced to introduce regularization. During the

initial period, corresponding to time instants $n < l - 1$, the corresponding system of equations will be underdetermined and $X_n^T X_n$ is not invertible. Indeed, we have that

$$X_n^T X_n = \sum_{i=0}^n \mathbf{x}_i \mathbf{x}_i^T.$$

In other words, $X_n^T X_n$ is the sum of rank one matrices. Hence, for $n < l - 1$ its rank is necessarily less than l , and it cannot be inverted. For larger values of n , it can become full rank, provided that at least l of the input vectors are linearly independent, which is usually assumed to be the case. The previous arguments lead to the following modifications of the “conventional” least-squares, known as the *exponentially weighted least-squares* cost function, minimized by

$$\boldsymbol{\theta}_n = \arg \min_{\boldsymbol{\theta}} \left(\sum_{i=0}^n \beta^{n-i} (y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2 + \lambda \beta^{n+1} \|\boldsymbol{\theta}\|^2 \right),$$

(6.36)

where β is a user-defined parameter, very close to unity, as when $\beta = 0.999$. In this way, the more recent samples are weighted heavier than the older ones. Note that the regularizing parameter has been made time varying. This is because for large values of n , no regularization is required. Indeed for $n > l$, matrix $X_n^T X_n$ becomes, in general, invertible. Moreover, recall from Chapter 3 that the use of regularization also takes precautions for overfitting. However, for very large values of $n \gg l$, this is not a problem, and one wishes to get rid of the imposed bias. The parameter $\lambda > 0$ is also a user-defined variable and its choice will be discussed later on.

Minimizing (6.36) results in

$$\Phi_n \boldsymbol{\theta}_n = \mathbf{p}_n, \quad (6.37)$$

where,

$$\Phi_n = \sum_{i=0}^n \beta^{n-i} \mathbf{x}_i \mathbf{x}_i^T + \lambda \beta^{n+1} I, \quad (6.38)$$

and

$$\mathbf{p}_n = \sum_{i=0}^n \beta^{n-i} \mathbf{x}_i y_i, \quad (6.39)$$

which for $\beta = 1$ coincides with the ridge regression.

Time-iterative computations of Φ_n , \mathbf{p}_n

By the respective definitions, we have that

$$\Phi_n = \beta \Phi_{n-1} + \mathbf{x}_n \mathbf{x}_n^T, \quad (6.40)$$

and

$$\mathbf{p}_n = \beta \mathbf{p}_{n-1} + \mathbf{x}_n y_n. \quad (6.41)$$

Recall Woodbury’s matrix inversion formula (Appendix A.1),

$$(A + BD^{-1}C)^{-1} = A^{-1} - A^{-1}B(D + CA^{-1}B)^{-1}CA^{-1}.$$

Plugging it in (6.40) and after the appropriate inversion and substitutions we obtain,

$$\Phi_n^{-1} = \beta^{-1} \Phi_{n-1}^{-1} - \beta^{-1} K_n \mathbf{x}_n^T \Phi_{n-1}^{-1}, \quad (6.42)$$

$$K_n = \frac{\beta^{-1} \Phi_{n-1}^{-1} \mathbf{x}_n}{1 + \beta^{-1} \mathbf{x}_n^T \Phi_{n-1}^{-1} \mathbf{x}_n}. \quad (6.43)$$

K_n is known as the *Kalman gain*. For notational convenience, define

$$P_n = \Phi_n^{-1}.$$

Also, rearranging the terms in (6.43), we get

$$K_n = \left(\beta^{-1} P_{n-1} - \beta^{-1} K_n \mathbf{x}_n^T P_{n-1} \right) \mathbf{x}_n,$$

and taking into account (6.42) results in

$$K_n = P_n \mathbf{x}_n. \quad (6.44)$$

Time updating of θ_n

From (6.37), (6.41)–(6.43) we obtain

$$\begin{aligned} \theta_n &= \left(\beta^{-1} P_{n-1} - \beta^{-1} K_n \mathbf{x}_n^T P_{n-1} \right) \beta \mathbf{p}_{n-1} + P_n \mathbf{x}_n y_n \\ &= \theta_{n-1} - K_n \mathbf{x}_n^T \theta_{n-1} + K_n y_n, \end{aligned}$$

and finally,

$$\theta_n = \theta_{n-1} + K_n e_n, \quad (6.45)$$

where,

$$e_n := y_n - \theta_{n-1}^T \mathbf{x}_n. \quad (6.46)$$

The derived algorithm is summarized in [Algorithm 6.1](#).

Algorithm 6.1 .[(The RLS algorithm)]

- Initialize
 - $\theta_{-1} = \mathbf{0}$; any other value is also possible.
 - $P_{-1} = \lambda^{-1} I$; $\lambda > 0$ a user-defined variable.
 - Select β ; close to 1.
- **For** $n = 0, 1, \dots$, **Do**
 - $e_n = y_n - \theta_{n-1}^T \mathbf{x}_n$
 - $z_n = P_{n-1} \mathbf{x}_n$
 - $K_n = \frac{z_n}{\beta + \mathbf{x}_n^T z_n}$
 - $\theta_n = \theta_{n-1} + K_n e_n$
 - $P_n = \beta^{-1} P_{n-1} - \beta^{-1} K_n z_n^T$
- **End For**

Remarks 6.2.

- The complexity of the RLS algorithm is of the order $\mathcal{O}(l^2)$ per iteration, due to the matrix-product operations. That is, there is an order of magnitude difference compared to the LMS and the other schemes that were discussed in [Chapter 5](#). In other words, RLS does not scale well with dimensionality.
- The RLS algorithm shares similar numerical behavior with the Kalman filter, which was discussed in [Section 4.10](#). P_n may lose its positive definite and symmetric nature, which then leads the algorithm to divergence. To remedy such a tendency, symmetry-preserving versions of the RLS algorithm have been derived; see [65, 68]. Note that the use of $\beta < 1$, has a beneficial effect on the error propagation [30, 34]. In Ref. [58], it is shown that for $\beta = 1$ the error propagation mechanism is of a random walk type, hence the algorithm is unstable. In Ref. [5], it is pointed out that due to numerical errors the term $\frac{1}{\beta + \mathbf{x}_n^T P_{n-1} \mathbf{x}_n}$ may become negative, leading to divergence. The numerical performance of the RLS becomes a more serious concern in implementations using limited precision, such as fixed point arithmetic. Compared to the LMS, RLS would require the use of higher precision implementations; otherwise, divergence may occur after a few iteration steps. This adds further to its computational disadvantage compared to the LMS.
- The choice of λ in the initialization step has been considered in Ref. [46]. The related theoretical analysis suggests that λ has a direct influence on the convergence speed, and it should be chosen so as to be a small positive for high Signal-to-Noise (SNR) ratios and a large positive constant for low SNRs.
- In Ref. [56], it has been shown that the RLS algorithm can be obtained as a special case of the Kalman filter.
- The main advantage of the RLS is that it converges to the steady state much faster than the LMS and the rest of the members of the LMS family. This can be justified by the fact that the RLS can be seen as an offspring of Newton's iterative optimization method.
- Distributed versions of the RLS have been proposed in Refs. [8, 39, 40].

6.7 NEWTON'S ITERATIVE MINIMIZATION METHOD

The steepest descent formulation was presented in [Chapter 5](#). It was noted that it exhibits a linear convergence rate and a heavy dependence on the condition number of the Hessian matrix associated with the cost function. Newton's method is a way to overcome this dependence on the condition number and at the same time improve upon the rate of convergence toward the solution.

In [Section 5.2](#), a first order Taylor expansion was used around the current value $J(\boldsymbol{\theta}^{(i-1)})$. Let us now consider a second order expansion (assume $\mu_i = 1$),

$$\begin{aligned} J(\boldsymbol{\theta}^{(i-1)} + \Delta\boldsymbol{\theta}^{(i)}) &= J(\boldsymbol{\theta}^{(i-1)}) + (\nabla J(\boldsymbol{\theta}^{(i-1)}))^T \Delta\boldsymbol{\theta}^{(i)} \\ &\quad + \frac{1}{2} (\Delta\boldsymbol{\theta}^{(i)})^T \nabla^2 J(\boldsymbol{\theta}^{(i-1)}) \Delta\boldsymbol{\theta}^{(i)}. \end{aligned}$$

Assuming $\nabla^2 J(\theta^{(i-1)})$ to be positive definite (this is always the case if $J(\theta)$ is a strictly convex function⁴), the above is a convex quadratic function w.r.t. the step $\Delta\theta^{(i)}$; the latter is computed so as to *minimize* the above second order approximation. The minimum results by equating the corresponding gradient to $\mathbf{0}$, which results in

$$\Delta\theta^{(i)} = -\left(\nabla^2 J(\theta^{(i-1)})\right)^{-1} \nabla J(\theta^{(i-1)}). \quad (6.47)$$

Note that this is indeed a descent direction, because

$$\nabla^T J(\theta^{(i-1)}) \Delta\theta^{(i)} = -\nabla^T J(\theta^{(i-1)}) \left(\nabla^2 J(\theta^{(i-1)})\right)^{-1} \nabla J(\theta^{(i-1)}) < 0,$$

due to the positive definite nature of the Hessian; equality to zero is achieved only at a minimum. Thus, the iterative scheme takes the following form:

$\theta^{(i)} = \theta^{(i-1)} - \mu_i \left(\nabla^2 J(\theta^{(i-1)})\right)^{-1} \nabla J(\theta^{(i-1)})$: Newton's Iterative Scheme.

(6.48)

Figure 6.5 illustrates the method.

Note that if the cost function is quadratic, then the minimum is achieved at the first iteration!

Observe that in the case of Newton's algorithm, the correction direction is not that of 180° with respect to $\nabla J(\theta^{(i-1)})$, as it is the case for the steepest descent method. An alternative point of view is to look at (6.48) as the steepest descent direction under the following norm (see [Section 5.2](#))

$$\|\mathbf{v}\|_P = (\mathbf{v}^T P \mathbf{v})^{1/2},$$

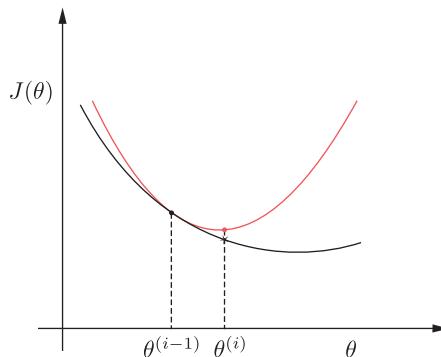
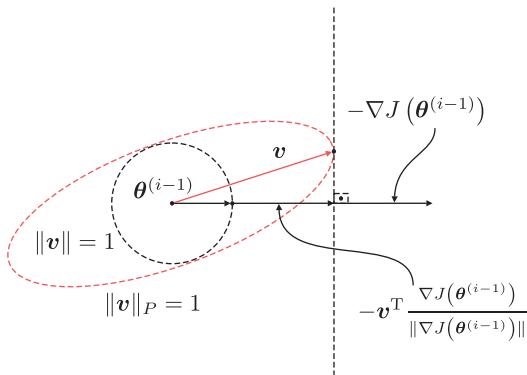


FIGURE 6.5

According to Newton's method, a local quadratic approximation of the cost function is considered (red curve), and the correction pushes the new estimate toward the minimum of this approximation. If the cost function is quadratic, then convergence can be achieved in one step.

⁴ See [Chapter 8](#) for related definitions.

**FIGURE 6.6**

The graphs of the unit Euclidean (black circle) and quadratic (red ellipse) norms centered at $\theta^{(i-1)}$ are shown. In both cases, the goal is to move as far as possible in the direction of $-\nabla J(\theta^{(i-1)})$, while remaining at the ellipse (circle). The result is different for the two cases. The Euclidean norm corresponds to the steepest descent and the quadratic norm to Newton's method.

where P is a symmetric positive definite matrix. For our case, we set

$$P = \nabla^2 J(\theta^{(i-1)}).$$

Then, searching for the respective normalized steepest descent direction, i.e.,

$$\begin{aligned} v &= \arg \min_z z^T \nabla J(\theta^{(i-1)}) \\ \text{s.t. } \|z\|_P^2 &= 1, \end{aligned}$$

results to the normalized vector pointing in the same direction as the one in (6.47) (Problem 6.15). For $P = I$, the gradient descent algorithm results. The geometry is illustrated in Figure 6.6. Note that Newton's direction accounts for the *local shape* of the cost function.

The convergence rate for Newton's method is, in general, high and it becomes quadratic close to the solution. Assuming θ_* to be the minimum, quadratic convergence means that at each iteration, i , the deviation from the optimum value follows the pattern:

$$\ln \ln \frac{1}{\|\theta^{(i)} - \theta_*\|^2} \propto i : \quad \text{Quadratic Convergence Rate.}$$

(6.49)

In contrast, for the linear convergence, the iterations approach the optimal according to:

$$\ln \frac{1}{\|\theta^{(i)} - \theta_*\|^2} \propto i : \quad \text{Linear Convergence Rate.}$$

(6.50)

Furthermore, the presence of the Hessian in the correction term remedies, to a large extent, the influence of the condition number of the Hessian matrix on the convergence [6] (Problem 6.16).

6.7.1 RLS AND NEWTON'S METHOD

The RLS algorithm can be rederived following Newton's iterative scheme applied to the MSE and adopting *stochastic approximation* arguments. Let

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}[(y - \boldsymbol{\theta}^T \mathbf{x})^2] = \frac{1}{2} \sigma_y^2 + \frac{1}{2} \boldsymbol{\theta}^T \Sigma_x \boldsymbol{\theta} - \frac{1}{2} \boldsymbol{\theta}^T \mathbf{p},$$

or

$$-\nabla J(\boldsymbol{\theta}) = [\mathbf{p} - \Sigma_x \boldsymbol{\theta}] = \mathbb{E}[\mathbf{x}\mathbf{e}],$$

and

$$\nabla^2 J(\boldsymbol{\theta}) = \Sigma_x.$$

Newton's iteration becomes

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \Sigma_x^{-1} \mathbb{E}[\mathbf{x}\mathbf{e}].$$

Following stochastic approximation arguments and replacing iteration steps with time updates and expectations with observations, we obtain

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \Sigma_x^{-1} \mathbf{x}_n e_n.$$

Let us now adopt the approximation,

$$\Sigma_x \simeq \frac{1}{n+1} \Phi_n = \left(\frac{1}{n+1} \lambda \beta^{n+1} I + \frac{1}{n+1} \sum_{i=0}^n \beta^{n-i} \mathbf{x}_i \mathbf{x}_i^T \right),$$

and set

$$\mu_n = \frac{1}{n+1}.$$

Then

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + K_n e_n,$$

with

$$K_n = P_n \mathbf{x}_n,$$

where

$$P_n = \left(\sum_{i=0}^n \beta^{n-i} \mathbf{x}_i \mathbf{x}_i^T + \lambda \beta^{n+1} I \right)^{-1},$$

which then, by using similar steps as for (6.40)–(6.42) leads to the RLS scheme.

Note that this point of view justifies the fast converging properties of the RLS and its relative insensitivity to the condition number of the covariance matrix.

Remarks 6.3.

- When dealing with the LMS in Chapter 5, we saw that LMS is optimal with respect to a min/max robustness criterion. However, this is not true for the RLS. It turns out that while LMS exhibits the best worst case performance, the RLS is expected to have better performance on average [23].

6.8 STEADY-STATE PERFORMANCE OF THE RLS

Compared to the stochastic gradient techniques, which were considered in [Chapter 5](#), we do not have to worry whether RLS converges and where it converges. The RLS computes the *exact* solution of the minimization task in [\(6.36\)](#) in an iterative way. Asymptotically and for $\beta = 1$ ($\lambda = 0$) solves the MSE optimization task. However, we have to consider its steady state performance for $\beta \neq 1$. Even for the stationary case, $\beta \neq 1$ results in an excess mean-square-error. Moreover, it is important to get a feeling of its tracking performance in time-varying environments. To this end, we adopt the same setting as the one followed in [Section 5.12](#). We will not provide all the details of the proof, because this follows similar steps as in the LMS case. We will point out where differences arise and state the results. For the detailed derivation, the interested reader may consult [[15](#), [48](#), [57](#)]; in the latter one, the energy conservation theory is employed.

As in [Chapter 5](#), we adopt the following models

$$\mathbf{y}_n = \boldsymbol{\theta}_{o,n-1}^T \mathbf{x}_n + \boldsymbol{\eta}_n, \quad (6.51)$$

and

$$\boldsymbol{\theta}_{o,n} = \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}_n, \quad (6.52)$$

with

$$\mathbb{E}[\boldsymbol{\omega}_n \boldsymbol{\omega}_n^T] = \boldsymbol{\Sigma}_\omega.$$

Hence, taking into account [\(6.51\)](#), [\(6.52\)](#), and the RLS iteration involving the respective random variables, we get

$$\boldsymbol{\theta}_n - \boldsymbol{\theta}_{o,n} = \boldsymbol{\theta}_{n-1} + \mathbf{K}_n \mathbf{e}_n - \boldsymbol{\theta}_{o,n-1} - \boldsymbol{\omega}_n,$$

or

$$\begin{aligned} \mathbf{c}_n &:= \boldsymbol{\theta}_n - \boldsymbol{\theta}_{o,n} = \mathbf{c}_{n-1} + \mathbf{P}_n \mathbf{x}_n \mathbf{e}_n - \boldsymbol{\omega}_n \\ &= (\mathbf{I} - \mathbf{P}_n \mathbf{x}_n \mathbf{x}_n^T) \mathbf{c}_{n-1} + \mathbf{P}_n \mathbf{x}_n \boldsymbol{\eta}_n - \boldsymbol{\omega}_n, \end{aligned}$$

which is the counterpart of [\(5.78\)](#); Note that the time indexes for the input and the noise variables can be dropped out, because their statistics is assumed to be time invariant.

We adopt the same assumptions as in [Section 5.12](#). In addition, we assume that \mathbf{P}_n is changing slowly compared to \mathbf{c}_n . Hence, every time \mathbf{P}_n appears inside an expectation, it is substituted by its mean $\mathbb{E}[\mathbf{P}_n]$, namely,

$$\mathbb{E}[\mathbf{P}_n] = \mathbb{E}[\Phi_n^{-1}],$$

where

$$\Phi_n = \lambda \beta^{n+1} \mathbf{I} + \sum_{i=0}^n \beta^{n-i} \mathbf{x}_i \mathbf{x}_i^T,$$

and

$$\mathbb{E}[\Phi_n] = \lambda \beta^{n+1} \mathbf{I} + \frac{1 - \beta^{n+1}}{1 - \beta} \boldsymbol{\Sigma}_x.$$

Assuming $\beta \simeq 1$, the variance at the steady state of Φ_n can be considered small and we can adopt the following approximation,

Table 6.1 The Steady-State Excess MSE, for Small Values of μ and β

Algorithm	Excess MSE, J_{exc} , at Steady-state
LMS	$\frac{1}{2}\mu\sigma_\eta^2\text{trace}\{\Sigma_x\} + \frac{1}{2}\mu^{-1}\text{trace}\{\Sigma_\omega\}$
APA	$\frac{1}{2}\mu\sigma_\eta^2\text{trace}\{\Sigma_x\}\mathbb{E}\left[\frac{q}{\ x\ ^2}\right] + \frac{1}{2}\mu^{-1}\text{trace}\{\Sigma_x\}\text{trace}\{\Sigma_\omega\}$
RLS	$\frac{1}{2}(1-\beta)\sigma_\eta^2l + \frac{1}{2}(1-\beta)^{-1}\text{trace}\{\Sigma_\omega\Sigma_x\}$

For $q = 1$, the normalized LMS results. Under a Gaussian input assumption and for long system orders, l , in the APA, $\mathbb{E}\left[\frac{q}{\|x\|^2}\right] \simeq \frac{q}{\sigma_x^2(l-2)}$ [11].

$$\mathbb{E}[P_n] \simeq [\mathbb{E}[\Phi_n]]^{-1} = \left[\beta^{n+1}\lambda I + \frac{1-\beta^{n+1}}{1-\beta} \Sigma_x \right]^{-1}.$$

Based on all the previously stated assumptions, then repeating carefully the same steps as in [Section 5.12](#), we end up with the result shown in [Table 6.1](#), which holds for small values of β . For comparison reasons, the excess MSE is shown together with the values obtained for the LMS as well as the APA algorithms. In stationary environments, one simply sets $\Sigma_\omega = 0$. According to [Table 6.1](#), the following remarks are in order:

Remarks 6.4.

- For stationary environments, the performance of the RLS is independent of Σ_x . Of course, if one knows that the environment is stationary, then ideally $\beta = 1$ should be the choice. Yet recall that for $\beta = 1$, the algorithm has stability problems.
- Note that for small μ and $\beta \simeq 1$, there is an “equivalence” of $\mu \simeq 1 - \beta$, for the two parameters in the LMS and RLS. That is, larger values of μ are beneficial to the tracking performance of LMS, while smaller values of β are required for faster tracking; this is expected because the algorithm forgets the past.
- It is clear from [Table 6.1](#) that an algorithm may converge to the steady-state quickly, but it may not necessarily track fast. It all depends on the specific scenario. For example, under the modeling assumptions associated with [Table 6.1](#), the optimal value μ_{opt} for the LMS ([Section 5.12](#)) is given by

$$\mu_{\text{opt}} = \sqrt{\frac{\text{trace}\{\Sigma_\omega\}}{\sigma_\eta^2\text{trace}\{\Sigma_x\}}},$$

which corresponds to

$$J_{\min}^{\text{LMS}} = \sqrt{\sigma_\eta^2\text{trace}\{\Sigma_x\}\text{trace}\{\Sigma_\omega\}}.$$

Optimizing with respect to β for the RLS, it is easily shown that

$$\beta_{\text{opt}} = 1 - \sqrt{\frac{\text{trace}\{\Sigma_\omega\Sigma_x\}}{\sigma_\eta^2l}},$$

$$J_{\min}^{\text{RLS}} = \sqrt{\sigma_\eta^2l\text{trace}\{\Sigma_\omega\Sigma_x\}}.$$

Hence, the ratio

$$\frac{J_{\min}^{\text{LMS}}}{J_{\min}^{\text{RLS}}} = \sqrt{\frac{\text{trace}\{\Sigma_x\}\text{trace}\{\Sigma_\omega\}}{l\text{trace}\{\Sigma_\omega\Sigma_x\}}},$$

depends on Σ_ω and Σ_x . Sometimes LMS tracks better, yet in other problems RLS is the winner. Having said that, it must be pointed out that the RLS always converges faster, and the difference in the rate, compared to the LMS, increases with the condition number of the input covariance matrix.

6.9 COMPLEX-VALUED DATA: THE WIDELY LINEAR RLS

Following similar arguments as in [Section 5.7](#), let

$$\boldsymbol{\varphi} = \begin{bmatrix} \boldsymbol{\theta} \\ v \end{bmatrix}, \quad \tilde{\mathbf{x}}_n = \begin{bmatrix} \mathbf{x}_n \\ \mathbf{x}_n^* \end{bmatrix},$$

with

$$\hat{y}_n = \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n.$$

The least-squares regularized cost becomes

$$J(\boldsymbol{\varphi}) = \sum_{i=0}^n \beta^{n-i} (y_n - \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n) (y_n - \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n)^* + \lambda \beta^{n+1} \boldsymbol{\varphi}^H \boldsymbol{\varphi},$$

or

$$\begin{aligned} J(\boldsymbol{\varphi}) &= \sum_{i=0}^n \beta^{n-i} |y_n|^2 + \sum_{i=0}^n \beta^{n-i} \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^H \boldsymbol{\varphi} \\ &\quad - \sum_{i=0}^n \beta^{n-i} y_n \tilde{\mathbf{x}}_n^H \boldsymbol{\varphi} - \sum_{i=0}^n \beta^{n-i} \boldsymbol{\varphi}^H \tilde{\mathbf{x}}_n y_n^* + \lambda \beta^{n+1} \boldsymbol{\varphi}^H \boldsymbol{\varphi}. \end{aligned}$$

Taking the gradient with respect to $\boldsymbol{\varphi}^*$ and equating to zero, we obtain

$$\boxed{\tilde{\Phi}_n \boldsymbol{\varphi}_n = \tilde{\mathbf{p}}_n : \quad \text{Widely Linear LS Estimate,}} \quad (6.53)$$

where

$$\tilde{\Phi}_n = \beta^{n+1} \lambda I + \sum_{i=0}^n \beta^{n-i} \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^H, \quad (6.54)$$

$$\tilde{\mathbf{p}}_n = \sum_{i=0}^n \beta^{n-i} \tilde{\mathbf{x}}_n y_n^*. \quad (6.55)$$

Following similar steps as for the real-valued RLS, the [Algorithm 6.2](#) results, where $\tilde{P}_n := \tilde{\Phi}^{-1}$.

Algorithm 6.2 .[(The widely linear RLS algorithm)]

- Initialize
 - $\varphi_0 = \mathbf{0}$
 - $\tilde{P}_{-1} = \lambda^{-1}I$
 - Select β
- **For** $n = 0, 1, 2, \dots$, **Do**
 - $e_n = y_n - \varphi_{n-1}^H \tilde{x}_n$
 - $z_n = \tilde{P}_{n-1} \tilde{x}_n$
 - $K_n = \frac{z_n}{\beta + \tilde{x}_n^H z_n}$
 - $\varphi_n = \varphi_{n-1} + K_n e_n^*$
 - $\tilde{P}_n = \beta^{-1} \tilde{P}_{n-1} - \beta^{-1} K_n z_n^H$
- **End For**

Setting $v_n = 0$ and replacing \tilde{x}_n with x_n and φ_n with θ_n , the linear complex-valued RLS results.

6.10 COMPUTATIONAL ASPECTS OF THE LS SOLUTION

The literature concerning the efficient solution of the least-squares equation as well as the computationally efficient implementation of the RLS is huge. In this section, we will only highlight some of the basic directions that have been followed over the years. Most of the available software packages are implementing such efficient schemes.

A major direction in developing various algorithmic schemes was to cope with the numerical stability issues, as we have already discussed in [Remarks 6.2](#). The main concern is to guarantee the symmetry and positive definiteness of Φ_n . The path followed toward this end is to work with the square root factors of Φ_n .

Cholesky factorization

It is known from linear algebra that every positive definite symmetric matrix, such as Φ_n , accepts the following factorization

$$\Phi_n = L_n L_n^T,$$

where L_n is lower-triangular with positive entries along its diagonal. Moreover, this factorization is unique.

Concerning our least-squares task, one focuses on updating the factor L_n , instead of Φ_n , in order to improve numerical stability. Computation of the Cholesky factors can be achieved via a modified version of the Gauss elimination scheme [22].

QR factorization

A better option for computing square factors of a matrix, from a numerical stability point of view, is via the QR decomposition method. To simplify the discussion, let us consider $\beta = 1$ and $\lambda = 0$ (no regularization). Then the positive definite (sample) covariance matrix can be factored as

$$\Phi_n = U_n^T U_n.$$

From linear algebra, [22], we know that the $(n + 1) \times l$ matrix U_n can be written as a product

$$U_n = Q_n R_n,$$

where Q_n is a $(n + 1) \times (n + 1)$ orthogonal matrix and R_n is a $(n + 1) \times l$ upper triangular matrix. Note that R_n is related to the Cholesky factor L_n^T . It turns out that working with the QR factors of U_n is preferable, with respect to numerical stability, to working on the Cholesky factorization of Φ_n . QR factorization can be achieved via different paths:

- *Gram-Schmidt* orthogonalization of the input matrix columns. We have seen this path in Chapter 4 while discussing the lattice-ladder algorithm for solving the normal equations for the filtering case. Under the time shift property of the input signal, lattice-ladder-type algorithms have also been developed for the least-squares filtering task [31, 32].
- *Givens rotations*: This has also been a popular line [10, 41, 52, 54].
- *Householder reflections*: This line has been followed in Refs. [53, 55]. The use of Householder reflections leads to a particularly robust scheme from a numerical point of view. Moreover, the scheme presents a high degree of parallelism, which can be exploited appropriately in a parallel processing environment.

A selection of related to QR factorization review papers is given in Ref. [2].

Fast RLS versions

Another line of intense activity, especially in the 1980s, was that of exploiting the special structure associated with the filtering task; that is, the input to the filter comprises the samples from a realization of a random signal/process. Abiding by our adopted notational convention, the input vector will now be denoted as \mathbf{u} instead of \mathbf{x} . Also, for the needs of the discussion we will bring into the notation the order of the filter, m . In this case, the input vectors (regressors), at two successive time instants, share all but two of their components. Indeed, for an m th order system, we have that

$$\mathbf{u}_{m,n} = \begin{bmatrix} u_n \\ \vdots \\ u_{n-m+1} \end{bmatrix}, \quad \mathbf{u}_{m,n-1} = \begin{bmatrix} u_{n-1} \\ \vdots \\ u_{n-m} \end{bmatrix},$$

and we can partition the input vector as

$$\mathbf{u}_{m,n} = [u_n \ \mathbf{u}_{m-1,n-1}]^T = [\mathbf{u}_{m-1,n}, \ u_{n-m+1}]^T.$$

This property is also known as *time-shift* structure. Such a partition of the input vector leads to

$$\begin{aligned} \Phi_{m,n} &= \begin{bmatrix} \sum_{i=0}^n u_i^2 & \sum_{i=0}^n \mathbf{u}_{m-1,i-1}^T \mathbf{u}_i \\ \sum_{i=0}^n \mathbf{u}_{m-1,i-1} \mathbf{u}_i & \Phi_{m-1,n-1} \end{bmatrix} \\ &= \begin{bmatrix} \Phi_{m-1,n} & \sum_{i=0}^n \mathbf{u}_{m-1,i} \mathbf{u}_{i-m+1} \\ \sum_{i=0}^n \mathbf{u}_{m-1,i}^T \mathbf{u}_{i-m+1} & \sum_{i=0}^n u_{i-m+1}^2 \end{bmatrix}, \quad m = 2, 3, \dots, l \end{aligned} \tag{6.56}$$

where for complex variables transposition is replaced by the Hermitian one. Compare (6.56) with (4.60). The two partitions look alike, yet they are different. Matrix $\Phi_{m,n}$ is no longer Toeplitz. Its low partition is given in terms of $\Phi_{m-1,n-1}$. Such matrices are known as *near-to-Toeplitz*. All that is needed is to “correct” $\Phi_{m-1,n-1}$ back to $\Phi_{m-1,n}$ subtracting a rank one matrix, i.e.,

$$\Phi_{m-1,n-1} = \Phi_{m-1,n} - \mathbf{u}_{m-1,n} \mathbf{u}_{m-1,n}^T.$$

It turns out that such corrections, although they may slightly complicate the derivation, can still lead to computational efficient order recursive schemes, via the application of the matrix inversion lemma, as was the case in the MSE of Section 4.8. Such schemes have their origin in the pioneering PhD thesis of Martin Morf at Stanford [42]. Levinson-type, Schur-type, split-Levinson type, lattice-ladder algorithms have been derived for the least-squares case [3, 27, 28, 43, 44, 60, 61]. Some of the schemes noted previously under the *QR* factorization exploit the time-shift structure of the input signal.

Besides the order recursive schemes, a number of fixed order fast RLS-type schemes have been developed following the work in Ref. [33]. Recall from the definition of the Kalman gain in (6.43) that for an l th order system we have

$$\begin{aligned} K_{l+1,n} &= \Phi_{l+1,n}^{-1} \mathbf{u}_{l+1,n} = \begin{bmatrix} * & * \\ * & \Phi_{l,n-1} \end{bmatrix}^{-1} \begin{bmatrix} * \\ \mathbf{u}_{l,n-1} \end{bmatrix} \\ &= \begin{bmatrix} \Phi_{l,n} & * \\ * & * \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{u}_{l,n} \\ * \end{bmatrix}, \end{aligned}$$

where $*$ denotes any value of the element. Without going into detail, the low partition can relate the Kalman gain of order l and time $n-1$ to the Kalman gain of order $l+1$ and time n (step up). Then the upper partition can be used to obtain the time update Kalman gain at order l and time n (step down). Such a procedure bypasses the need for matrix operations leading to $\mathcal{O}(l)$ RLS type algorithms [7, 9] with complexity $7l$ per time update. However, these versions turned out to be numerically unstable. Numerically stabilized versions, at only a small extra computational cost, were proposed in Refs. [5, 58]. All the aforementioned schemes have also been developed for solving the (regularized) exponentially weighted LS cost function.

Besides this line, variants that obtain approximate solutions have been derived in an attempt to reduce complexity; these schemes use an approximation of the covariance or inverse covariance matrix [14, 38]. The *fast Newton transversal filter* (FNTF) algorithm [45], approximates the inverse covariance matrix by a banded matrix of width p . Such a modeling has a specific physical interpretation. A banded inverse covariance matrix corresponds to an AR process of order p . Hence, if the input signal can sufficiently be modeled by an AR model, FNTF obtains a least-squares performance. Moreover, this performance is obtained at $\mathcal{O}(p)$ instead of $\mathcal{O}(l)$ computational cost. This can be very effective in applications where $p \ll l$. This is the case, for example, in audio conferencing, where the input signal is speech. Speech can efficiently be modeled by an AR of the order of 15, yet the filter order can be of a few hundred taps [49]. FNTF bridges the gap between LMS ($p=1$) and (fast) RLS ($p=l$). Moreover, FNTF builds upon the structure of the stabilized fast RLS. More recently, the banded inverse covariance matrix approximation has been successfully applied in spectral analysis [21].

More on the efficient least-squares schemes can be found in Refs. [15, 17, 20, 24, 29, 57].

6.11 THE COORDINATE AND CYCLIC COORDINATE DESCENT METHODS

So far, we have discussed the steepest descent and Newton's method for optimization. We will conclude the discussion with a third method, which can also be seen as a member of the steepest descent family of methods. Instead of the Euclidean and quadratic norms, let us consider the following minimization task for obtaining the normalized descent direction,

$$\mathbf{v} = \arg \min_{\mathbf{z}} \mathbf{z}^T \nabla J, \quad (6.57)$$

$$\text{s.t. } \|\mathbf{z}\|_1 = 1, \quad (6.58)$$

where $\|\cdot\|_1$ denotes the ℓ_1 norm, defined as

$$\|\mathbf{z}\|_1 := \sum_{i=1}^l |z_i|.$$

Most of [Chapter 9](#) is dedicated to this norm and its properties. Observe that this is not differentiable. Solving the minimization task ([Problem 6.17](#)) results in,

$$\mathbf{v} = -\operatorname{sgn}((\nabla J)_k) \mathbf{e}_k,$$

where \mathbf{e}_k is the direction of the coordinate corresponding to the component $(\nabla J)_k$ with the largest absolute value, i.e.,

$$|(\nabla J)_k| > |(\nabla J)_j|, \quad j \neq k,$$

and $\operatorname{sgn}(\cdot)$ is the sign function. The geometry is illustrated in [Figure 6.7](#). In other words, the descent direction is along a single basis vector; that is, each time only a *single component* of θ is updated. It is the component that corresponds to the directional derivative, $(\nabla J(\theta^{(i-1)}))_k$, with the largest increase and the update rule becomes

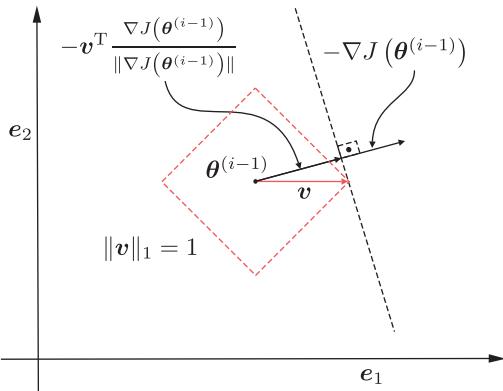


FIGURE 6.7

The unit norm $\|\cdot\|_1$ ball centered at $\theta^{(i-1)}$ is a rhombus (in \mathbb{R}^2). The direction \mathbf{e}_1 is the one corresponding to the largest component of ∇J . Recall that the components of the vector ∇J are the respective directional derivatives.

$$\theta_k^{(i)} = \theta_k^{(i-1)} - \mu_i \frac{\partial J(\boldsymbol{\theta}^{(i-1)})}{\partial \theta_k} : \text{ Coordinate-Descent Scheme} \quad (6.59)$$

$$\theta_j^{(i)} = \theta_j^{(i-1)}, \quad j = 1, 2, \dots, l, \quad j \neq k. \quad (6.60)$$

Because only one component is updated at each iteration, this greatly simplifies the update mechanism. The method is known as *Coordinate Descent* (CD).

Based on this rationale, a number of variants of the basic coordinate descent have been proposed. The *Cyclic Coordinate Descent* (CCD) in its simplest form entails a *cyclic* update with respect to one coordinate per iteration cycle; that is, at the i th iteration the following minimization is solved:

$$\theta_k^{(i)} := \arg \min_{\theta} J(\theta_1^{(i)}, \dots, \theta_{k-1}^{(i)}, \theta, \theta_{k+1}^{(i-1)}, \dots, \theta_l^{(i-1)}).$$

In words, all components but θ_k are assumed constant; those components θ_j , $j < k$ are fixed to their updated values, $\theta_j^{(i)}$, $j = 1, 2, \dots, k-1$, and the rest, θ_j , $j = k+1, \dots, l$, to the available estimates, $\theta_j^{(i-1)}$, from the previous iteration. The nice feature of such a technique is that a simple close form solution for the minimizer may be obtained. A revival of such techniques has happened in the context of sparse learning models ([Chapter 10](#)) [18, 67]. Convergence issues of CCD have been considered in Ref. [36, 62]. CCD algorithms for the LS task have also been considered [66] and the references therein. Besides the basic CCD scheme, variants are also available, using different scenarios for the choice of the direction to be updated each time, in order to improve convergence, ranging from a random choice to a change of the coordinate systems, which is known as an *adaptive coordinate descent* scheme [35].

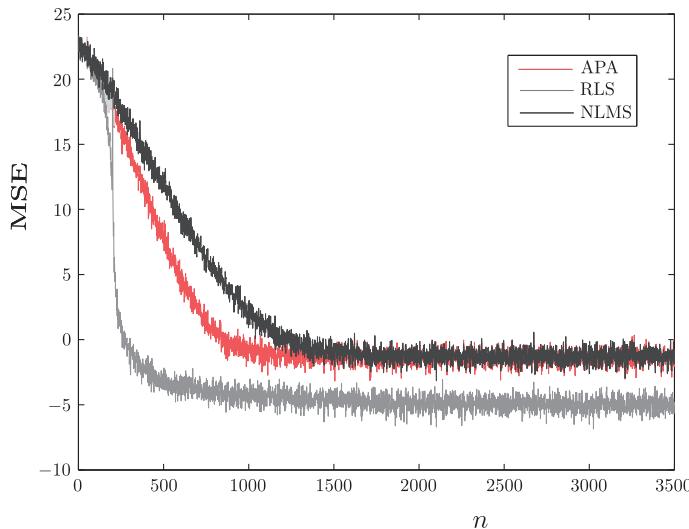
6.12 SIMULATION EXAMPLES

In this section, simulation examples are presented concerning the convergence and tracking performance of the RLS compared to algorithms of the gradient descent family, which have been derived in [Chapter 5](#).

Example 6.1. The focus of this example is to demonstrate the comparative performance, with respect to the convergence rate of the RLS, the NLMS, and the APA algorithms, which have been discussed in [Chapter 5](#). To this end, we generate data according to the regression model

$$y_n = \boldsymbol{\theta}_o^T \mathbf{x}_n + \eta_n,$$

where $\boldsymbol{\theta}_o \in \mathbb{R}^{200}$. Its elements are generated randomly according to the normalized Gaussian. The noise samples are i.i.d. generated via the zero mean Gaussian with variance equal to $\sigma_\eta^2 = 0.01$. The elements of the input vector are also i.i.d. generated via the normalized Gaussian. Using the generated samples (y_n, \mathbf{x}_n) , $n = 0, 1, \dots$, as the training sequence for all three previously stated algorithms, the convergence curves of [Figure 6.8](#) are obtained. The curves show the squared error in dBs ($10 \log_{10}(e_n^2)$), averaged over 100 different realizations of the experiments, as a function of the time index n . The parameters used for the involved algorithms are: (a) For the NLMS, we used $\mu = 1.2$ and $\delta = 0.001$; (b) for the APA, we used $\mu = 0.2$, $\delta = 0.001$, and $q = 30$, and (c) for the RLS, $\beta = 1$ and $\lambda = 0.1$. The parameters for the NLMS and the APA were chosen so that both algorithms converge to the same error floor. The improved performance of the APA concerning the convergence rate compared to the NLMS is readily seen. However, both algorithms fall short when compared to the RLS. Note that the

**FIGURE 6.8**

MSE curves as a function of the number of iterations for NLMS, APA, and RLS. The RLS converges faster and at lower error floor.

RLS converges to lower error floor, because no forgetting factor was used. To be consistent, a forgetting factor $\beta < 1$ should have been used in order for this algorithm to settle at the same error floor as the other two algorithms; this would have a beneficial effect on the convergence rate. However, having chosen $\beta = 1$, is demonstrated that the RLS can converge really fast, even to lower error floors. However, this improved performance is obtained at substantial higher complexity. In case the input vector is part of a random process, and the special time-shift structure can be exploited, as discussed in [Section 6.10](#), the lower complexity versions are at the disposal of the designer. A further comparative performance example, including another family of online algorithms, will be given in [Chapter 8](#).

However, it has to be stressed that this notable advantage (between RLS and LMS-type schemes) in convergence speed, from the initial conditions to steady-state may not be the case concerning the tracking performance, when the algorithms have to track time varying environments. This is demonstrated next.

Example 6.2. This example focuses on the comparative tracking performance of the RLS and NLMS. Our goal is to demonstrate some cases where the RLS fails to do as well as the NLMS. Of course, it must be kept in mind that according to the theory, the comparative performance is very much dependent on the specific application.

For the needs of our example, let us mobilize the time varying model of the parameters given in [\(6.52\)](#) in its more practical version and generate the data according to the following linear system

$$y_n = \mathbf{x}_n^T \boldsymbol{\theta}_{o,n-1} + \eta_n, \quad (6.61)$$

where

$$\boldsymbol{\theta}_{o,n} = \alpha \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}_n,$$

with $\boldsymbol{\theta}_{o,n} \in \mathbb{R}^5$. It turns out that such a time varying model is closely related (for the right choice of the involved parameters) to what is known in communications as a Rayleigh fading channel, if the

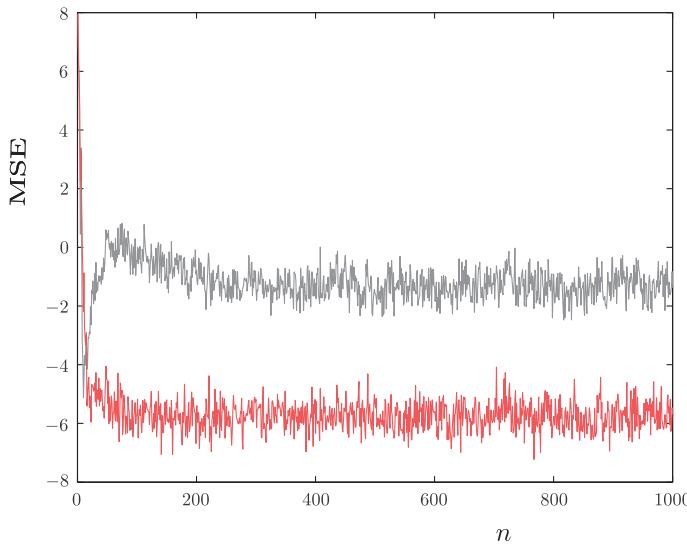


FIGURE 6.9

For a fast time varying parameter model, the RLS (gray) fails to track it, in spite of its very fast initial convergence, compared to the NLMS (red).

parameters comprising $\theta_{o,n}$ are thought to represent the impulse response of such a channel [57]. Rayleigh fading channels are very common and can adequately model a number of transmission channels in wireless communications. Playing with the parameter α and the variance of the corresponding noise source, ω , one can achieve fast or slow time varying scenarios. In our case, we chose $\alpha = 0.97$ and the noise followed a Gaussian distribution of zero mean and covariance matrix $\Sigma_\omega = 0.1I$.

Concerning the data generation, the input samples were generated i.i.d. from a Gaussian $\mathcal{N}(0, 1)$, and the noise was also Gaussian of zero mean value and variance equal to $\sigma_\eta^2 = 0.01$. Initialization of the time varying model ($\theta_{o,0}$) was randomly done by drawing samples from a $\mathcal{N}(0, 1)$.

Figure 6.9 shows the obtained MSE curve as a function of the iterations, for the NLMS and the RLS. For the RLS, the forgetting factor was set equal to $\beta = 0.995$ and for the NLMS, $\mu = 0.5$ and $\delta = 0.001$. Such a choice resulted in the best performance, for both algorithms, after extensive experimentation. The curves are the result of averaging out 200 independent runs.

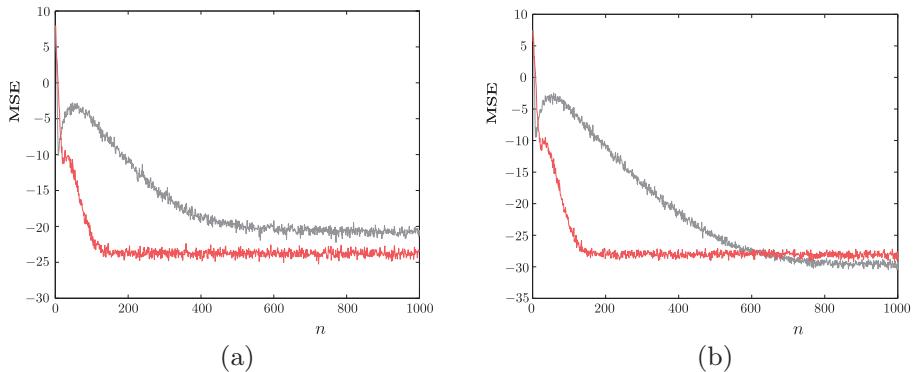
Figure 6.10 shows the resulting curves for medium and slow time varying channels, corresponding to $\Sigma_\omega = 0.01I$ and $\Sigma_\omega = 0.001I$, respectively.

6.13 TOTAL-LEAST-SQUARES

In this section, the least-squares optimization task will be formulated from a different perspective. Assume zero mean (centered) data and our familiar linear regression model, employing the observed samples,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta},$$

as in Section 6.2. We have seen that the least-squares task is equivalent with (orthogonally) projecting \mathbf{y} onto the span $\{\mathbf{x}_1^c, \dots, \mathbf{x}_l^c\}$ of the columns of \mathbf{X} , hence, making the error

**FIGURE 6.10**

MSE curves as a function of iteration for (a) a medium and (b) a slow time varying parameter model. The red curve corresponds to the NLMS and the gray one to the RLS.

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$$

orthogonal to the column space of X . Equivalently, this can be written as

$$\begin{aligned} & \text{minimize} \quad \|\mathbf{e}\|^2, \\ & \text{s.t.} \quad \mathbf{y} - \mathbf{e} \in \mathcal{R}(X), \end{aligned} \quad (6.62)$$

where $\mathcal{R}(X)$ is the range space of X (see [Remarks 6.1](#) for the respective definition). Moreover, once $\hat{\theta}_{\text{LS}}$ has been obtained, we can write that

$$\hat{\mathbf{y}} = X\hat{\theta}_{\text{LS}} = \mathbf{y} - \mathbf{e}$$

or

$$[X : \mathbf{y} - \mathbf{e}] \begin{bmatrix} \hat{\theta}_{\text{LS}} \\ -1 \end{bmatrix} = \mathbf{0}, \quad (6.63)$$

where $[X : \mathbf{y} - \mathbf{e}]$ is the matrix that results after extending X by an extra column, $\mathbf{y} - \mathbf{e}$. Thus, all the points $(y_n - e_n, \mathbf{x}_n) \in \mathbb{R}^{l+1}$, $n = 1, 2, \dots, N$, lie on the same hyperplane, crossing the origin, as shown in [Figure 6.11](#). In other words, in order to fit a hyperplane to the data, the LS method applies a correction e_n , $n = 1, 2, \dots, N$, to the *output samples*. Thus, we have silently assumed that the regressors have been obtained via exact measurements and that the noise affects *only* the output observations.

In this section, the more general case will be considered, where we allow for both the input (regressors) as well as the output variables to be perturbed by (unobserved) noise samples. Such a treatment has a long history, dating back to the nineteenth century [1]. The method remained in obscurity until it was revived 50 years later for two-dimensional models by Deming [13] and it is sometimes known as *Deming regression*; see also [19] for a historical overview. Such models are also known as *errors-in-variables* regression models.

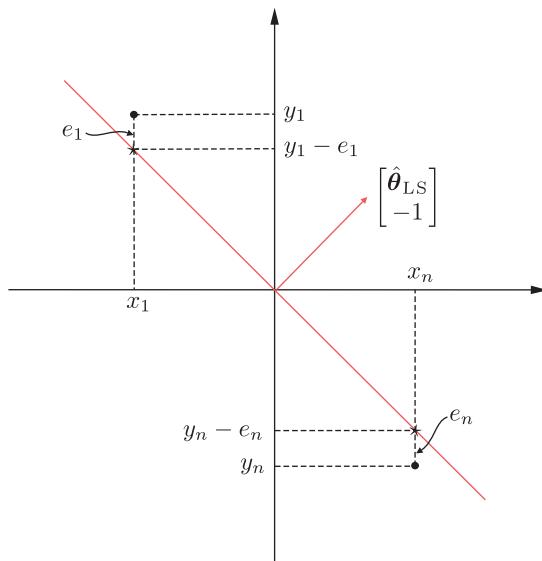


FIGURE 6.11

According to the least-squares method, *only* the output points y_n are corrected to $y_n - e_n$, so that the pairs $(y_n - e_n, x_n)$ lie on a hyperplane, crossing the origin for centered data. If the data are not centered, it crosses the centroid, (\bar{y}, \bar{x}) .

Our kickoff point is the formulation in (6.62). Let e be the correction vector to be applied on y and E the correction matrix to be applied on X . The method of *total-least-squares* computes the unknown parameter vector by solving the following optimization task,

$$\begin{aligned} & \text{minimize} && \| [E : e] \|_F, \\ & \text{s.t.} && y - e \in \mathcal{R}(X - E). \end{aligned} \quad (6.64)$$

Recall (Remarks 6.1) that the Frobenius norm of a matrix is defined as the square root of the sum of squares of all its entries, and it is the direct generalization of the Euclidean norm defined for vectors. Let us first focus on solving the task in (6.64), and we will comment on its geometric interpretation, later on. The set of constraints in (6.64) can equivalently be written as

$$(X - E)\theta = y - e. \quad (6.65)$$

Define

$$F := X - E, \quad (6.66)$$

and let $f_i^T \in \mathbb{R}^l$, $i = 1, 2, \dots, N$, be the rows of F , i.e.,

$$F^T = [f_1, \dots, f_N],$$

and $\mathbf{f}_i^c \in \mathbb{R}^N$, $i = 1, 2, \dots, l$, the respective columns, i.e.,

$$\mathbf{F} = [\mathbf{f}_1^c, \dots, \mathbf{f}_l^c].$$

Let also

$$\mathbf{g} := \mathbf{y} - \mathbf{e}. \quad (6.67)$$

Hence, (6.65) can be written in terms of the columns of \mathbf{F} , i.e.,

$$\boxed{\theta_1 \mathbf{f}_1^c + \dots + \theta_l \mathbf{f}_l^c - \mathbf{g} = 0.} \quad (6.68)$$

Equation (6.68) implies that the $l + 1$ vectors, $\mathbf{f}_1^c, \dots, \mathbf{f}_l^c, \mathbf{g} \in \mathbb{R}^N$, are linearly dependent, which in turn dictates that

$$\boxed{\text{rank}\{[\mathbf{F} : \mathbf{g}]\} \leq l.} \quad (6.69)$$

There is a subtle point here. The opposite is not necessarily true; that is, (6.69) does not necessarily imply (6.68). If the $\text{rank}\{\mathbf{F}\} < l$, there is not, in general, $\boldsymbol{\theta}$ to satisfy (6.68). This can easily be verified, for example, by considering the extreme case where $\mathbf{f}_1^c = \mathbf{f}_2^c = \dots = \mathbf{f}_l^c$. Keeping that in mind, we need to impose some extra assumptions.

Assumptions:

1. The $N \times l$ matrix \mathbf{X} is full rank. This implies that all its singular values are nonzero, and we can write (recall (6.12))

$$\mathbf{X} = \sum_{i=1}^l \sigma_i \mathbf{u}_i \mathbf{v}_i^T,$$

where we have assumed that

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_l > 0. \quad (6.70)$$

2. The $(N \times (l + 1))$ matrix $[\mathbf{X} : \mathbf{y}]$ is also full rank, hence

$$[\mathbf{X} : \mathbf{y}] = \sum_{i=1}^{l+1} \bar{\sigma}_i \bar{\mathbf{u}}_i \bar{\mathbf{v}}_i^T$$

with

$$\bar{\sigma}_1 \geq \bar{\sigma}_2 \geq \dots \geq \bar{\sigma}_{l+1} > 0. \quad (6.71)$$

3. Assume that

$$\bar{\sigma}_{l+1} < \sigma_l.$$

As we will see soon, this guarantees the existence of a unique solution. If this condition is not valid, still solutions can exist; however, this corresponds to a degenerate case and such solutions have been the subject of study in the related literature [37, 64]. However, we will not deal with such cases here. Note that, in general, it can be shown that $\bar{\sigma}_{l+1} \leq \sigma_l$ [26]. Thus, our assumption demands *strict* inequality.

4. Assume that

$$\bar{\sigma}_l > \bar{\sigma}_{l+1}.$$

This condition will also be used in order to guarantee uniqueness of the solution.

We are now ready to solve the following optimization task:

$$\begin{aligned} & \underset{F, g}{\text{minimize}} && \| [X \vdots y] - [F \vdots g] \|_F^2, \\ & \text{s.t.} && \text{rank}\{[F \vdots g]\} = l. \end{aligned}$$

(6.72)

In words, compute the best, in the Frobenius norm sense, rank l approximation, $[F \vdots g]$, to the (rank $l + 1$) matrix $[X \vdots y]$. We know from [Remarks 6.1](#) that

$$[F \vdots g] = \sum_{i=1}^l \bar{\sigma}_i \bar{u}_i \bar{v}_i^T,$$
(6.73)

and consequently

$$[E \vdots e] = \bar{\sigma}_{l+1} \bar{u}_{l+1} \bar{v}_{l+1}^T,$$
(6.74)

with the corresponding Frobenius and spectral norms of the error matrix being equal to

$$\|E \vdots e\|_F = \bar{\sigma}_{l+1} = \|E \vdots e\|_2.$$
(6.75)

Note that the above choice is unique, because $\bar{\sigma}_{l+1} < \bar{\sigma}_l$.

So far, we have uniquely solved the task in [\(6.72\)](#). However, we still have to recover the estimate $\hat{\theta}_{\text{TLS}}$, which will satisfy [\(6.68\)](#). In general, the existence of a unique vector cannot be guaranteed from the F and g given in [\(6.73\)](#). Uniqueness is imposed by assumption (3), which guarantees that the rank of F is equal to l .

Indeed, assume that the rank of F is, k , less than l , $k < l$. Let the best (in the Frobenius/spectral norm sense) rank k approximation of X be X_k and $X - X_k = E_k$. We know from [Remarks 6.1](#) that

$$\|E_k\|_F = \sqrt{\sum_{i=k+1}^l \sigma_i^2} \geq \sigma_l.$$

Also, because E_k is the perturbation (error) associated with the best approximation, then

$$\|E\|_F \geq \|E_k\|_F$$

or

$$\|E\|_F \geq \sigma_l.$$

However, from [\(6.75\)](#) we have that

$$\bar{\sigma}_{l+1} = \|E \vdots e\|_F \geq \|E\|_F \geq \sigma_l,$$

which violates assumption (3). Thus, $\text{rank}\{F\} = l$. Hence, there is a *unique* $\hat{\theta}_{\text{TLS}}$, such as

$$[F : g] \begin{bmatrix} \hat{\theta}_{\text{TLS}} \\ -1 \end{bmatrix} = \mathbf{0}. \quad (6.76)$$

In other words, $[\hat{\theta}_{\text{TLS}}^T, -1]^T$ belongs to the null space of

$$\begin{bmatrix} F & g \end{bmatrix}, \quad (6.77)$$

which is a rank-deficient matrix; hence, its null space is of dimension one, which is easily checked out that it is spanned by \bar{v}_{l+1} , leading to

$$\begin{bmatrix} \hat{\theta}_{\text{TLS}} \\ -1 \end{bmatrix} = \frac{-1}{\bar{v}_{l+1}(l+1)} \bar{v}_{l+1},$$

where $\bar{v}_{l+1}(l+1)$ is the last component of \bar{v}_{l+1} . Moreover, it can be shown that (Problem 6.18),

$$\boxed{\hat{\theta}_{\text{TLS}} = (X^T X - \bar{\sigma}_{l+1}^2 I)^{-1} X^T y : \quad \text{Total-Least-Squares Estimate.}} \quad (6.78)$$

Note that Assumption (3) guarantees that $X^T X - \bar{\sigma}_{l+1}^2 I$ is positive definite (think of why this is so).

Geometric interpretation of the total-least-squares method

From (6.65) and the definition of F in terms of its rows, f_1^T, \dots, f_N^T , we get

$$f_n^T \hat{\theta}_{\text{TLS}} - g_n = 0, \quad n = 1, 2, \dots, N, \quad (6.79)$$

or

$$\hat{\theta}_{\text{TLS}}^T (x_n - e_n) - (y_n - e_n) = 0. \quad (6.80)$$

In words, both the regressors, x_n , as well as the outputs, y_n , are corrected in order for the points, $(y_n - e_n, x_n - e_n)$, $n = 1, 2, \dots, N$, to lie on a hyperplane in \mathbb{R}^{l+1} . Also, once such a hyperplane is computed and it is unique, it has an interesting interpretation. It is the hyperplane that *minimizes the total square distance* of all the training points, (y_n, x_n) from it. Moreover, the corrected points $(y_n - e_n, x_n - e_n) = (g_n, f_n)$, $n = 1, 2, \dots, N$ are the *orthogonal projections* of the respective (y_n, x_n) training points onto this hyperplane. This is shown in Figure 6.12.

To prove the previous two claims, it suffices to show (Problem 6.19) that the direction of the hyperplane that minimizes the total distance from a set of points, (y_n, x_n) , $n = 1, 2, \dots, N$, is that defined by \bar{v}_{l+1} ; the latter is the eigenvector associated with the smallest singular value of $[X : y]$, assuming $\bar{\sigma}_l > \bar{\sigma}_{l+1}$.

To see that (g_n, f_n) is the orthogonal projection of (y_n, x_n) on this hyperplane, recall that our task minimizes the following Frobenius norm:

$$\|X - F : y - g\|_F^2 = \sum_{n=1}^N \left((y_n - g_n)^2 + \|x_n - f_n\|^2 \right).$$

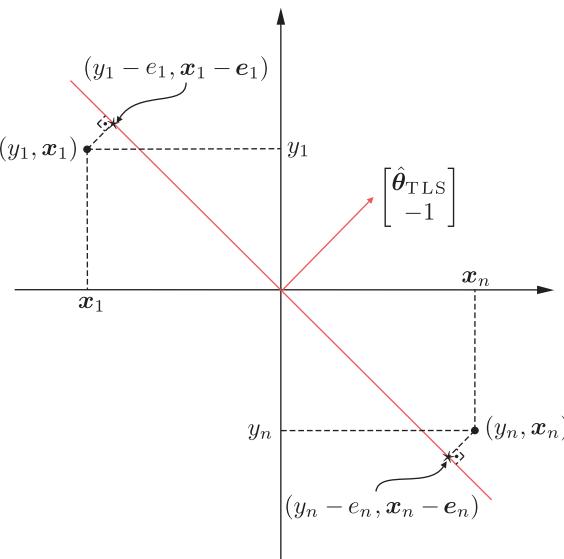


FIGURE 6.12

The total-least-squares method corrects both the values of the output variable as well as the input vector so that the points, after the correction, lie on a hyperplane. The corrected points are the orthogonal projections of (y_n, x_n) on the respective hyperplane; for centered data, this crosses the origin. For noncentered data, it crosses the centroid (\bar{y}, \bar{x}) .

However, each one of the terms in the above summation is the Euclidean distance between the points (y_n, x_n) and (g_n, f_n) , which is minimized if the latter is the orthogonal projection of the former on the hyperplane.

Remarks 6.5.

- The hyperplane defined by the TLS solution,

$$\left[\hat{\theta}_{TLS}^T, -1 \right]^T$$

minimizes the total distance of all the points (y_n, x_n) from it. We know from geometry that, the squared distance of each point from this hyperplane, is given by

$$\frac{|\hat{\theta}_{TLS}^T x_n - y_n|^2}{||\hat{\theta}_{TLS}^T||^2 + 1}.$$

Thus, $\hat{\theta}_{TLS}$ minimizes the following ratio:

$$\hat{\theta}_{TLS} = \arg \min_{\theta} \frac{||X\theta - y||^2}{||\theta||^2 + 1}.$$

This is basically a normalized (weighted) version of the least-squares cost. Looking at it more carefully, TLS promotes vectors of larger norm. This could be seen as a “deregularizing” tendency

of the TLS. From a numerical point of view, this can also be verified by (6.78). The matrix to be inverted for the TLS solution is more ill-conditioned than its LS counterpart. Robustness of the TLS can be improved via the use of regularization. Furthermore, extensions of TLS that employ other cost functions, in order to address the presence of outliers, have also been proposed.

- The TLS method has also been extended to deal with the more general case where y and θ become matrices. For further reading, the interested reader can look at [37, 64] and the references therein. A distributed algorithm for solving the total-least-squares task in Ad Hoc sensor networks has been proposed in Ref. [4]. A recursive scheme for the efficient solution of the TLS task has appeared in Ref. [12].
- TLS has widely been used in a number of applications, such as computer vision [47]; system identification [59]; speech and image processing [25], [51]; and spectral analysis [63].

Example 6.3. To demonstrate the potential of the total-least-squares to improve upon the performance of the least-squares estimator, in this example, we use noise not only in the input but also in the output samples. To this end, we generate randomly an input matrix, $X \in \mathbb{R}^{150 \times 90}$, filling it with elements according to the normalized Gaussian, $\mathcal{N}(0, 1)$. In the sequel, we generate the vector $\theta_o \in \mathbb{R}^{90}$ by randomly drawing samples also from the normalized Gaussian. The output vector is formed as

$$y = X\theta_o.$$

Then, we generate a noise vector $\eta \in \mathbb{R}^{150}$, filling it with elements randomly drawn from $\mathcal{N}(0, 0.01)$, and form

$$\tilde{y} = y + \eta.$$

A noisy version of the input matrix is obtained as

$$\tilde{X} = X + E,$$

where E is filled with elements randomly by drawing samples from $\mathcal{N}(0, 0.2)$.

Using the generated \tilde{y} , X , \tilde{X} and pretending that we do not know θ_o , the following three estimates are obtained for its value.

- Using the LS estimator (6.5) together with X and \tilde{y} , the average (over 10 different realizations) Euclidean distance of the obtained estimate, $\hat{\theta}$, from the true one is equal to $\|\hat{\theta} - \theta_o\| = 0.0125$.
- Using the LS estimator (6.5) together with \tilde{X} and \tilde{y} , the average (over 10 different realizations) Euclidean distance of the obtained estimate $\hat{\theta}$ from the true one is equal to $\|\hat{\theta} - \theta_o\| = 0.4272$.
- Using the TLS estimator (6.78) together with \tilde{X} and \tilde{y} , the average (over 10 different realizations) Euclidean distance of the obtained estimate $\hat{\theta}$ from the true one is equal to $\|\hat{\theta} - \theta_o\| = 0.2652$.

Observe that using noisy input data, the LS estimator resulted in higher error compared to the TLS one. Note, however, that the successful application of the TLS presupposes that the assumptions that led to the TLS estimator are valid.

PROBLEMS

6.1 Show that if $A \in \mathbb{C}^{m \times m}$ is positive semidefinite, its trace is nonnegative.

6.2 Show that under (a) the independence assumption of successive observation vectors and (b) the presence of white noise independent of the input, the LS estimator is asymptotically distributed according to the normal distribution, i.e.,

$$\sqrt{N}(\boldsymbol{\theta} - \boldsymbol{\theta}_0) \xrightarrow{} \mathcal{N}(\mathbf{0}, \sigma^2 \Sigma_x^{-1}),$$

where σ^2 is the noise variance and Σ_x the covariance matrix of the input observation vectors, assuming that it is invertible.

- 6.3** Let $X \in \mathbb{C}^{m \times l}$. Then show that the two matrices,

$$XX^H \text{ and } X^H X,$$

have the same eigenvalues.

- 6.4** Show that if $X \in \mathbb{C}^{m \times l}$, then the eigenvalues of XX^H ($X^H X$) are real and nonnegative. Moreover, show that if $\lambda_i \neq \lambda_j$, $\mathbf{v}_i \perp \mathbf{v}_j$.
- 6.5** Let $X \in \mathbb{C}^{m \times l}$. Then show that if \mathbf{v}_i is the normalized eigenvector of $X^H X$, corresponding to λ_i , then the corresponding normalized eigenvector \mathbf{u}_i of XX^H is given by

$$\mathbf{u}_i = \frac{1}{\sqrt{\lambda_i}} X \mathbf{v}_i.$$

- 6.6** Show Eq. (6.17).

- 6.7** Show that the eigenvectors, $\mathbf{v}_1, \dots, \mathbf{v}_r$, corresponding to the r singular values of a rank- r matrix, X , solve the following iterative optimization task: compute \mathbf{v}_k , $k = 2, 3, \dots, r$, such as,

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|X\mathbf{v}\|^2, \\ & \text{subject to} && \|\mathbf{v}\|^2 = 1, \\ & && \mathbf{v} \perp \{\mathbf{v}_1, \dots, \mathbf{v}_{k-1}\}, k \neq 1, \end{aligned}$$

where $\|\cdot\|$ denotes the Euclidean norm.

- 6.8** Show that projecting the rows of X onto the k -rank subspace, $V_k = \text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$, results in the largest variance, compared to any other k -dimensional subspace, Z_k .
- 6.9** Show that the squared Frobenius norm is equal to the sum of the squared singular values.
- 6.10** Show that the best k rank approximation of a matrix X or rank $r > k$, in the Frobenius norm sense, is given by

$$\hat{X} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^T,$$

where σ_i are the singular values and \mathbf{v}_i , \mathbf{u}_i , $i = 1, 2, \dots, r$, are the right and left singular vectors of X , respectively. Then show that the approximation error is given by

$$\sqrt{\sum_{i=k+1}^r \sigma_i^2}.$$

- 6.11** Show that \hat{X} , as given in Problem 6.10, also minimizes the spectral norm and that

$$\|X - \hat{X}\|_2 = \sigma_{k+1}.$$

- 6.12** Show that the Frobenius and spectral norms are unaffected by multiplication with orthogonal matrices, i.e.,

$$\|X\|_F = \|QXU\|_F$$

and

$$\|X\|_2 = \|QXU\|_2,$$

if $QQ^T = UU^T = I$.

- 6.13** Show that the null and range spaces of an $m \times l$ matrix, X , of rank r are given by

$$\mathcal{N}(X) = \text{span}\{\mathbf{v}_{r+1}, \dots, \mathbf{v}_l\},$$

$$\mathcal{R}(X) = \text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\},$$

where

$$X = [\mathbf{u}_1, \dots, \mathbf{u}_m] \begin{bmatrix} D & O \\ O & O \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \vdots \\ \mathbf{v}_l^T \end{bmatrix}.$$

- 6.14** Show that for the ridge regression

$$\hat{\mathbf{y}} = \sum_{i=1}^l \frac{\sigma_i^2}{\lambda + \sigma_i^2} (\mathbf{u}_i^T \mathbf{y}) \mathbf{u}_i.$$

- 6.15** Show that the normalized steepest descent direction of $J(\boldsymbol{\theta})$ at a point $\boldsymbol{\theta}_0$ for the quadratic norm $\|\mathbf{v}\|_P$ is given by

$$\mathbf{v} = -\frac{1}{\|P^{-1}\nabla J(\boldsymbol{\theta}_0)\|_P} P^{-1} \nabla J(\boldsymbol{\theta}_0).$$

- 6.16** Justify why the convergence of Newton's iterative minimization method is relatively insensitive on the Hessian matrix.

Hint. Let P be a positive definite matrix. Define a change of variables,

$$\tilde{\boldsymbol{\theta}} = P^{\frac{1}{2}} \boldsymbol{\theta},$$

and carry gradient descent minimization based on the new variable.

- 6.17** Show that the steepest descent direction, \mathbf{v} , of $J(\boldsymbol{\theta})$ at a point, $\boldsymbol{\theta}_0$, constrained to

$$\|\mathbf{v}\|_1 = 1,$$

is given by \mathbf{e}_k , where \mathbf{e}_k is the standard basis vector in the direction, k , such that

$$|(\nabla J(\boldsymbol{\theta}_0))_k| > |(\nabla J(\boldsymbol{\theta}_0))_j|, \quad k \neq j.$$

- 6.18** Show that the TLS solution is given by

$$\hat{\boldsymbol{\theta}} = \left(X^T X - \bar{\sigma}_{l+1}^2 I \right)^{-1} X^T \mathbf{y},$$

where $\bar{\sigma}_{l+1}$ is the smallest singular value of $[X : \mathbf{y}]$.

- 6.19** Given a set of centered data points, $(y_n, \mathbf{x}_n) \in \mathbb{R}^{l+1}$, derive a hyperplane

$$\mathbf{a}^T \mathbf{x} + y = 0,$$

which crosses the origin, such as the total square distance of all the points from it to be minimum.

MATLAB Exercises

- 6.20** Consider the regression model,

$$y_n = \boldsymbol{\theta}_o^T \mathbf{x}_n + \eta_n,$$

where $\boldsymbol{\theta}_o \in \mathbb{R}^{200}$ ($l = 200$) and the coefficients of the unknown vector are obtained randomly via the Gaussian distribution $\mathcal{N}(0, 1)$. The noise samples are also i.i.d., according to a Gaussian of zero mean and variance $\sigma_\eta^2 = 0.01$. The input sequence is a white noise one, i.i.d. generated via the Gaussian, $\mathcal{N}(0, 1)$.

Using as training data the samples, $(y_n, \mathbf{x}_n) \in \mathbb{R} \times \mathbb{R}^{200}$, $n = 1, 2, \dots$, run the APA (Algorithm 5.2), the NLMS (Algorithm 5.3), and the RLS (Algorithm 6.1) algorithms to estimate the unknown $\boldsymbol{\theta}_o$.

For the APA algorithm, choose $\mu = 0.2$, $\delta = 0.001$, and $q = 30$. Furthermore, in the NLMS set $\mu = 1.2$ and $\delta = 0.001$. Finally, for the RLS set the forgetting factor β equal to 1. Run 100 independent experiments and plot the average error per iteration in dBs, i.e., $10 \log_{10}(e_n^2)$, where $e_n^2 = (y_n - \mathbf{x}_n^T \boldsymbol{\theta}_{n-1})^2$. Compare the performance of the algorithms.

Keep playing with different parameters and study their effect on the convergence speed and the error floor in which the algorithms converge.

- 6.21** Consider the linear system

$$y_n = \mathbf{x}_n^T \boldsymbol{\theta}_{o,n-1} + \eta_n, \quad (6.81)$$

where $l = 5$ and the unknown vector is time varying. Generate the unknown vector with respect to the following model

$$\boldsymbol{\theta}_{o,n} = \alpha \boldsymbol{\theta}_{o,n-1} + \boldsymbol{\omega}_n,$$

where $\alpha = 0.97$ and the coefficients of $\boldsymbol{\omega}_n$ are i.i.d. drawn from the Gaussian distribution, with zero mean and variance equal to 0.1. Generate the initial value $\boldsymbol{\theta}_{o,0}$ with respect to the $\mathcal{N}(0, 1)$.

The noise samples are i.i.d., having zero mean and variance equal to 0.001. Furthermore, generate the input samples so that they follow the Gaussian distribution $\mathcal{N}(0, 1)$. Compare the performance of the NLMS and RLS algorithms. For the NLMS, set $\mu = 0.5$ and $\delta = 0.001$. For the RLS, set the forgetting factor β equal to 0.995. Run 200 independent experiments and plot the average error per iteration in dBs, i.e., $10 \log_{10}(e_n^2)$, with $e_n^2 = (y_n - \mathbf{x}_n^T \boldsymbol{\theta}_{n-1})^2$. Compare the performance of the algorithms.

Keep the same parameters, but set the variance associated with $\boldsymbol{\omega}_n$ equal to 0.01, 0.001. Play with different values of the parameters and the variance of the noise $\boldsymbol{\omega}$.

- 6.22** Generate an 150×90 matrix X , the entries of which follow the Gaussian distribution $\mathcal{N}(0, 1)$. Generate the vector $\boldsymbol{\theta}_o \in \mathbb{R}^{90}$. The coefficients of this vector are i.i.d. obtained, also, via the Gaussian $\mathcal{N}(0, 1)$. Compute the vector $\mathbf{y} = X\boldsymbol{\theta}_o$. Add a 90×1 noise vector, $\boldsymbol{\eta}$, to \mathbf{y} in order to generate $\tilde{\mathbf{y}} = \mathbf{y} + \boldsymbol{\eta}$. The elements of $\boldsymbol{\eta}$ are generated via the Gaussian $\mathcal{N}(0, 0.01)$. In the sequel,

add a 150×90 noise matrix, E , so as to produce $\tilde{X} = X + E$; the elements of E are generated according to the Gaussian $\mathcal{N}(0, 0.2)$. Compute the LS estimate, via (6.5), by employing (a) the true input matrix, X , and the noisy output \tilde{y} ; and (b) the noisy input matrix, \tilde{X} , and the noisy output \tilde{y} .

In the sequel, compute the TLS estimate via (6.78) using the noisy input matrix, \tilde{X} , and the noisy output \tilde{y} .

Repeat the experiments a number of times and compute the average Euclidean distances between the obtained estimates for the previous three cases, and the true parameter vector, θ_o .

Play with different noise levels and comment on the results.

REFERENCES

- [1] R.J. Adcock, Note on the method of least-squares, *Analyst* 4(6) (1877) 183-184.
- [2] J.A. Apolinario Jr. (Ed.), QRD-RLS Adaptive Filtering, Springer, New York, 2009.
- [3] K. Berberidis, S. Theodoridis, Efficient symmetric algorithms for the modified covariance method for autoregressive spectral analysis, *IEEE Trans. Signal Process.* 41 (1993) 43.
- [4] A. Bertrand, M. Moonen, Consensus-based distributed total least-squares estimation in Ad Hoc wireless sensor networks, *IEEE Trans. Signal Process.* 59 (5) (2011) 2320-2330.
- [5] J.L. Botto, G.V. Moustakides, Stabilizing the fast Kalman algorithms, *IEEE Trans. Acoust. Speech Signal Process.* 37 (1989) 1344-1348.
- [6] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.
- [7] G. Carayannis, D. Manolakis, N. Kalouptsidis, A fast sequential algorithm for least-squares filtering and prediction, *IEEE Trans. Acoust. Speech Signal Process.* 31 (1983) 1394-1402.
- [8] F.S. Cattivelli, C.G. Lopes, A.H. Sayed, Diffusion recursive least-squares for distributed estimation over adaptive networks, *IEEE Trans. Signal Process.* 56 (5) (2008) 1865-1877.
- [9] J.M. Cioffi, T. Kailath, Fast recursive-least-squares transversal filters for adaptive filtering, *IEEE Trans. Acoust. Speech Signal Process.* 32 (1984) 304-337.
- [10] J.M. Cioffi, The fast adaptive ROTOR's RLS algorithm, *IEEE Trans. Acoust. Speech Signal Process.* 38 (1990) 631-653.
- [11] M.H. Costa, J.C.M. Bermudez, An improved model for the normalized LMS algorithm with Gaussian inputs and large number of coefficients, in: Proceedings, IEEE Conference in Acoustics Speech and Signal Processing, 2002, pp. 1385-1388.
- [12] C.E. Davila, An efficient recursive total least-squares algorithm for FIR adaptive filtering, *IEEE Trans. Signal Process.* 42 (1994) 268-280.
- [13] W.E. Deming, Statistical Adjustment of Data, J. Wiley and Sons, 1943.
- [14] P.S.R. Diniz, M.L.R. De Campos, A. Antoniou, Analysis of LMS-Newton adaptive filtering algorithms with variable convergence factor, *IEEE Trans. Signal Process.* 43 (1995) 617-627.
- [15] P.S.R. Diniz, Adaptive Filtering: Algorithms and Practical Implementation, third ed., Springer, 2008.
- [16] Y.C. Eldar, Minimax MSE estimation of deterministic parameters with noise covariance uncertainties, *IEEE Trans. Signal Process.* 54 (2006) 138-145.
- [17] B. Farhang-Boroujeny, Adaptive Filters: Theory and Applications, J. Wiley, NY, 1999.
- [18] J. Friedman, T. Hastie, H. Hofling, R. Tibshirani, Pathwise coordinate optimization, *Ann. Appl. Stat.* 1 (2007) 302-332.
- [19] J.W. Gillard, A historical review of linear regression with errors in both variables, Technical Report, University of Cardiff, School of Mathematics, 2006.

- [20] G. Glentis, K. Berberidis, S. Theodoridis, Efficient least-squares adaptive algorithms for FIR transversal filtering, *IEEE Signal Process. Mag.* 16 (1999) 13-42.
- [21] G.O. Glentis, A. Jakobsson, Superfast Approximative Implementation of the IAA Spectral Estimate, *IEEE Trans. Signal Process.* 60 (1) (2012) 472-478.
- [22] G.H. Golub, C.F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1983.
- [23] B. Hassibi, A.H. Sayed, T. Kailath, H^∞ optimality of the LMS algorithm, *IEEE Trans. Signal Process.* 44 (1996) 267-280.
- [24] S. Haykin, *Adaptive Filter Theory*, fourth ed., Prentice Hall, NJ, 2002.
- [25] K. Hermus, W. Verhelst, P. Lemmerling, P. Wambacq, S. Van Huffel, Perceptual audio modeling with exponentially damped sinusoids, *Signal Process.* 85 (1) (2005) 163-176.
- [26] R.A. Horn, C.R. Johnson, *Matrix Analysis*, second ed., Cambridge University Press, 2013.
- [27] N. Kalouptsidis, G. Carayannis, D. Manolakis, E. Koukoutsis, Efficient recursive in order least-squares FIR filtering and prediction, *IEEE Trans. Acoust. Speech Signal Process.* 33 (1985) 1175-1187.
- [28] N. Kalouptsidis, S. Theodoridis, Parallel implementation of efficient LS algorithms for filtering and prediction, *IEEE Trans. Acoust. Speech Signal Process.* 35 (1987) 1565-1569.
- [29] N. Kalouptsidis, S. Theodoridis, *Adaptive System Identification and Signal Processing Algorithms*, Prentice Hall, 1993.
- [30] A.P. Liavas, P.A. Regalia, On the numerical stability and accuracy of the conventional recursive least-squares algorithm, *IEEE Trans. Signal Process.* 47 (1999) 88-96.
- [31] F. Ling, D. Manolakis, J.G. Proakis, Numerically robust least-squares lattice-ladder algorithms with direct updating of the reflection coefficients, *IEEE Trans. Acoust. Speech Signal Process.* 34 (1986) 837-845.
- [32] D.L. Lee, M. Morf, B. Friedlander, Recursive least-squares ladder estimation algorithms, *IEEE Trans. Acoust. Speech Signal Process.* 29 (1981) 627-641.
- [33] L. Ljung, M. Morf, D. Falconer, Fast calculation of gain matrices for recursive estimation schemes, *Int. J. Control* 27 (1984) 304-337.
- [34] S. Ljung, L. Ljung, Error propagation properties of recursive least-squares adaptation algorithms, *Automatica* 21 (1985) 157-167.
- [35] I. Loshchilov, M. Schoenauer, M. Sebag, Adaptive coordinate descent, in: *Proceedings Genetic and Evolutionary Computation Conference (GECCO)*, ACM Press, 2011, pp. 885-892.
- [36] Z. Luo, P. Tseng, On the convergence of the coordinate descent method for convex differentiable minimization, *J. Optim. Theory Appl.* 72 (1992) 7-35.
- [37] I. Markovsky, S. Van Huffel, Overview of total least-squares methods, *Signal Process.* 87 (10) (2007) 2283-2302.
- [38] D.F. Marshall, W.K. Jenkins, A fast quasi-Newton adaptive filtering algorithm, *IEEE Trans. Signal Process.* 40 (1993) 1652-1662.
- [39] G. Mateos, I. Schizas, G.B. Giannakis, Distributed recursive least-squares for consensus-based in-network adaptive estimation, *IEEE Trans. Signal Process.* 57 (11) (2009) 4583-4588.
- [40] G. Mateos, G.B. Giannakis, Distributed recursive least-squares: stability and performance analysis, *IEEE Trans. Signal Process.* 60 (7) (2012) 3740-3754.
- [41] J.G. McWhirter, Recursive least-squares minimization using a systolic array, *Proc. SPIE Real Time Signal Process.* VI 431 (1983) 105-112.
- [42] M. Morf, Fast algorithms for multivariable systems, Ph.D. Thesis, Stanford University, Stanford, CA, 1974.
- [43] M. Morf, T. Kailath, Square-root algorithms for least-squares estimation, *IEEE Trans. Automat. Control* 20 (1975) 487-497.
- [44] M. Morf, B. Dickinson, T. Kailath, A. Vieira, Efficient solution of covariance equations for linear prediction, *IEEE Trans. Acoust. Speech Signal Process.* 25 (1977) 429-433.

- [45] G.V. Moustakides, S. Theodoridis, Fast Newton transversal filters: A new class of adaptive estimation algorithms, *IEEE Trans. Signal Process.* 39 (1991) 2184-2193.
- [46] G.V. Moustakides, Study of the transient phase of the forgetting factor RLS, *IEEE Trans. Signal Process.* 45 (1997) 2468-2476.
- [47] M. Mühllich, R. Mester, The role of total least-squares in motion analysis, in: H. Burkhardt (Ed.), *Proceedings of the 5th European Conference on Computer Vision*, Springer-Verlag, 1998, pp. 305-321.
- [48] V.H. Nascimento, M.T.M. Silva, Adaptive filters, in: R. Chellappa, S. Theodoridis (Eds.), *Signal Process, E-Ref.* 1 (2014) 619-747.
- [49] T. Petillon, A. Gilloire, S. Theodoridis, Fast Newton transversal filters: An efficient way for echo cancellation in mobile radio communications, *IEEE Trans. Signal Process.* 42 (1994) 509-517.
- [50] T. Piotrowski, I. Yamada, MV-PURE estimator: Minimum-variance pseudo-unbiased reduced-rank estimator for linearly constrained ill-conditioned inverse problems, *IEEE Trans. Signal Process.* 56 (2008) 3408-3423.
- [51] A. Pruessner, D. O'Leary, Blind deconvolution using a regularized structured total least norm algorithm, *SIAM Journal on Matrix Analysis and Applications* 24(4) (2003) 1018-1037.
- [52] P.A. Regalia, Numerical stability properties of a QR-based fast least-squares algorithm, *IEEE Trans. Signal Process.* 41 (1993) 2096-2109.
- [53] A.A. Rondogiannis, S. Theodoridis, On inverse factorization adaptive least-squares algorithms, *Signal Processing* 52 (1997) 35-47.
- [54] A.A. Rondogiannis, S. Theodoridis, New fast QR decomposition least-squares adaptive algorithms, *IEEE Trans. Signal Process.* 46 (1998) 2113-2121.
- [55] A. Rontogiannis, S. Theodoridis, Householder-Based RLS Algorithms, in: J.A. Apolonario Jr. (Ed.), *QRD-RLS Adaptive Filtering*, Springer, 2009.
- [56] A.H. Sayed, T. Kailath, A state space approach to adaptive RLS filtering, *IEEE Signal Processing Magazine* 11 (1994) 18-60.
- [57] A.H. Sayed, *Fundamentals of Adaptive Filtering*, J. Wiley Interscience, 2003.
- [58] D.T.M. Slock, R. Kailath, Numerically stable fast transversal filters for recursive least-squares adaptive filtering, *IEEE Trans. Signal Process.* 39 (1991) 92-114.
- [59] T. Söderström, Errors-in-variables methods in system identification, *Automatica* 43(6) (2007) 939-958.
- [60] S. Theodoridis, Pipeline architecture for block adaptive LS FIR filtering and prediction, *IEEE Trans. Acoust. Speech Signal Process.* 38 (1990) 81-90.
- [61] S. Theodoridis, A. Liavas, Highly concurrent algorithm for the solution of ρ -Toeplitz system of equations, *Signal Process.* 24 (1991) 165-176.
- [62] P. Tseng, Convergence of a block coordinate descent method for nondifferentiable minimization, *J. Optim. Theory Appl.* 109 (2001) 475-494.
- [63] D. Tufts, R. Kumaresan, Estimation of frequencies of multiple sinusoids: Making linear prediction perform like maximum likelihood, *Proc. IEEE* 70 (9) (1982) 975-989.
- [64] S. Van Huffel, J. Vandewalle, *The Total-Least-Squares Problem: Computational Aspects and Analysis*, SIAM, Philadelphia, 1991.
- [65] M.H. Verhaegen, Round-off error propagation in four generally-applicable, recursive, least-squares estimation schemes, *Automatica* 25 (1989) 437-444.
- [66] G.P. White, Y.V. Zakharov, J. Liu, Low complexity RLS algorithms using dichotomous coordinate descent iterations, *IEEE Transactions on Signal Processing* 56 (2008) 3150-3161.
- [67] T.T. Wu, K. Lange, Coordinate descent algorithms for lasso penalized regression, *Ann. Appl. Stat.* 2 (2008) 224-244.
- [68] B. Yang, A note on the error propagation analysis of recursive least-squares algorithms, *IEEE Trans. Signal Process.* 42 (1994) 3523-3525.

CLASSIFICATION: A TOUR OF THE CLASSICS

7

CHAPTER OUTLINE

7.1	Introduction	277
7.2	Bayesian Classification	278
	<i>The Bayesian Classifier Minimizes the Misclassification Error</i>	279
	7.2.1 Average Risk	280
7.3	Decision (Hyper)Surfaces	283
	7.3.1 The Gaussian Distribution Case	284
	<i>Minimum Distance Classifiers</i>	288
7.4	The Naive Bayes Classifier	290
7.5	The Nearest Neighbor Rule	291
7.6	Logistic Regression	293
7.7	Fisher's Linear Discriminant	297
7.8	Classification Trees	303
7.9	Combining Classifiers	307
	<i>Experimental Comparisons</i>	307
	<i>Schemes for Combining Classifiers</i>	308
7.10	The Boosting Approach	310
	<i>The AdaBoost Algorithm</i>	310
	<i>The Log-Loss Function</i>	314
7.11	Boosting Trees	316
7.12	A Case Study: Protein Folding Prediction	317
	<i>Protein Folding Prediction as a Classification Task</i>	319
	<i>Classification of Folding Prediction via Decision Trees</i>	321
Problems		321
	<i>MATLAB Exercises</i>	323
References		326

7.1 INTRODUCTION

The classification task was introduced in [Chapter 3](#). There, it was pointed out that, in principle, one could employ the same loss functions as those used for regression in order to optimize the design of a classifier; however, for most cases in practice, this is not the most reasonable way to attack such problems. This

is because in classification the output variable, y , is of a *discrete nature*; hence, different measures than those used for the regression task are more appropriate for quantifying performance quality.

The goal of this chapter is to present a number of widely used loss functions and methods. Most of the techniques covered are conceptually simple and constitute the basic pillars on which classification is built. Besides their pedagogical importance, these techniques are still in use in a number of practical applications and often form the basis for the development of more advanced methods, to be covered later in the book.

The classical Bayesian classification rule; the notion of minimum distance classifiers; the logistic regression loss function; classification trees; and the method of combining classifiers, including the powerful technique of boosting, will be discussed. The perceptron rule, although it boasts to be among the most basic classification rules, will be treated in [Chapter 18](#) and it will be used as the starting point for introducing neural networks and deep learning techniques. Support vector machines are treated in the framework of reproducing Kernel Hilbert spaces, in [Chapter 11](#).

In a nutshell, this chapter can be considered a beginner's tour of the task of designing classifiers.

7.2 BAYESIAN CLASSIFICATION

In [Chapter 3](#), a linear classifier was designed via the least-squares (LS) cost function. However, the LS criterion cannot serve well the needs of the classification task. In [Chapters 3 and 6](#), we have proved that the LS estimator is an efficient one only if the conditional distribution of the output variable, y , given the feature values, \mathbf{x} , follows a Gaussian distribution of a special type. However, in classification, the dependent variable is discrete, hence it is not Gaussian; thus, the use of the LS criterion cannot be justified, in general. We will return to this issue in [Section 7.10 \(Remarks 7.7\)](#), when the LS criterion is discussed against other loss functions used in classification.

In this section, the classification task will be approached via a different path, inspired by the *Bayesian decision theory*. In spite of its conceptual simplicity, which ties very well with common sense, Bayesian classification possesses a strong optimality flavor with respect to the probability of error; that is, the probability of wrong decisions/class predictions that a classifier commits.

Bayesian classification rule: Given a set of M classes, ω_i , $i = 1, 2, \dots, M$, and the respective *posterior probabilities* $P(\omega_i|\mathbf{x})$, classify an unknown feature vector, \mathbf{x} , according to the rule:

$$\boxed{\text{Assign } \mathbf{x} \text{ to } \omega_i = \arg \max_{\omega_j} P(\omega_j|\mathbf{x}), \quad j = 1, 2, \dots, M.} \quad (7.1)$$

In words, the unknown pattern, represented by \mathbf{x} , is assigned to the class for which the posterior probability becomes maximum.

Note that prior to receiving any observation, our uncertainty concerning the classes is expressed via the prior probabilities, denoted by $P(\omega_i)$, $i = 1, 2, \dots, M$. Once the observation \mathbf{x} has been obtained, this extra information removes part of our original uncertainty, and the related statistical information is now provided by the posterior probabilities, which are then used for the classification.

Employing in (7.1) Bayes theorem,

$$P(\omega_j|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_j)P(\omega_j)}{p(\mathbf{x})}, \quad j = 1, 2, \dots, M, \quad (7.2)$$

where $p(\mathbf{x}|\omega_j)$ are the respective conditional probability distribution densities (pdf), the Bayesian classification rule becomes

$$\text{Assign } \mathbf{x} \text{ to } \omega_i = \arg \max_{\omega_j} p(\mathbf{x}|\omega_j)P(\omega_j), \quad j = 1, 2, \dots, M. \quad (7.3)$$

Note that the data pdf, $p(\mathbf{x})$, in the denominator of (7.2) does not enter in the maximization task, because it is a positive quantity independent of the classes ω_j ; hence, it does not affect the maximization. In other words, the classifier depends on the a priori class probabilities and the respective conditional pdfs. Also, note that

$$p(\mathbf{x}|\omega_j)P(\omega_j) = p(\omega_j, \mathbf{x}) := p(y, \mathbf{x}).$$

The last equation verifies what said in [Chapter 3](#): the Bayesian classifier is a generative modeling technique.

We now turn our attention to how one can obtain estimates of the involved quantities. Recall that in practice, all one has at one's disposal is a set of training data, from which estimates of the prior probabilities as well as the conditional pdfs must be obtained. Let us assume that we are given a set of training points, $(y_n, \mathbf{x}_n) \in D \times \mathbb{R}^l$, $n = 1, 2, \dots, N$, where D is the set of class labels, and consider the general task comprising M classes. Assume that each class, ω_i , $i = 1, 2, \dots, M$, is represented by N_i points in the training set, with $\sum_{i=1}^M N_i = N$. Then, the a priori probabilities can be approximated by

$$P(\omega_i) \approx \frac{N_i}{N}, \quad i = 1, 2, \dots, M. \quad (7.4)$$

For the conditional pdfs, $p(\mathbf{x}|\omega_i)$, $i = 1, 2, \dots, M$, any method for estimating pdfs can be mobilized. For example, one can assume a known parametric form for each one of the conditionals and adopt the maximum likelihood (ML) method, discussed in [Section 3.10](#), or the maximum a posteriori (MAP) estimator, discussed in [Section 3.11.1](#), in order to obtain estimates of the parameters using the training data from each one of the classes. Another alternative is to resort to nonparametric histogram-like techniques, such as Parzen windows and the k -nearest neighbor density estimation techniques, as discussed in [Section 3.15](#). Other methods for pdf estimation can also be employed, such as mixture modeling, to be discussed later in [Chapter 12](#). The interested reader may also consult [52, 53].

The Bayesian classifier minimizes the misclassification error

In [Section 3.4](#), it was pointed out that the goal of designing a classifier is to partition the space in which the feature vectors lie into regions, and associate each one of the regions to one and only one class. For a two-class task (the generalization to more classes is straightforward), let $\mathcal{R}_1, \mathcal{R}_2$ be the two regions in \mathbb{R}^l , where we decide in favor of class ω_1 and ω_2 , respectively. The probability of classification error is given by

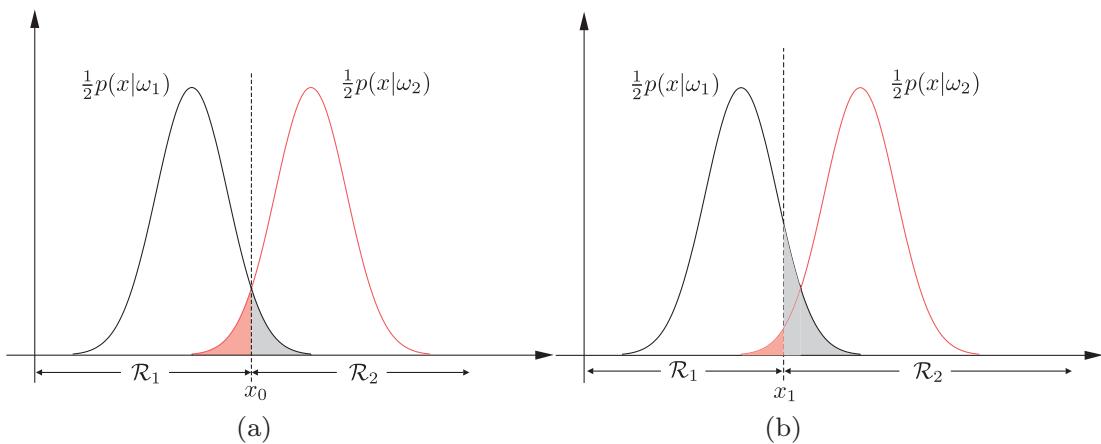
$$P_e = P(\mathbf{x} \in \mathcal{R}_1, \mathbf{x} \in \omega_2) + P(\mathbf{x} \in \mathcal{R}_2, \mathbf{x} \in \omega_1). \quad (7.5)$$

That is, it is equal to the probability of the feature vector to belong to class ω_1 (ω_2) *and* to lie in the “wrong” region \mathcal{R}_2 (\mathcal{R}_1) in the feature space.

Equation (7.5) can be written as

$$P_e = P(\omega_2) \int_{\mathcal{R}_1} p(\mathbf{x}|\omega_2) d\mathbf{x} + P(\omega_1) \int_{\mathcal{R}_2} p(\mathbf{x}|\omega_1) d\mathbf{x} : \quad \text{Probability of Error.} \quad (7.6)$$

It turns out that the Bayesian classifier, as defined in (7.3), minimizes P_e with respect to \mathcal{R}_1 and \mathcal{R}_2 [17, 52]. This is also true for the general case of M classes ([Problem 7.1](#)).

**FIGURE 7.1**

(a) The classification error probability for dividing the feature space, according to the Bayesian optimal classifier, is equal to the area of the shaded region. (b) Moving the threshold value away from the value corresponding to the optimal Bayes rule increases the probability of error, as is indicated by the increase of the area of the corresponding shaded region.

Figure 7.1a demonstrates geometrically the optimality of the Bayesian classifier for the two-class one-dimensional case and assuming equiprobable classes ($P(\omega_1) = P(\omega_2) = 1/2$). The region \mathcal{R}_1 , to the left of the threshold value, x_0 , corresponds to $p(x|\omega_1) > p(x|\omega_2)$, and the opposite is true for region \mathcal{R}_2 . The probability of error is equal to the area of the shaded region, which is equal to the sum of the two integrals in (7.6). In **Figure 7.1b**, the threshold has been moved away from the optimal Bayesian value, and as a result the probability of error, given by the total area of the corresponding shaded region, increases.

7.2.1 AVERAGE RISK

Because in classification the dependent variable (label), y , is of a discrete nature, the classification error probability may seem like the most natural cost function to be optimized. However, this is not always true. In certain applications, not all errors are of the same importance. For example, in a medical diagnosis system, committing an error by predicting the class of a finding in an X-ray image as being “malignant” while its true class is “normal” is less significant than an error the other way around. In the former case, the wrong diagnosis will be revealed in the next set of medical tests. However, the opposite may have unwanted consequences. For such cases, one uses an alternative to the probability of error cost function that puts relative weights on the errors according to their importance. This cost function is known as the *average risk* and it results in a rule that resembles that of the Bayesian classifier, yet it is slightly modified due to the presence of the weights.

For the M -class problem, the *risk* or *loss* associated with class ω_k is defined as

$$r_k = \sum_{i=1}^M \lambda_{ki} \int_{\mathcal{R}_i} p(x|\omega_k) dx, \quad (7.7)$$

where, $\lambda_{kk} = 0$ and λ_{ki} is the weight that controls the significance of committing an error by assigning a pattern from class ω_k to class ω_i . The *average risk* is given by

$$r = \sum_{k=1}^M P(\omega_k) r_k = \sum_{i=1}^M \int_{\mathcal{R}_i} \left(\sum_{k=1}^M \lambda_{ki} P(\omega_k) p(\mathbf{x}|\omega_k) \right) d\mathbf{x}. \quad (7.8)$$

The average risk is minimized if we partition the input space by selecting each \mathcal{R}_i (where we decide in favor of class ω_i) so that each one of the M integrals in the summation becomes minimum; this is achieved if we adopt the rule

$$\text{Assign } \mathbf{x} \text{ to } \omega_i : \sum_{k=1}^M \lambda_{ki} P(\omega_k) p(\mathbf{x}|\omega_k) < \sum_{k=1}^M \lambda_{kj} P(\omega_k) p(\mathbf{x}|\omega_k), \quad \forall j \neq i,$$

or equivalently,

$$\boxed{\text{Assign } \mathbf{x} \text{ to } \omega_i : \sum_{k=1}^M \lambda_{ki} P(\omega_k|\mathbf{x}) < \sum_{k=1}^M \lambda_{kj} P(\omega_k|\mathbf{x}), \quad \forall j \neq i.} \quad (7.9)$$

For the two-class case, it is readily seen that the rule becomes

$$\text{Assign } \mathbf{x} \text{ to } \omega_1 (\omega_2) \text{ if} : \lambda_{12} P(\omega_1|\mathbf{x}) > (<) \lambda_{21} P(\omega_2|\mathbf{x}),$$

or equivalently

$$\text{Assign } \mathbf{x} \text{ to } \omega_1 (\omega_2) \text{ if} : \lambda_{12} P(\omega_1)p(\mathbf{x}|\omega_1) > (<) \lambda_{21} P(\omega_2)p(\mathbf{x}|\omega_2). \quad (7.10)$$

If one sets $P'(\omega_1) := \lambda_{12} P(\omega_1)$ ($P'(\omega_2) := \lambda_{21} P(\omega_2)$), one may view the effect of the weights as modifying the respective prior class probabilities by relatively increasing the value of the more important class.

It is common to consider the weights λ_{ij} as defining an $M \times M$ matrix

$$L := [\lambda_{ij}], \quad i, j = 1, 2, \dots, M, \quad (7.11)$$

which is known as the *loss matrix*. Note that if we set $\lambda_{ki} = 1$, $k = 1, 2, \dots, M$, $i = 1, 2, \dots, M$, $k \neq i$, then we obtain the Bayes rule (verify it).

Remarks 7.1.

- *The reject option:* Bayesian classification relies on the maximum value of the posterior probabilities, $P(\omega_i|\mathbf{x})$, $i = 1, 2, \dots, M$. However, often in practice, it may happen that for some value, \mathbf{x} , the maximum value is comparable to the values the posterior obtains for other classes. For example, in a two-class task, it may turn out that $P(\omega_1|\mathbf{x}) = 0.51$ and $P(\omega_2|\mathbf{x}) = 0.49$. If this happens, it may be more sensible not to make a decision for this particular pattern, \mathbf{x} . This is known as the *reject option*. If such a decision scenario is adopted, a user-defined threshold value, θ , is chosen and classification is carried out only if the maximum posterior is larger than this threshold value, so that, $P(\omega_i|\mathbf{x}) > \theta$. Otherwise, no decision is taken. Similar arguments can be adopted for the average risk classification.

Example 7.1. In a two-class, one-dimensional classification task, the data in the two classes are distributed according to the following two Gaussians:

$$p(x|\omega_1) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right),$$

and

$$p(x|\omega_2) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x-1)^2}{2}\right).$$

The problem is more sensitive with respect to errors committed on patterns from class ω_1 , which is expressed via the following loss matrix:

$$L = \begin{bmatrix} 0 & 1 \\ 0.5 & 0 \end{bmatrix}.$$

In other words, $\lambda_{12} = 1$ and $\lambda_{21} = 0.5$. The two classes are considered equiprobable. Derive the threshold value, x_r , which partitions the feature space, \mathbb{R} , into the two regions, \mathcal{R}_1 , \mathcal{R}_2 , in which we decide in favor of class ω_1 and ω_2 , respectively. What is the value of the threshold when the Bayesian classifier is used instead?

Solution: According to the average risk rule, the region for which we decide in favor of class ω_1 is given by

$$\mathcal{R}_1 : \lambda_{12} \frac{1}{2} p(x|\omega_1) > \lambda_{21} \frac{1}{2} p(x|\omega_2),$$

and the respective threshold value, x_r , is computed by the equation

$$\exp\left(-\frac{x_r^2}{2}\right) = 0.5 \exp\left(-\frac{(x_r-1)^2}{2}\right),$$

which, after taking the logarithm and solving the respective equation, trivially results in

$$x_r = \frac{1}{2}(1 - 2 \ln 0.5).$$

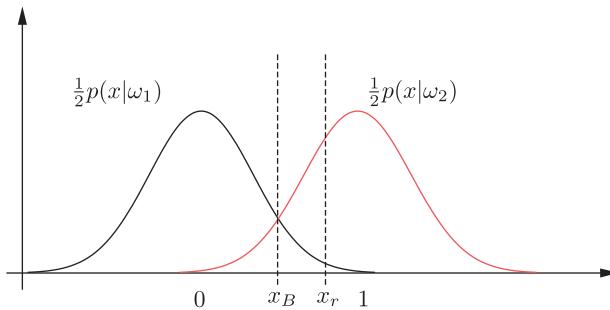
The threshold for the Bayesian classifier results if we set $\lambda_{21} = 1$, which gives

$$x_B = \frac{1}{2}.$$

The geometry is shown in [Figure 7.2](#). In other words, the use of the average risk moves the threshold to the right of the value corresponding to the Bayesian classifier; that is, it enlarges the region in which we decide in favor of the more significant class, ω_1 . Note that this would also be the case if the two classes were not equiprobable, as shown by $P(\omega_1) > P(\omega_2)$ (for our example, $P(\omega_1) = 2P(\omega_2)$).

7.3 DECISION (HYPER)SURFACES

The goal of any classifier is to partition the feature space into regions. The partition is achieved via points in (\mathbb{R}) , curves in (\mathbb{R}^2) , surfaces in (\mathbb{R}^3) , and hypersurfaces in (\mathbb{R}^l) . Any hypersurface, S , is expressed in terms of a function

**FIGURE 7.2**

The class distributions and the resulting threshold values for the two cases of [Example 7.1](#). Note that minimizing the average risk enlarges the region in which we decide in favor of the most sensitive class, ω_1 .

$$g : \mathbb{R}^l \mapsto \mathbb{R},$$

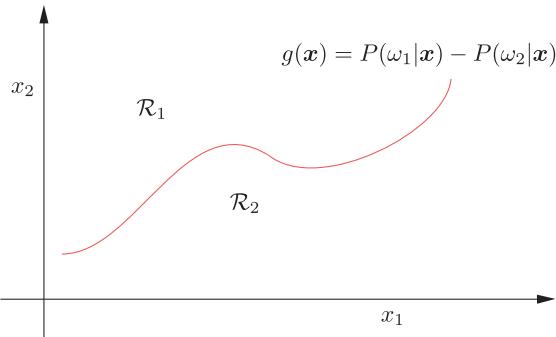
and it comprises all the points such that

$$S = \left\{ \mathbf{x} \in \mathbb{R}^l : g(\mathbf{x}) = 0 \right\}.$$

Recall that all points lying on one side of this hypersurface score $g(\mathbf{x}) > 0$ and all the points on the other side score $g(\mathbf{x}) < 0$. The resulting (hyper)surfaces are known as *decision (hyper)surfaces*, for obvious reasons. Take as an example the case of the two-class Bayesian classifier. The respective decision hypersurface is (implicitly) formed by

$$g(\mathbf{x}) := P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0. \quad (7.12)$$

Indeed, we decide in favor of class ω_1 (region \mathcal{R}_1) if \mathbf{x} falls on the positive side of the hypersurface defined in (7.12), and in favor of ω_2 for the points falling on the negative side (region \mathcal{R}_2). This is illustrated in [Figure 7.3](#). At this point, recall the reject option from [Remarks 7.1](#). Points where no decision is taken are those that lie close to the decision hypersurface.

**FIGURE 7.3**

The Bayesian classifier implicitly forms hypersurfaces defined by $g(\mathbf{x}) = P(\omega_1|\mathbf{x}) - P(\omega_2|\mathbf{x}) = 0$.

Once we move away from the Bayesian concept of designing classifiers (as we will soon see, and this will be done for a number of reasons), different families of functions for selecting $g(\mathbf{x})$ can be adopted and the specific form will be obtained via different optimization criteria, which are not necessarily related to the probability of error/average risk.

In the sequel, we focus on investigating the form that the decision hypersurfaces take for the special case of the Bayesian classifier and where the data in the classes are distributed according to the Gaussian pdf. This can provide further insight into the way a classifier partitions the feature space and it will also lead to some useful implementations of the Bayesian classifier, under certain scenarios. For simplicity, the focus will be on two-class classification tasks, but the results are trivially generalized to the more general M -class case.

7.3.1 THE GAUSSIAN DISTRIBUTION CASE

Assume that the data in each class are distributed according to the Gaussian pdf, so that,

$$p(\mathbf{x}|\omega_i) = \frac{1}{(2\pi)^{l/2} |\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^T \Sigma_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i)\right), \quad i = 1, 2, \dots, M.$$

Because the logarithmic function is a monotonically increasing one, it does not affect the maximum of a function. Thus, taking into account the exponential form of the Gaussian, the computations can be facilitated if the Bayesian rule is expressed in terms of the following functions:

$$g_i(\mathbf{x}) := \ln P(\omega_i|\mathbf{x}) = \ln p(\mathbf{x}|\omega_i) + \ln P(\omega_i), \quad i = 1, 2, \dots, M, \quad (7.13)$$

and search for the class for which the respective function scores the maximum value. Such functions are also known as *discriminant functions*.

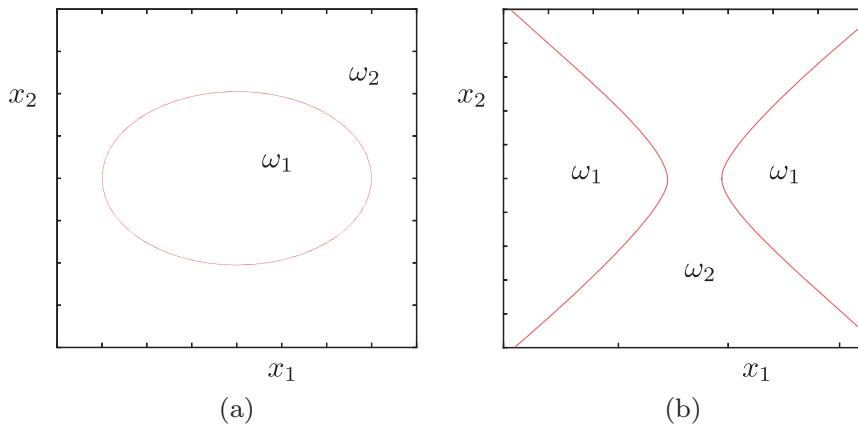
Let us now focus on the two-class classification task. The decision hypersurface, associated with the Bayesian classifier, is expressed as

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) = 0, \quad (7.14)$$

which, after plugging into (7.13) the specific forms of the Gaussian conditionals, and after a bit of trivial algebra, becomes

$$\begin{aligned} g(\mathbf{x}) &= \underbrace{\frac{1}{2} \left(\mathbf{x}^T \Sigma_2^{-1} \mathbf{x} - \mathbf{x}^T \Sigma_1^{-1} \mathbf{x} \right)}_{\text{quadratic terms}} \\ &\quad + \underbrace{\boldsymbol{\mu}_1^T \Sigma_1^{-1} \mathbf{x} - \boldsymbol{\mu}_2^T \Sigma_2^{-1} \mathbf{x}}_{\text{linear terms}} \\ &\quad - \underbrace{\frac{1}{2} \boldsymbol{\mu}_1^T \Sigma_1^{-1} \boldsymbol{\mu}_1 + \frac{1}{2} \boldsymbol{\mu}_2^T \Sigma_2^{-1} \boldsymbol{\mu}_2 + \ln \frac{P(\omega_1)}{P(\omega_2)} + \frac{1}{2} \ln \frac{|\Sigma_2|}{|\Sigma_1|}}_{\text{constant terms}} = 0. \end{aligned} \quad (7.15)$$

This is of a quadratic nature, hence the corresponding (hyper)surfaces are *(hyper)quadrics*, including (hyper)ellipsoids, (hyper)parabolas, hyperbolas. Figure 7.4 shows two examples, in the two-dimensional space, corresponding to $P(\omega_1) = P(\omega_2)$, and

**FIGURE 7.4**

The Bayesian classifier for the case of Gaussian distributed classes partitions the feature space via quadrics.
(a) The case of an ellipse and (b) the case of a hyperbola.

$$(a) \quad \boldsymbol{\mu}_1 = [0, 0]^T, \quad \boldsymbol{\mu}_2 = [4, 0]^T, \quad \Sigma_1 = \begin{bmatrix} 0.3 & 0.0 \\ 0.0 & 0.35 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1.2 & 0.0 \\ 0.0 & 1.85 \end{bmatrix},$$

and

$$(b) \quad \boldsymbol{\mu}_1 = [0, 0]^T, \quad \boldsymbol{\mu}_2 = [3.2, 0]^T, \quad \Sigma_1 = \begin{bmatrix} 0.1 & 0.0 \\ 0.0 & 0.75 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 0.75 & 0.0 \\ 0.0 & 0.1 \end{bmatrix},$$

respectively. In Figure 7.4a, the resulting curve for scenario (a) is an ellipse, and in Figure 7.4b, the corresponding curve for scenario (b) is a hyperbola.

Looking carefully at (7.15), it is readily noticed that once the covariance matrices for the two classes become equal, then the quadratic terms cancel out and the discriminant function becomes linear; thus, the corresponding hypersurface is a hyperplane. That is, under the previous assumptions, the optimal Bayesian classifier becomes a *linear* classifier, which after some straightforward algebraic manipulations (try it) can be written as

$$g(\mathbf{x}) = \boldsymbol{\theta}^T(\mathbf{x} - \mathbf{x}_0) = 0, \tag{7.16}$$

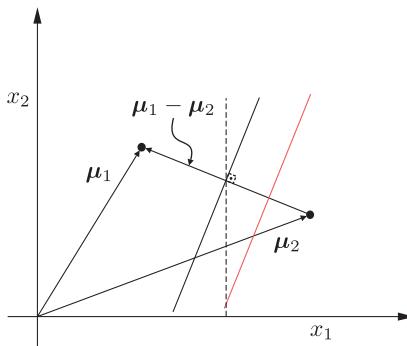
$$\boldsymbol{\theta} := \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2), \tag{7.17}$$

$$\mathbf{x}_0 := \frac{1}{2}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) - \ln \frac{P(\omega_1)}{P(\omega_2)} \frac{\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2}{\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|_{\Sigma^{-1}}}, \tag{7.18}$$

where Σ is common to the two-class covariance matrix and

$$\|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|_{\Sigma^{-1}} := \sqrt{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)},$$

is the Σ^{-1} -norm of the vector $(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$; alternatively, this is also known as the *Mahalanobis distance* between $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$. For $\Sigma = I$ this becomes the Euclidean distance.

**FIGURE 7.5**

The full gray line corresponds to the Bayesian classifier for two equiprobable Gaussian classes that share a common covariance matrix of the specific form, $\Sigma = \sigma^2 I$; the line bisects the segment joining the two mean values (minimum Euclidean distance classifier). The red one is for the same case but for $P(\omega_1) > P(\omega_2)$. The dotted line is the optimal classifier for equiprobable classes, and a common covariance of a more general form, different than $\sigma^2 I$ (minimum Mahalanobis distance classifier).

[Figure 7.5](#) shows three cases for the two-dimensional space. The full black line corresponds to the case of equiprobable classes with a covariance matrix of the special form, $\Sigma = \sigma^2 I$. The corresponding decision hyperplane is given by

$$g(x) = (\mu_1 - \mu_2)^T (x - x_0) = 0. \quad (7.19)$$

The separating line (hyperplane) crosses the middle point of the line segment joining the mean value points, μ_1 and μ_2 ($x_0 = \frac{1}{2}(\mu_1 + \mu_2)$). Also, it is perpendicular to this segment, defined by the vector $\mu_1 - \mu_2$, as is readily verified by the above hyperplane definition. The red line corresponds to the case where $P(\omega_1) > P(\omega_2)$. It gets closer to the mean value point of class ω_2 , thus enlarging the region where one decides in favor of the more probable class. Finally, the dotted line corresponds to the equiprobable case with the common covariance matrix being of a more general form, $\Sigma \neq \sigma^2 I$. The separating hyperplane crosses x_0 but it is rotated in order to be perpendicular to the vector $\Sigma^{-1}(\mu_1 - \mu_2)$, according to (7.16)–(7.17). An unknown point is classified according to the side of the respective hyperplane on which it lies.

What was said before for the two-class task is generalized to the more general M -class problem; the separating hypersurfaces of two contiguous regions, $\mathcal{R}_i, \mathcal{R}_j$, associated with two classes, ω_i and ω_j , obey the same arguments as the ones adopted before. For example, assuming that all covariance matrices are the same, then the regions are partitioned via hyperplanes, as illustrated in [Figure 7.6](#). Moreover, each region \mathcal{R}_i , $i = 1, 2, \dots, M$, is convex ([Problem 7.2](#)); in other words, joining any two points within \mathcal{R}_i , all the points lying on the respective segment lie in \mathcal{R}_i , too.

Two special cases are of particular interest, leading to a simple classification rule. The rule will be expressed for the general M -class problem.

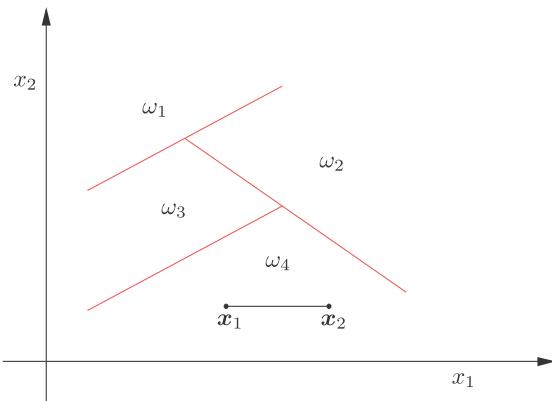


FIGURE 7.6

When data are distributed according to the Gaussian distribution and they share the same covariance matrix in all classes, the feature space is partitioned via hyperplanes, which form polyhedral regions. Note that each region is associated with one class and it is convex.

Minimum distance classifiers

- *Minimum Euclidean distance classifier:* Under the assumptions of (a) Gaussian distributed data in each one of the classes, (b) equiprobable classes, and (c) common covariance matrix in all classes of the special form $\Sigma = \sigma^2 I$ (individual features are independent and share a common variance), the Bayesian classification rule is equivalent with

$$\text{Assign } \mathbf{x} \text{ to class } \omega_i : i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T (\mathbf{x} - \boldsymbol{\mu}_j), \quad j = 1, 2, \dots, M. \quad (7.20)$$

This is a direct consequence of the Bayesian rule under the adopted assumptions. In other words, the Euclidean distance of \mathbf{x} is computed from the mean values of all classes and it is assigned to the class for which this distance becomes smaller.

For the case of the two classes, this classification rule corresponds to the full black line of [Figure 7.5](#). Indeed, recalling our geometry basics, any point that lies on the left of this hyperplane is closer to $\boldsymbol{\mu}_1$ than to $\boldsymbol{\mu}_2$. The opposite is true for any point lying on the right of the hyperplane.

- *Minimum Mahalanobis distance classifier:* Under the previously adopted assumptions, but with the covariance matrix being of the more general form, $\Sigma \neq \sigma^2 I$, the rule becomes

$$\text{Assign } \mathbf{x} \text{ to class } \omega_i : i = \arg \min_j (\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_j), \quad j = 1, 2, \dots, M. \quad (7.21)$$

Thus, instead of looking for the minimum Euclidean distance, one searches for the minimum Mahalanobis distance; the latter is a weighted form of the Euclidean distance, in order to account for the shape of the underlying Gaussian distributions [52]. For the two-class case, this rule corresponds to the dotted line of [Figure 7.5](#).

Remarks 7.2.

- In Statistics, adopting the Gaussian assumption for the data distribution is sometimes called *linear discriminant analysis* (LDA) or *quadratic discriminant analysis* (QDA), depending on the adopted assumptions with respect to the underlying covariance matrices, which will lead to either linear or quadratic discriminant functions, respectively. In practice, the ML method is usually employed in order to obtain estimates of the unknown parameters, namely, the mean values and the covariance matrices. Recall from [Example 3.5 of Chapter 3](#) that the ML estimate of the mean value of a Gaussian pdf, obtained via N observations, \mathbf{x}_n , $n = 1, 2, \dots, N$, is equal to

$$\hat{\boldsymbol{\mu}}_{ML} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n.$$

Moreover, the ML estimate of the covariance matrix of a Gaussian distribution, using N observations, is given by ([Problem 7.4](#)),

$$\hat{\Sigma}_{ML} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})^T. \quad (7.22)$$

This corresponds to a biased estimator of the covariance matrix. An unbiased estimator results if ([Problem 7.5](#)),

$$\hat{\Sigma} = \frac{1}{N-1} \sum_{n=1}^N (\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_{ML})^T.$$

Note that the number of parameters to be estimated in the covariance matrix is $O(l^2/2)$, taking into account its symmetry.

Example 7.2.

Consider a two-class classification task in the two-dimensional space with, $P(\omega_1) = P(\omega_2) = 1/2$. Generate 100 points, 50 from each class. The data from each class, ω_i , $i = 1, 2$, stem from a corresponding Gaussian, $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$, where

$$\boldsymbol{\mu}_1 = [0, -2]^T, \quad \boldsymbol{\mu}_2 = [0, 2]^T,$$

and (a)

$$\Sigma_1 = \Sigma_2 = \begin{bmatrix} 1.2 & 0.4 \\ 0.4 & 1.2 \end{bmatrix},$$

or (b)

$$\Sigma_1 = \begin{bmatrix} 1.2 & 0.4 \\ 0.4 & 1.2 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1 & -0.4 \\ -0.4 & 1 \end{bmatrix}.$$

[Figure 7.7](#) shows the decision curves formed by the Bayesian classifier. Observe that in the case of [Figure 7.7a](#), the classifier turns out to be a linear one, while for the case of [Figure 7.7b](#), it is nonlinear of a parabola shape.

Example 7.3. In a two-class classification task, the data in each one of the classes are distributed according to the Gaussian distribution, with mean values, $\boldsymbol{\mu}_1 = [0, 0]^T$ and $\boldsymbol{\mu}_2 = [3, 3]^T$, respectively, sharing a common covariance matrix

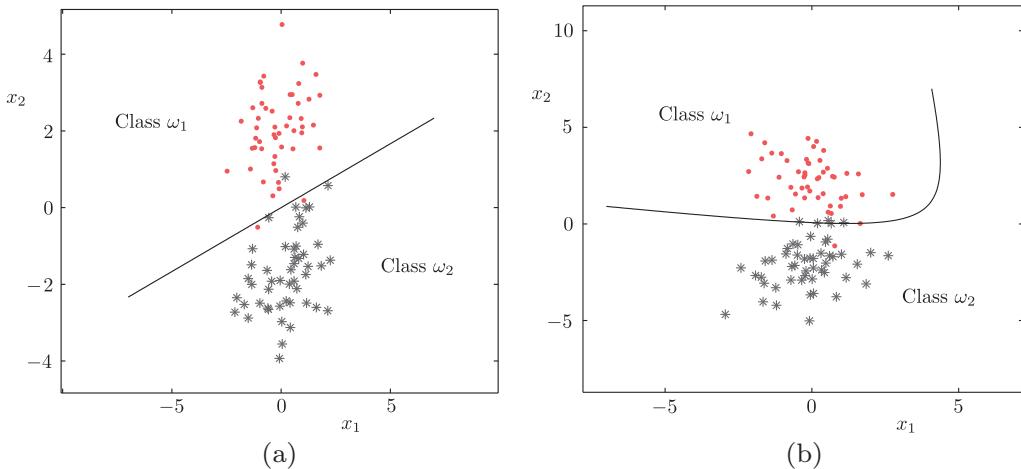


FIGURE 7.7

If the data in the feature space follow a Gaussian distribution in each one of the classes, then the Bayesian classifier is (a) a hyperplane, if all the covariance matrices are equal; (b) otherwise, it is a quadric hypersurface.

$$\Sigma = \begin{bmatrix} 1.1 & 0.3 \\ 0.3 & 1.9 \end{bmatrix}.$$

Use the Bayesian classifier to classify the point $x = [1.0, 2.2]^T$ into one of the two classes.

Because the classes are distributed according to the Gaussian distribution and share the same covariance matrix, the Bayesian classifier is equivalent with the minimum Mahalanobis distance classifier. The (square) Mahalanobis distance of the point x from the mean value of class ω_1 is

$$d_1^2 = [1.0, 2.2] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.2 \end{bmatrix} = 2.95,$$

where the matrix in the middle on the left-hand side is the inverse of the covariance matrix. Similarly for class ω_2 , we obtain that

$$d_2^2 = [-2.0, -0.8] \begin{bmatrix} 0.95 & -0.15 \\ -0.15 & 0.55 \end{bmatrix} \begin{bmatrix} -2.0 \\ -0.8 \end{bmatrix} = 3.67.$$

Hence, the pattern is assigned to class ω_1 , because its distance from μ_1 is smaller compared to that from μ_2 . Verify that if the Euclidean distance were used instead, the pattern would be assigned to class ω_2 .

7.4 THE NAIVE BAYES CLASSIFIER

We have already seen that in case the covariance matrix is to be estimated, the number of unknown parameters is of the order of $\mathcal{O}(l^2/2)$. For high dimensional spaces, besides the fact that this estimation task is a formidable one, it also requires a large number of data points, in order to obtain statistically

good estimates and avoid overfitting, as discussed in [Chapter 3](#). In such cases, one has to be content with suboptimal solutions. Indeed, adopting an optimal method, while using bad estimates of the involved parameters can lead to a bad overall performance.

The *naive Bayes classifier* is a typical and popular example of a suboptimal classifier. The basic assumption is that the components (features) in the feature vector are statistically independent; hence, the joint pdf can be written as a product of l marginals,

$$p(\mathbf{x}|\omega_i) = \prod_{k=1}^l p(x_k|\omega_i), \quad i = 1, 2, \dots, M.$$

Having adopted the Gaussian assumption, each one of the marginals is described by two parameters, the mean and the variance; this leads to a total of $2l$ per class, unknown parameters to be estimated. This is a substantial saving compared to the $O(l^2/2)$ number of parameters. It turns out that this simplistic assumption can end up with better results compared to the optimal Bayes classifier, when the size of the data samples is limited.

Although the naive Bayes classifier was introduced in the context of Gaussian distributed data, its use is also justified for the more general case. In [Chapter 3](#), we discussed the curse of dimensionality issue and it was stressed that high dimensional spaces are sparsely populated. In other words, for a fixed finite number of data points, N , within a cube of fixed size for each dimension, the larger the dimension of the space, the larger the average distance between any two points becomes. Hence, in order to get good estimates of a set of parameters in large spaces, increased number of data is required. Roughly speaking, if N data points are needed in order to get a good enough estimate of a pdf in the real axis, as happens when using the histogram method, N^l data points would be needed for similar accuracy in an l -dimensional space. Thus, by assuming the features to be mutually independent, one will end up estimating l one-dimensional pdfs, hence substantially reducing the need for data.

The independence assumption is a common one in a number of machine learning and statistics tasks. As we will see in [Chapter 15](#), one can adopt more “mild” independence assumptions that lie in between the two extremes, which are full independence and full dependence.

7.5 THE NEAREST NEIGHBOR RULE

Although the Bayesian rule provides the optimal solution with respect to the classification error probability, its application requires the estimation of the respective conditional pdfs; this is not an easy task, once the dimensionality of the feature space assumes relatively large values. This paves the way for considering alternative classification rules, which becomes our focus from now on.

The *k-nearest neighbor* (*k*NN) rule is a typical nonparametric classifier and it is one among the most popular and well-known classifiers. In spite of its simplicity, it is still in use and stands next to more elaborate schemes.

Consider N training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, for an M -class classification task. At the heart of the method lies a parameter k , which is a user-defined parameter. Once k is selected, then given a pattern, \mathbf{x} , assign it to the class in which the majority of its k nearest (according to a metric, e.g., Euclidean or Mahalanobis distance) neighbors, among the training points, belong. The parameter k should not be a multiple of M , in order to avoid ties. The simplest form of this rule is to assign the pattern to the class in which its nearest neighbor belongs, meaning $k = 1$.

It turns out that this conceptually simple rule tends to the Bayesian classifier if (a) $N \rightarrow \infty$, (b) $k \rightarrow \infty$, and (c) $k/N \rightarrow 0$. More specifically, it can be shown that the classification errors P_{NN} and $P_{k\text{NN}}$ satisfy, asymptotically, the following bounds [14],

$$P_B \leq P_{\text{NN}} \leq 2P_B, \quad (7.23)$$

for the $k = 1$ NN rule and,

$$P_B \leq P_{k\text{NN}} \leq P_B + \sqrt{\frac{2P_{\text{NN}}}{k}}, \quad (7.24)$$

for the more general k -NN version. P_B is the error corresponding to the optimal Bayesian classifier. The previous two formulae are quite interesting. Take for example (7.23). It says that the simple NN rule will never give an error larger than twice the optimal one. If, for example, $P_B = 0.01$, then $P_{\text{NN}} \leq 0.02$. This is not bad for such a simple classifier. All this says is that if one has an easy task (as indicated by the very low value of P_B), the NN rule can also do a good job. This, of course, is not the case if the problem is not an easy one and larger error values are involved. The bound in (7.24) says that for large values of k (provided, of course, N is large enough), the performance of the k -NN tends to that of the optimal classifier. In practice, one has to make sure that k does not get values close to N , but remains a relatively small fraction of it.

One may wonder how a performance close to the optimal classifier can be obtained, even in theory and asymptotically, because the Bayesian classifier exploits the statistical information for the data distribution while the k -NN does not take into account such information. The reason is that if N is a very large value (hence the space is densely populated) and k is a relatively small number, with respect to N , then the nearest neighbors will be located very close to \mathbf{x} . Then, due to the *continuity* of the involved pdfs, the values of their posterior probabilities will be close to $P(\omega_i|\mathbf{x})$, $i = 1, 2, \dots, M$. Furthermore, for large enough k , the majority of the neighbors must come from the class that scores the maximum value of the posterior probability given \mathbf{x} .

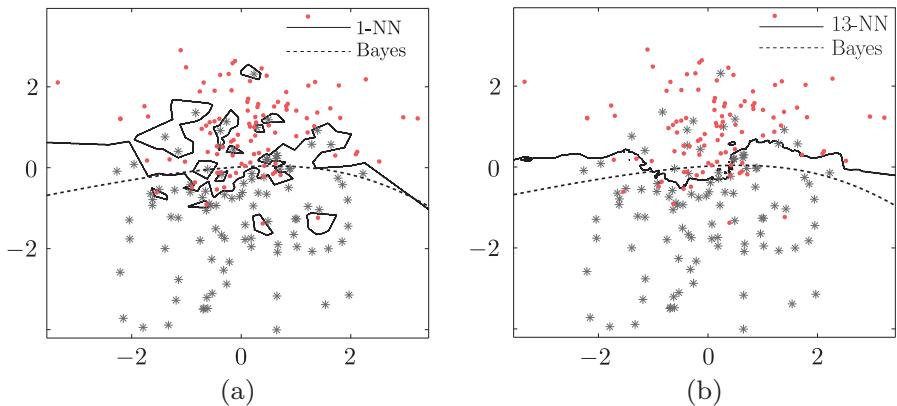
A major drawback of the k -NN rule is that every time a new pattern is considered, its distance from all the training points has to be computed, then selecting the k closest to it points. To this end, various searching techniques have been suggested over the years. The interested reader may consult [52] for a related discussion.

Remarks 7.3.

- The use of the k -nearest rule concept can also be adopted in the context of the regression task. Given an observation, \mathbf{x} , one searches for its k closer input vectors in the training set, denoted as $\mathbf{x}_{(1)}, \dots, \mathbf{x}_{(k)}$, and computes an estimate of the output value, \hat{y} , as an average of the respective outputs in the training set, represented by

$$\hat{y} = \frac{1}{k} \sum_{i=1}^k y_{(i)}.$$

Example 7.4. An example that illustrates the decision curves for a two-class classification task in the two-dimensional space, obtained by the Bayesian, the 1-NN and the 13-NN classifier, is given in Figure 7.8. A number of $N = 100$ data are generated for each class by Gaussian distributions. The decision curve of the Bayes classifier has the form of a parabola, while the 1-NN classifier exhibits a highly nonlinear nature. The 13-NN rule forms a decision line close to the Bayesian one.

**FIGURE 7.8**

A two-class classification task. The dotted curve corresponds to the optimal Bayesian classifier. The full line curves correspond to (a) the 1-NN and (b) the 13-NN classifiers. Observe that the 13-NN is closer to the Bayesian one.

7.6 LOGISTIC REGRESSION

In Bayesian classification, the assignment of a pattern in a class is performed based on the posterior probabilities, $P(\omega_i|\mathbf{x})$. The posteriors are estimated via the respective conditional pdfs, which is not, in general, an easy task. The goal in this section is to model the posterior probabilities directly, via the *logistic regression* method. This name has been established in the statistics community, although the model refers to classification and not to regression. This is a typical example of the discriminative modeling approach, where the distribution of data is of no interest.

The two-class case: The starting point is to model the ratio of the posteriors as

$$\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} = \boldsymbol{\theta}^T \mathbf{x} : \quad \text{Two-class Logistic Regression,} \quad (7.25)$$

where the constant term, θ_0 , has been absorbed in $\boldsymbol{\theta}$. Taking into account that

$$P(\omega_1|\mathbf{x}) + P(\omega_2|\mathbf{x}) = 1,$$

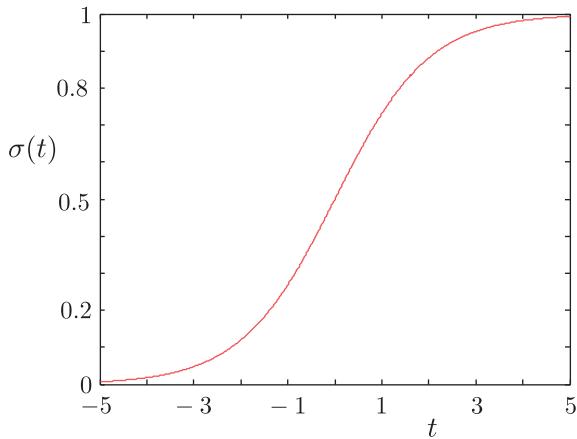
and defining

$$t := \boldsymbol{\theta}^T \mathbf{x},$$

it is readily seen that the model in (7.25) is equivalent to

$$P(\omega_1|\mathbf{x}) = \sigma(t) \quad (7.26)$$

$$\sigma(t) := \frac{1}{1 + \exp(-t)}, \quad (7.27)$$

**FIGURE 7.9**

The sigmoid link function.

and

$$P(\omega_2|\mathbf{x}) = 1 - P(\omega_1|\mathbf{x}) = \frac{\exp(-t)}{1 + \exp(-t)}. \quad (7.28)$$

The function $\sigma(t)$ is known as the *logistic sigmoid* or *sigmoid link* function and it is shown in Figure 7.9.

Although it may sound a bit mystical as to how one thought of such a model, it suffices to look more carefully at (7.13)–(7.15) to demystify it. Assuming the data in the classes follow Gaussian distributions with $\Sigma_1 = \Sigma_2 \equiv \Sigma$ and for simplicity that $P(\omega_1) = P(\omega_2)$, the latter of the previously stated equations is written as

$$\ln \frac{P(\omega_1|\mathbf{x})}{P(\omega_2|\mathbf{x})} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} \mathbf{x} + \text{constants}. \quad (7.29)$$

In other words, when the distributions underlying the data are Gaussians with a common covariance matrix, then the log ratio of the posteriors is a linear function. Thus, in logistic regression, all we do is adopt such a model, irrespective of the data distribution. Moreover, even if the data are distributed according to Gaussians, it may still be preferable to adopt the logistic regression formulation instead of that in (7.29). In the latter formulation, the covariance matrix has to be estimated, amounting to $\mathcal{O}(l^2/2)$ parameters. The logistic regression formulation only involves $l + 1$ parameters. That is, once we know about the linear dependence of the log ratio on \mathbf{x} , we can use this a priori information to simplify the model. Of course, assuming that the Gaussian assumption is valid, if one can obtain good estimates of the covariance matrix, employing this extra information can lead to more efficient estimates, in the sense of lower variance. The issue is treated in Ref. [18]. This is natural, because more information concerning the distribution of the data is exploited. In practice, it turns out that using the logistic regression is, in general, a safer bet compared to the linear discriminant analysis (LDA).

The parameter vector, θ , is estimated via the ML method applied on the set of training samples, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, $y_n \in \{0, 1\}$. The likelihood function can be written as

$$P(y_1, \dots, y_N; \theta) = \prod_{n=1}^N (\sigma(\theta^\top \mathbf{x}_n))^{y_n} (1 - \sigma(\theta^\top \mathbf{x}_n))^{1-y_n}. \quad (7.30)$$

Usually, we consider the negative log-likelihood given by

$$L(\theta) = - \sum_{n=1}^N (y_n \ln s_n + (1 - y_n) \ln(1 - s_n)), \quad (7.31)$$

where

$$s_n := \sigma(\theta^\top \mathbf{x}_n). \quad (7.32)$$

The log-likelihood cost function in (7.31) is also known as the *cross-entropy* error. Minimization of $L(\theta)$ with respect to θ is carried out iteratively by any iterative minimization scheme, such as the steepest descent or Newton's method. Both schemes need the computation of the respective gradient, which in turn is based on the derivative of the sigmoid link function ([Problem 7.6](#))

$$\frac{d\sigma(t)}{dt} = \sigma(t)(1 - \sigma(t)). \quad (7.33)$$

The gradient is given by ([Problem 7.7](#))

$$\begin{aligned} \nabla L(\theta) &= \sum_{n=1}^N (s_n - y_n) \mathbf{x}_n \\ &= X^\top (s - y), \end{aligned} \quad (7.34)$$

where

$$X^\top = [\mathbf{x}_1, \dots, \mathbf{x}_N], \quad s := [s_1, \dots, s_N]^\top, \quad y = [y_1, \dots, y_N]^\top.$$

The Hessian matrix is given by ([Problem 7.8](#))

$$\begin{aligned} \nabla^2 L(\theta) &= \sum_{n=1}^N s_n(1 - s_n) \mathbf{x}_n \mathbf{x}_n^\top \\ &= X^\top R X, \end{aligned} \quad (7.35)$$

where

$$R := \text{diag}\{s_1(1 - s_1), \dots, s_N(1 - s_N)\}. \quad (7.36)$$

Note that because $0 < s_n < 1$, by definition of the sigmoid link function, matrix R is positive definite; hence, the Hessian matrix is also positive definite ([Problem 7.9](#)). This is a necessary and sufficient condition for convexity.¹ Thus, the negative log-likelihood function is convex, which guarantees the existence of a unique minimum (e.g., [[3](#)]).

¹ Convexity is discussed in more detail in [Chapter 8](#).

Two of the possible iterative minimization schemes to be used are

- Steepest descent

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \mu_i X^T (\mathbf{s}^{(i-1)} - \mathbf{y}). \quad (7.37)$$

- Newton's scheme

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu_i \left(X^T R^{(i-1)} X \right)^{-1} X^T (\mathbf{s}^{(i-1)} - \mathbf{y}) \\ &= \left(X^T R^{(i-1)} X \right)^{-1} X^T R^{(i-1)} \mathbf{z}^{(i-1)} \end{aligned} \quad (7.38)$$

where

$$\mathbf{z}^{(i-1)} := X \boldsymbol{\theta}^{(i-1)} - \left(R^{(i-1)} \right)^{-1} (\mathbf{s}^{(i-1)} - \mathbf{y}). \quad (7.39)$$

Equation (7.38) is a weighted version of the LS solution (Chapters 3 and 6); however, the involved quantities are iteration-dependent and the resulting scheme is known as *iterative reweighted least squares* scheme (IRLS) [50].

Maximizing the likelihood may run into problems if the training data set is linearly separable. In this case, any point on a hyperplane, $\boldsymbol{\theta}^T \mathbf{x} = 0$, that solves the classification task and separates the samples from each class (note that there are infinite many such hyperplanes), results in $\sigma(\mathbf{x}) = 0.5$, and every training point from each class is assigned a posterior probability equal to one. Thus, ML forces the logistic sigmoid to become a step function in the feature space and equivalently $\|\boldsymbol{\theta}\| \rightarrow \infty$. This can lead to overfitting and it is remedied by including a regularization term, $\|\boldsymbol{\theta}\|^2$, in the respective cost function.

The M-class case: For the more general M -class classification task, the logistic regression is defined for $m = 1, 2, \dots, M$, as

$$P(\omega_m | \mathbf{x}) = \frac{\exp(\boldsymbol{\theta}_m^T \mathbf{x})}{\sum_{j=1}^M \exp(\boldsymbol{\theta}_j^T \mathbf{x})} : \quad \text{Multiclass Logistic Regression.}$$

(7.40)

The previous definition is easily brought into the form of a linear model for the log ratio of the posteriors. Divide, for example, by $P(\omega_M | \mathbf{x})$ to obtain

$$\ln \frac{P(\omega_m | \mathbf{x})}{P(\omega_M | \mathbf{x})} = (\boldsymbol{\theta}_m - \boldsymbol{\theta}_M)^T \mathbf{x} = \hat{\boldsymbol{\theta}}_m^T \mathbf{x}.$$

Let us define, for notational convenience,

$$\phi_{nm} := P(\omega_m | \mathbf{x}_n), \quad n = 1, 2, \dots, N, \quad m = 1, 2, \dots, M,$$

and

$$t_m := \boldsymbol{\theta}_m^T \mathbf{x}, \quad m = 1, 2, \dots, M.$$

The likelihood function is now written as

$$P(\mathbf{y}; \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M) = \prod_{n=1}^N \prod_{m=1}^M (\phi_{nm})^{y_{nm}}, \quad (7.41)$$

where $y_{nm} = 1$ if $\mathbf{x}_n \in \omega_m$ and zero otherwise. The respective negative log-likelihood function becomes

$$L(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M) = - \sum_{n=1}^N \sum_{m=1}^M y_{nm} \ln \phi_{nm}. \quad (7.42)$$

Minimization with respect to $\boldsymbol{\theta}_m$, $m = 1, \dots, M$, takes place iteratively. To this end, the following gradients are used (Problems 7.10-7.12):

$$\frac{\partial \phi_{nm}}{\partial t_j} = \phi_{nm} (\delta_{mj} - \phi_{nj}), \quad (7.43)$$

where δ_{mj} is one if $m = j$ and zero otherwise. Also,

$$\nabla_{\boldsymbol{\theta}_j} L(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M) = \sum_{n=1}^N (\phi_{nj} - y_{nj}) \mathbf{x}_n. \quad (7.44)$$

The respective Hessian matrix is an $(lM) \times (lM)$ matrix, comprising $l \times l$ blocks. Its k, j block element is given by

$$\nabla_{\boldsymbol{\theta}_k} \nabla_{\boldsymbol{\theta}_j} L(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M) = \sum_{n=1}^N \phi_{nj} (\delta_{kj} - \phi_{nk}) \mathbf{x}_n \mathbf{x}_n^T. \quad (7.45)$$

The Hessian matrix is also positive definite, which guarantees uniqueness of the minimum as in the two-class case.

Remarks 7.4.

- *Probit regression:* Instead of using the logistic sigmoid function in (7.26) (for the two-class case), other functions can also be adopted. A popular function in the statistical community is the *probit* function, which is defined as

$$\begin{aligned} \Phi(t) &:= \int_{-\infty}^t \mathcal{N}(z|0, 1) dz \\ &= \frac{1}{2} \left(1 + \frac{1}{\sqrt{2}} \text{erf}(t) \right), \end{aligned} \quad (7.46)$$

where erf is the *error* function defined as

$$\text{erf}(t) = \frac{2}{\sqrt{\pi}} \int_0^t \exp\left(-\frac{z^2}{2}\right) dz.$$

In other words, $P(\omega_1|t)$ is modeled to be equal to the probability of a normalized Gaussian variable to lie in the interval $(-\infty, t]$. The graph of the probit function is very similar to that of the logistic one.

7.7 FISHER'S LINEAR DISCRIMINANT

We now turn our focus to designing linear classifiers. In other words, irrespective of the data distribution in each class, we decide to partition the space in terms of hyperplanes, so that

$$g(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0 = 0. \quad (7.47)$$

We have dealt with the task of designing linear classifiers in the framework of the LS cost in [Chapter 3](#). In this section, the unknown parameter vector will be estimated via a path that exploits a number of important notions relevant to classification. The method is known as *Fisher's discriminant* and it can be dressed up with different interpretations. Thus, its significance lies not only in its practical use but also in its pedagogical value.

Two of the major phases in designing a pattern recognition system are the *feature generation* and *feature selection* phases. Selecting information-rich features is of paramount importance. If “bad” features are selected, whatever smart classifier one adopts, the performance is bound to be poor. Feature generation/selection techniques are treated in detail in Refs. [52, 53], to which the interested reader may refer to for further information. At this point, we only touch on a few notions that are relevant to our current design of a linear classifier. Let us first quantify what a “bad” and a “good” feature is. The main goal in selecting features, and, thus, in selecting the feature space in which one is going to work, can be summarized this way: Select the features to create a feature space in which the points, which represent the training patterns, are distributed such as to have

Large Between-Classes Distance
and
Small Within-Class Variance.

[Figure 7.10](#) illustrates three different choices for the case of two-dimensional feature spaces. Common sense dictates that the choice in [Figure 7.10c](#) is the best one; the points in the three classes form groups that lie relatively far away from each other, and at the same time the data in each class are compactly clustered together. The worst of the three choices is that of [Figure 7.10b](#), where data in each class are spread around their mean value and the three groups are relatively close to each other. The goal in feature selection is to develop measures that quantify the “slogan” given in the box above. The notion

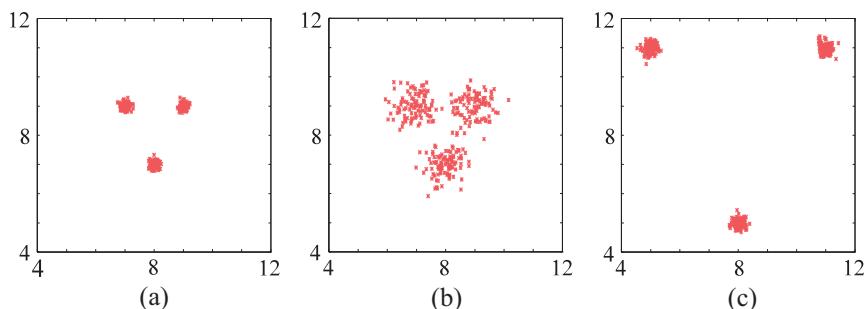


FIGURE 7.10

Three different choices of two-dimensional feature spaces: (a) small within-class variance and small between-classes distance; (b) large within class variance and small between classes distance; and (c) small within class variance and large between-classes distance. The last one is the best choice out of the three.

of *scatter matrices* is of relevance to us here. Although we could live without these definitions, it is a good opportunity to present this important notion and put our discussion in a more general context.

- *Within-class scatter matrix*

$$\Sigma_w = \sum_{k=1}^M P(\omega_k) \Sigma_k, \quad (7.48)$$

where Σ_k is the covariance matrix of the points in the k th among M classes. In words, Σ_w is the average covariance matrix of the data in the specific l -dimensional feature space.

- *Between-classes scatter matrix*

$$\Sigma_b = \sum_{m=1}^M P(\omega_m) (\mu_m - \mu_0)(\mu_m - \mu_0)^T, \quad (7.49)$$

where μ_0 is the overall mean value defined by

$$\mu_0 = \sum_{m=1}^M P(\omega_m) \mu_m. \quad (7.50)$$

Another commonly used related matrix is the following:

- *Mixture scatter matrix*

$$\Sigma_m = \Sigma_w + \Sigma_b. \quad (7.51)$$

A number of criteria that measure the “goodness” of the selected feature space are built around these scatter matrices; three typical examples are: [23, 52]:

$$J_1 := \frac{\text{trace}\{\Sigma_m\}}{\text{trace}\{\Sigma_w\}}, \quad J_2 = \frac{|\Sigma_m|}{|\Sigma_w|}, \quad J_3 = \text{trace}\{\Sigma_w^{-1} \Sigma_b\}, \quad (7.52)$$

where $|\cdot|$ denotes the determinant of a matrix.

The two-class case: In Fisher’s linear discriminant analysis, the emphasis in Eq. (7.47) is only on θ ; the bias term, θ_0 , is left out of the discussion. The inner product $\theta^T x$ can be viewed as the projection of x along the vector θ . From geometry, we know that the respective projection is also a vector, y , given by (e.g., Section 5.6)

$$y = \frac{\theta^T x}{\|\theta\|} \frac{\theta}{\|\theta\|}.$$

From now on, we will focus on the scalar value of the projection, $y := \theta^T x$, and ignore the scaling factor in the denominator, because scaling all features by the same value has no effect on our discussion. The goal, now, is to select that direction, θ , so that after projecting along this direction, (a) the data in the two classes are as far away as possible from each other, and (b) the respective variances of the points around their means, in each one of the classes, are as small as possible. A criterion that quantifies the aforementioned goal is *Fisher’s discriminant ratio* (FDR), defined as

$$\boxed{\text{FDR} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} : \text{ Fisher's Discriminant Ratio,}}$$

(7.53)

where, μ_1 and μ_2 are the (scalar) mean values of the two classes, after the projection along θ , meaning

$$\mu_i = \theta^T \mu_i, \quad i = 1, 2.$$

However, we have that

$$\begin{aligned} (\mu_1 - \mu_2)^2 &= \theta^T (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \theta = \theta^T S_b \theta, \\ S_b &:= (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T. \end{aligned}$$

Note that if the classes are equiprobable, S_b is a scaled version of the between-classes scatter matrix in (7.49) (under this assumption, $\mu_0 = 1/2(\mu_1 + \mu_2)$), and we have

$$(\mu_1 - \mu_2)^2 \propto \theta^T \Sigma_b \theta. \quad (7.54)$$

Moreover,

$$\sigma_i^2 = \mathbb{E}[(y - \mu_i)^2] = \mathbb{E}[\theta^T (x - \mu_i)(x - \mu_i)^T \theta] = \theta^T \Sigma_i \theta, \quad i = 1, 2, \quad (7.55)$$

which leads to

$$\sigma_1^2 + \sigma_2^2 = \theta^T S_w \theta,$$

where $S_w = \Sigma_1 + \Sigma_2$. Note that if the classes are equiprobable, S_w becomes a scaled version of the within-class scatter matrix defined in (7.48), and we have that

$$\sigma_1^2 + \sigma_2^2 \propto \theta^T \Sigma_w \theta. \quad (7.56)$$

Combining (7.53), (7.54), and (7.56) and neglecting the proportionality constants, we end up with

$$\text{FDR} = \frac{\theta^T \Sigma_b \theta}{\theta^T \Sigma_w \theta} : \quad \text{Generalized Rayleigh Quotient.}$$

(7.57)

Our goal now becomes that of maximizing the FDR with respect to θ . This is a case of the *generalized Rayleigh ratio*, and it is known from linear algebra that it is maximized if θ satisfies

$$\Sigma_b \theta = \lambda \Sigma_w \theta,$$

where λ is the maximum eigenvalue of the matrix $\Sigma_w^{-1} \Sigma_b$ (Problem 7.14). However, for our specific case² here, we can bypass the need for solving an eigenvalue-eigenvector problem. Observe that the last equation can be rewritten as

$$\lambda \Sigma_w \theta \propto (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \theta \propto (\mu_1 - \mu_2).$$

In other words, $\Sigma_w \theta$ lies in the direction of $(\mu_1 - \mu_2)$, and because we are only interested in the direction, we can finally write that

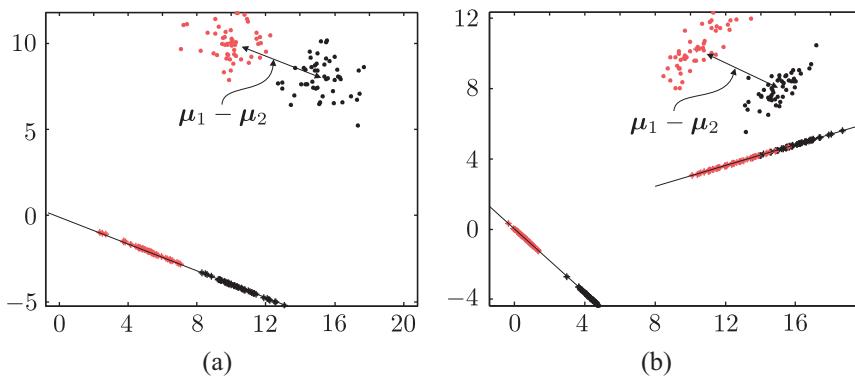
$$\theta = \Sigma_w^{-1}(\mu_1 - \mu_2),$$

(7.58)

assuming of course that Σ_w is invertible. In practice, Σ_w is obtained as the respective sample mean using the available observations.

Figure 7.11a shows the resulting direction for two spherically distributed (isotropic) classes in the two-dimensional space. In this case, the direction for projecting the data is parallel to $(\mu_1 - \mu_2)$.

² Σ_b is a rank-one matrix and there is only one nonzero eigenvalue.

**FIGURE 7.11**

(a) The optimal direction resulting from Fisher's discriminant for two spherically distributed classes. The direction on which projection takes place is parallel to the segment joining the mean values of the data in the two classes. (b) The line on the bottom left of the figure corresponds to the direction that results from Fisher's discriminant; observe that it is no longer parallel to $\mu_1 - \mu_2$. For the sake of comparison, observe that projecting on the other line on the right results in class overlap.

In Figure 7.11b, the distribution of the data in the two classes is not spherical, and the direction of projection (the line to the bottom left of the figure) is not parallel to the segment joining the two mean points. Observe that if the line to the right is selected, then after projection the classes do overlap.

In order for the Fisher's discriminant method to be used as a classifier, a threshold θ_0 must be adopted, and decision in favor of a class is performed according to the rule

$$y = (\mu_1 - \mu_2)^T \Sigma_w^{-1} x + \theta_0 \begin{cases} > 0, \text{ class } \omega_1, \\ < 0, \text{ class } \omega_2. \end{cases} \quad (7.59)$$

Compare, now, (7.59) with (7.16)–(7.18); the latter were obtained via the Bayes rule for the Gaussian case, when both classes share the same covariance matrix. Observe that for this case, the resulting hyperplanes are parallel and the only difference is in the threshold value. Note, however, that the Gaussian assumption was not needed for Fisher's discriminant. This justifies the use of (7.16)–(7.18), even when the data are not normally distributed. In practice, depending on the data, different threshold values may be used.

Finally, because the world is often small, it can be shown that Fisher's discriminant can also be seen as a special case of the LS solution, if the target class labels, instead of ± 1 , are chosen as $\frac{N_1}{N}$ and $\frac{-N_2}{N}$, respectively, where N is the total number of training samples, N_1 is the number of samples in class ω_1 , and N_2 is the corresponding number in class ω_2 [55].

Another point of view for Fisher's discriminant method is that it performs dimensionality reduction by projecting the data from the original l -dimensional space to a lower one-dimensional space. This reduction in dimensionality is performed in a *supervised* way, by exploiting the class labels of the training data. As we will see in Chapter 19, there are other techniques during which the dimensionality reduction takes place in an *unsupervised* way. The obvious question now is whether it is possible to use Fisher's idea in order to reduce the dimensionality not to one but to another intermediate value

between one and l . It turns out that this is possible, but it also depends on the number of classes. More on dimensionality reduction techniques can be found in [Chapter 19](#).

Multiclass Fisher's discriminant: Our starting point is the J_3 criterion defined in [\(7.52\)](#). It can be readily shown that the FDR criterion, used in the two-class case, is directly related to the J_3 one, once the latter is considered for the one-dimensional case and for equiprobable classes. For the more general multiclass formulation, the task becomes that of estimating an $l \times m$, $m < l$ matrix, A , such that the linear transformation from the original \mathbb{R}^l to the new \mathbb{R}^m space, or

$$\mathbf{y} = A^T \mathbf{x}, \quad (7.60)$$

to retain as much classification-related information as possible. Note that in any dimensionality reduction technique, some of the original information is, in general, bound to be lost. Our goal is for the loss to be as small as possible. Because we chose to measure classification-related information by the J_3 criterion, the goal is to compute A in order to maximize

$$J_3(A) = \text{trace}\{\Sigma_{wy}^{-1} \Sigma_{by}\}, \quad (7.61)$$

where Σ_{wy} and Σ_{by} are the within-class and between-classes scatter matrices measured in the transformed lower dimensional space. Maximization follows standard arguments of optimization with respect to matrices. The algebra gets a bit involved and we will state the final result. Details of the proof can be found in Refs. [\[23, 52\]](#). Matrix A is given by the following equation:

$$(\Sigma_{wx}^{-1} \Sigma_{bx})A = A\Lambda. \quad (7.62)$$

Matrix Λ is a diagonal matrix having as elements m of the eigenvalues of the $l \times l$ matrix, $\Sigma_{wx}^{-1} \Sigma_{bx}$, where Σ_{wx} and Σ_{bx} are the within-class and between-classes scatter matrices, respectively, in the original \mathbb{R}^l space. The matrix of interest, A , comprises columns that are the respective eigenvectors. The problem, now, becomes to select the m eigenvalues/eigenvectors. Note that by its definition, Σ_b , being the sum of M related (via μ_0) rank one matrices, is of rank $M - 1$ ([Problem 7.15](#)). Thus, the product $\Sigma_{wx}^{-1} \Sigma_{bx}$ has only $M - 1$ nonzero eigenvalues. This imposes a stringent constraint on the dimensionality reduction. The maximum dimension, m , that one can obtain is $m = M - 1$ (for the two-class task, $m = 1$), irrespective of the original dimension l . There are two cases, that are worth focusing on:

- $m = M - 1$. In this case, it is shown that if A is formed having as columns all the eigenvectors corresponding to the nonzero eigenvalues, then

$$J_{3y} = J_{3x}.$$

In other words, there is no loss of information (as measured via the J_3 criterion) by reducing the dimension from l to $M - 1$! Note that in this case, Fisher's method produces $m = M - 1$ discriminant (linear) functions. This complies with a general result in classification stating that the minimum number of discriminant functions needed for an M -classification problem is $M - 1$ [\[52\]](#). Recall that in Bayesian classification, we need M functions, $P(\omega_i|\mathbf{x})$, $i = 1, 2, \dots, M$; however, only $M - 1$ of those are independent, because they must all add to one. Hence, Fisher's method provides the minimum number of linear discriminants required.

- $m < M - 1$. If A is built having as columns the eigenvectors corresponding to the maximum m eigenvalues, then

$$J_{3y} < J_{3x}.$$

However, the resulting value J_{3y} is the maximum possible one.

Remarks 7.5.

- If J_3 is used with other matrix combinations, as might be achieved by using Σ_m in place of Σ_b , the constraint of the rank being equal to $M - 1$ is removed, and larger values for m can be obtained.
- In a number of practical cases, Σ_w may not be invertible. This is, for example, the case in the *small sample size* problems, where the dimensionality of the feature space, l , may be larger than the number of the training data, N . Such problems may be encountered in applications such as web-document classification, gene expression profiling, and face recognition. There are different escape routes in this problem; see [52] for a discussion and related references.

7.8 CLASSIFICATION TREES

Classification trees are based on a simple, yet powerful, idea, and they are among the most popular techniques for classification. They are *multistage* systems, and classification of a pattern into a class is achieved *sequentially*. Through a series of tests, classes are *rejected* in a sequential fashion until a decision is finally reached in favor of one remaining class. Each one of the tests, whose outcome decides which classes are rejected, is of a *binary* “Yes” or “No” type and is applied to a *single* feature. Our goal is to present the main philosophy around a special type of trees known as *ordinary binary classification trees* (OBCT). They belong to a more general class of methods that construct trees, both for classification as well as regression, known as *classification and regression trees* (CART) [4, 45]. Variants of the method have also been proposed [49].

The basic idea around OBCTs is to partition the feature space into (*hyper*) *rectangles*; that is, the space is partitioned via hyperplanes, which are parallel to the axes. This is illustrated in Figure 7.12.

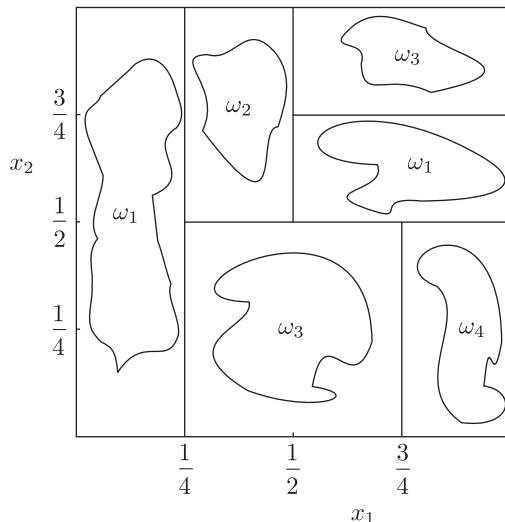
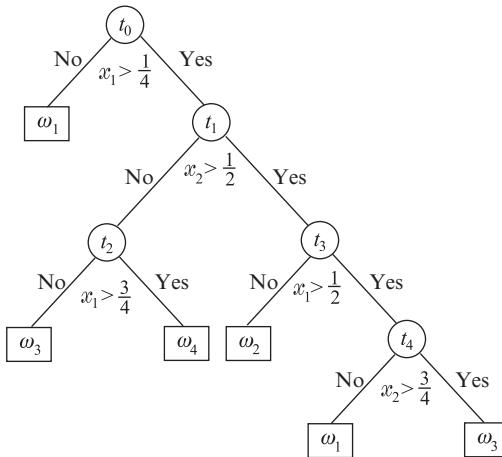


FIGURE 7.12

Partition of the two-dimensional features space, corresponding to three classes, via a classification (OBCT) tree.

**FIGURE 7.13**

The classification tree that performs the space partitioning for the task indicated in [Figure 7.12](#).

The partition of the space in (hyper)rectangles is performed via a series of “questions” of this form: is the value of the feature $x_i < a$? This is also known as the *splitting criterion*. The sequence of questions can nicely be realized via the use of a tree. [Figure 7.13](#) shows the tree corresponding to the case illustrated in [Figure 7.12](#). Each node of the tree performs a test against an *individual* feature and, if it is not a leaf node, it is connected to two *descendant* nodes: one is associated with the answer “Yes” and the other with the answer “No.”

Starting from the root node, a path of successive decisions is realized until a leaf node is reached. Each leaf node is associated with a *single* class. The assignment of a point to a class is done according to the label of the respective leaf node. This type of classification is conceptually simple and easily interpretable. For example, in a medical diagnosis system, one may start with a question: is the temperature high? If yes, a second question can be: is the nose runny? The process carries on until a final decision concerning the disease has been reached. Also, trees are useful in building up reasoning systems in artificial intelligence [51]. For example, the existence of specific objects, which is deduced via a series of related questions based on the values of certain (high-level) features, can lead to the recognition of a scene or of an object depicted in an image.

Once a tree has been developed, classification is straightforward. The major challenge lies in constructing the tree, by exploiting the information that resides in the training data set. The main questions one is confronted with while designing a tree are:

- Which splitting criterion should be adopted?
- When should one stop growing a tree and declare a node as final?
- How is a leaf node associated with a specific class?

Besides the above issues, there are more that will be discussed later on.

Splitting criterion: We have already stated that the questions asked at each node are of the type

$$\text{is } x_i < a?$$

The goal is to select an appropriate value for the threshold value a . Assume that starting from the root node, the tree has grown up to the current node, t . Each node, t , is associated with a subset $X_t \subseteq X$ of the training data set, X . This is the set of the training points that have survived to this node, after the tests that have taken place at the previous nodes in the tree. For example, in Figure 7.13, a number of points, which belong to, say, class ω_1 , will not be involved in node t_1 because they have already been assigned in a previously labeled leaf node. The purpose of a splitting criterion is to split X_t into two *disjoint* subsets, namely X_{tY} , and X_{tN} , depending on the answer to the specific question at node t . For every split, the following is true:

$$X_{tY} \cap X_{tN} = \emptyset,$$

$$X_{tY} \cup X_{tN} = X_t.$$

The goal in each node is to select which feature is to be tested and also what is the best value of the corresponding threshold value a . The adopted philosophy is to make the choice so that every split generates sets, X_{tY} , X_{tN} , which are more *class-homogeneous* compared to X_t . In other words, the data in each one of the two descendant sets must show a higher preference to specific classes, compared to the ancestor set. For example, assume that the data in X_t consist of points that belong to four classes, ω_1 , ω_2 , ω_3 , ω_4 . The idea is to perform the splitting so that most of the data in X_{tY} belong to, say, ω_1 , ω_2 and most of the data in X_{tN} to ω_3 , ω_4 . In the adopted terminology, the sets X_{tY} and X_{tN} should be *purer* compared to X_t . Thus, we must first select a criterion that measures *impurity* and then compute the threshold value and choose the specific feature (to be tested) to maximize the decrease in node impurity. For example, a common measure to quantify impurity of node, t , is the *entropy*, defined as

$$I(t) = - \sum_{m=1}^M P(\omega_m|t) \log_2 P(\omega_m|t), \quad (7.63)$$

where $\log_2(\cdot)$ is the base-two logarithm. The maximum value of $I(t)$ occurs if all probabilities are equal (maximum impurity), and the smallest value, which is equal to zero, when only one of the probability values is one and the rest equal zero. Probabilities are approximated as

$$P(\omega_m|t) = \frac{N_t^m}{N_t}, \quad m = 1, 2, \dots, M,$$

where, N_t^m is the number of the points from class m in X_t , and N_t the total number of points in X_t . The decrease in node impurity, after splitting the data into two sets, is defined as

$$I(t) = \sum_{m=1}^M P(\omega_m|t) (1 - P(\omega_m|t)). \quad (7.64)$$

where $I(t_Y)$ and $I(t_N)$ are the impurities associated with the two new sets, respectively. The goal now becomes to select the specific feature, x_i , and the threshold a_i so that $\Delta I(t)$ becomes maximum. This will now define two new descendant nodes of t , namely, t_N and t_Y ; thus, the tree grows with two new nodes. A way to search for different threshold values is the following: For each one of the features, x_i , $i = 1, 2, \dots, l$, rank the values, x_{in} , $n = 1, 2, \dots, N_t$, which this feature takes among the training points in X_t . Then define a sequence of corresponding threshold values, a_{in} , to be halfway between consecutive distinct values of x_{in} . Then test the impurity change that occurs for each one of these

threshold values and keep the one that achieves the maximum decrease. Repeat the process for all features, and finally, keep the combination that results in the best maximum decrease.

Besides entropy, other impurity measuring indices can be used. A popular alternative, which results in a slightly sharper maximum compared to the entropy one, is the so-called *Gini index*, defined as

$$I(t) = \sum_{m=1}^M P(\omega_m|t)(1 - P(\omega_m|t)). \quad (7.65)$$

This index is also zero if one of the probability values is equal to 1 and the rest are zero, and it takes its maximum value when all classes are equiprobable.

Stop-splitting rule: The obvious question when growing a tree is when to stop growing it. One possible way is to adopt a threshold value, T , and stop splitting a node once the maximum value $\Delta I(t)$, for all possible splits, is smaller than T . Another possibility is to stop when the cardinality of X_t becomes smaller than a certain number or if the node is pure, in the sense that all points in it belong to a single class.

Class assignment rule: Once a node, t , is declared to be a leaf node, it is assigned a class label; usually this is done on a majority voting rationale. That is, it is assigned the label of the class where most of the data in X_t belong.

Pruning a tree: Experience has shown that growing a tree and using a stopping rule does not always work well in practice; growing may either stop early or may result in trees of very large size. A common practice is to first grow a tree up to a large size and then adopt a pruning technique to eliminate nodes. Different pruning criteria can be used; a popular one is to combine an estimate of the error probability with a complexity measuring index; see [4, 45].

Remarks 7.6.

- Among the notable advantages of decision trees is the fact that they can naturally treat mixtures of numeric and categorical variables. Moreover, they scale well with large data sets. Also, they can treat missing variables in an effective way. In many domains, not all the values of the features are known for every pattern. The values may have gone unrecorded, or they may be too expensive to obtain. Finally, due to their structural simplicity, they are easily interpretable; in other words, it is possible for a human to understand the reason for the output of the learning algorithm. In some applications, such as in financial decisions, this is a legal requirement.

On the other hand, the prediction performance of the tree classifiers is not as good as other methods, such as support vector machines and neural networks, to be treated in [Chapters 11](#) and [18](#), respectively.

- A major drawback associated with the tree classifiers is that they are *unstable*. That is, a small change in the training data set can result in a very different tree. The reason for this lies in the hierarchical nature of the tree classifiers. An error that occurs in a node at a high level of the tree propagates all the way down to the leaves below it.

Bagging (Bootstrap Aggregating) [5] is a technique that can reduce the variance and improve the generalization error performance. The basic idea is to create a number of B variants, X_1, X_2, \dots, X_B , of the training set, X , using *bootstrap* techniques, by uniformly sampling from X with replacement. For each of the training set variants, X_i , a tree, T_i , is constructed. The final decision for the classification of a given point is in favor of the class predicted by the majority of the sub-classifiers, T_i , $i = 1, 2, \dots, B$.

Random forests use the idea of bagging in tandem with random feature selection [7]. The difference with bagging lies in the way the decision trees are constructed. The feature to split in each node is selected as the best among a set of F randomly chosen features, where F is a user-defined parameter. This extra introduced randomness is reported to have a substantial effect in performance improvement.

Random forests often have very good predictive accuracy and have been used in a number of applications, including for body pose recognition in terms of Microsoft’s popular Kinect sensor [48].

Besides the previous methods, more recently, Bayesian techniques have also been suggested and used to stabilize the performance of trees; see [11, 58]. Of course, the effect of using multiple trees is losing a main advantage of the trees, that is, their fairly easy interpretability.

- Besides the OBCT rationale, a more general partition of the feature space has also been proposed via hyperplanes that are not parallel to the axes. This is possible via questions of the type: *Is $\sum_{i=1}^l c_i x_i < a$?* This can lead to a better partition of the space. However, the training now becomes more involved; see [49].
- Decision trees have also been proposed for regression tasks, albeit with less success. The idea is to split the space into regions, and prediction is performed based on the average of the output values in the region where the observed input vector lies; such an averaging approach has as a consequence the lack of smoothness, as one moves from one region to another, which is a major drawback of regression trees. The splitting into regions is performed based on the LS criterion [26].

7.9 COMBINING CLASSIFIERS

So far, we have discussed a number of classifiers, and more methods will be presented in Chapters 11, 13 and 18 concerning support vector machines, Bayesian methods, and neural/deep networks. The obvious question an inexperienced practitioner/researcher is confronted with is, which method then? Unfortunately, there is no definitive answer.

No free lunch theorem: The goal of the design of any classifier, and in general of any learning scheme, is to provide a good generalization performance. However, there are no context-independent or usage-independent reasons to support one learning technique over another. Each learning task, represented by the available data set, will show a preference for a specific learning scheme that fits the specificities of the particular problem at hand. An algorithm that scores tops in one problem can score low for another. This is sometimes summarized as the no free lunch theorem [57].

In practice, one should try different learning methods from the available palette, each optimized to the specific task, and test its generalization performance against an independent data set different from the one used for training, using, for example, the leave-one-out-method or any of its variants (Chapter 3). Then, keep and use the method that scored best for the specific task.

To this end, there are a number of major efforts to compare different classifiers against different data sets and measure the “average” performance, via the use of different statistical indices in order to quantify the overall performance of each classifier against the data sets.

Experimental comparisons

One of the very first efforts, to compare the performance of different classifiers, was the Statlog project, [36]. Two subsequent efforts are summarized in Refs. [9, 35]. In the former, 17 popular classifiers were

tested against 21 data sets. In the latter, 10 classifiers and 11 data sets were employed. The results verify what has already been said: different classifiers perform better for different sets. However, it is reported that boosted trees (Section 7.11), random forests, bagged decision trees, and support vector machines were ranked among the top ones for most of the data sets.

Neural Information Processing Systems Workshop (NIPS-2003) organized a classification competition based on five data sets. The results of the competition are summarized in Ref. [25]. The competition was focused on feature selection [38]. In a follow-up study, [29], more classifiers were added. Among the considered classifiers, a Bayesian-type neural network scheme (Chapter 18) scored at the top, albeit at significantly higher run time requirements. The other classifiers considered were random forests and boosting, where trees and neural networks were used as base classifiers (Section 7.10). Random forests also performed well, at much lower computational times compared to the Bayesian-type classifier.

Schemes for combining classifiers

A trend to improve performance is to combine different classifiers together and exploit their individual advantages. An observation that justifies such an approach is that during testing, there are patterns on which even the best classifier for a particular task fails to predict their true class. In contrast, the same patterns can be classified correctly by other classifiers, with an inferior overall performance. This shows that there may be some complementarity among different classifiers, and combination can lead to boosted performance compared to that obtained from the best (single) classifier. Recall that bagging, mentioned in Section 7.8, is a type of classifier combination.

The issue that arises now is to select a combination scheme. There are different schemes, and the results they provide can be different. Below, we summarize the more popular combination schemes.

- *Arithmetic averaging rule:* Assuming that we use L classifiers, where each one outputs a value of the posterior probability, $P_j(\omega_i|\mathbf{x})$, $i = 1, 2, \dots, M$, $j = 1, 2, \dots, L$, a decision concerning the class assignment is based on the following rule:

$$\text{Assign } \mathbf{x} \text{ to class } \omega_i = \arg \max_k \frac{1}{L} \sum_{j=1}^L P_j(\omega_k|\mathbf{x}), \quad k = 1, 2, \dots, M. \quad (7.66)$$

This rule is equivalent with computing the “final” posterior probability, $P(\omega_i|\mathbf{x})$, in order to minimize the average Kullback-Leibler distance (Problem 7.16),

$$D_{av} = \frac{1}{L} \sum_{j=1}^L D_j,$$

where

$$D_j = \sum_{i=1}^M P_j(\omega_i|\mathbf{x}) \ln \frac{P_j(\omega_i|\mathbf{x})}{P(\omega_i|\mathbf{x})}.$$

- *Geometric averaging rule:* This rule is the outcome of minimizing the alternative formulation of Kullback-Leibler distance (note that this distance is not symmetric); in other words,

$$D_j = \sum_{i=1}^M P(\omega_i|\mathbf{x}) \ln \frac{P(\omega_i|\mathbf{x})}{P_j(\omega_i|\mathbf{x})},$$

which results in (Problem 7.17),

$$\boxed{\text{Assign } \mathbf{x} \text{ to class } \omega_i = \arg \max_k \prod_{j=1}^L P_j(\omega_k | \mathbf{x}), \quad k = 1, 2, \dots, M.} \quad (7.67)$$

- *Stacking:* An alternative way is to use a weighted average of the outputs of the individual classifiers, where the combination weights are obtained optimally using the training data. Assume that the output of each individual classifier, $f_j(\mathbf{x})$, is of a soft-type; for example, an estimate of the posterior probability, as before. Then, the combined output is given by

$$f(\mathbf{x}) = \sum_{j=1}^L w_j f_j(\mathbf{x}), \quad (7.68)$$

where the weights are estimated via the following optimization task:

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) = \arg \min_{\mathbf{w}} \sum_{n=1}^N \mathcal{L}\left(y_n, \sum_{j=1}^L w_j f_j(\mathbf{x}_n)\right), \quad (7.69)$$

where, $\mathcal{L}(\cdot, \cdot)$ is a loss function; for example, the squared error one. However, adopting the previous optimization, based on the training data set, can lead to overfitting. According to stacking [56], a cross-validation rationale is adopted and instead of $f_j(\mathbf{x}_n)$, we employ the $f_j^{(-n)}(\mathbf{x}_n)$, where the latter is the output of the j th classifier trained on the data after *excluding* the pair (y_n, \mathbf{x}_n) . In other words, the weights are estimated by

$$\hat{\mathbf{w}} = \arg \min_{\mathbf{w}} \sum_{n=1}^N \mathcal{L}\left(y_n, \sum_{j=1}^L w_j f_j^{(-n)}(\mathbf{x}_n)\right). \quad (7.70)$$

Sometimes, the weights are constrained to be positive and add to one, giving rise to a constrained optimization task.

- *Majority voting rule:* The previous methods belong to the family of soft-type rules. A popular alternative is a hard-type rule, which is based on a voting scheme. One decides in favor of the class for which either there is a consensus or when at least l_c of the classifiers agree on the class label, where

$$l_c = \begin{cases} \frac{L}{2} + 1, & L \text{ is even}, \\ \frac{L+1}{2}, & L \text{ is odd}. \end{cases}$$

Otherwise, the decision is rejection (i.e., no decision is taken).

In addition to the sum, product, and majority voting, other combinations rules have also been suggested, which are inspired by the following inequalities [32]:

$$\prod_{j=1}^L P_j(\omega_i | \mathbf{x}) \leq \min_{j=1}^L P_j(\omega_i | \mathbf{x}) \leq \frac{1}{L} \sum_{j=1}^L P_j(\omega_i | \mathbf{x}) \leq \max_{j=1}^L P_j(\omega_i | \mathbf{x}), \quad (7.71)$$

and classification is achieved by using the max or min bounds instead of the sum and product. When outliers are present, one can instead use the *median* value:

$$\text{Assign } \mathbf{x} \text{ to class } \omega_i = \arg \max_k \text{median} \{P_j(\omega_k | \mathbf{x})\}, \quad k = 1, 2, \dots, M. \quad (7.72)$$

It turns out that a no free lunch theorem is also valid for the combination rules; there is not a universally optimal rule. It all depends on the data at hand; see [28].

There are a number of other issues related to the theory of combining classifiers; for example, how one chooses the classifiers to be combined. Should the classifiers be dependent or independent? Furthermore, combination does not necessarily imply improved performance; in some cases, one may experience a performance loss (higher error rate) compared to that of the best (single) classifier [27, 28]. Thus, combining has to take place with care. More on these issues can be found in Refs. [33, 52] and the references therein.

7.10 THE BOOSTING APPROACH

The origins of the *boosting* method for designing learning machines is traced back to the work of Valiant and Kearns [30, 54], who posed the question of whether a weak learning algorithm, meaning one that does slightly better than random guessing, can be *boosted* into a strong one with a good performance index. At the heart of such techniques lies the *base learner*, which is a weak one. Boosting consists of an iterative scheme, where at each step the base learner is optimally computed using a different training set; the set at the current iteration is generated either according to an iteratively obtained data distribution or, usually, via a weighting of the training samples, each time using a different set of weights. The latter are computed in order to take into account the achieved performance up to the current iteration step. The final learner is obtained via a *weighted average* of all the *hierarchically* designed base learners. Thus, boosting can also be considered a scheme for combining different learners.

It turns out that, given a sufficient number of iterations, one can significantly improve the (poor) performance of the weak learner. For example, in some cases in classification, the training error may tend to zero as the number of iterations increases. This is very interesting indeed. Training a weak classifier, by appropriate manipulation of the training data (as a matter of fact, the weighting mechanism identifies hard samples, the ones that keep failing, and places more emphasis on them) one can obtain a strong classifier. Of course, as we will discuss, the fact that the training error may tend to zero does *not* necessarily mean the test error goes to zero, too.

The AdaBoost algorithm

We now focus on the two-class classification task and assume that we are given a set of N training observations, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, with $y_n \in \{-1, 1\}$. Our goal is to design a binary classifier,

$$f(\mathbf{x}) = \text{sgn} \{F(\mathbf{x})\}, \quad (7.73)$$

where

$$F(\mathbf{x}) := \sum_{k=1}^K a_k \phi(\mathbf{x}; \boldsymbol{\theta}_k), \quad (7.74)$$

where $\phi(\mathbf{x}; \boldsymbol{\theta}_k) \in \{-1, 1\}$ is the base classifier at iteration k , defined in terms of a set of parameters, $\boldsymbol{\theta}_k$, $k = 1, 2, \dots, K$, to be estimated. The base classifier is selected to be a binary one. The set of unknown parameters is obtained in a *step wise* approach and in a *greedy* way; that is, at each iteration step, i , we only optimize with respect to a single pair, $(a_i, \boldsymbol{\theta}_i)$, by keeping the parameters, $a_k, \boldsymbol{\theta}_k$, $k = 1, 2, \dots, i - 1$, obtained from the previous steps, fixed. Note that ideally, one should optimize with respect to all the unknown parameters, $a_k, \boldsymbol{\theta}_k$, $k = 1, 2, \dots, K$, simultaneously; however, this would lead to a very computationally demanding optimization task. Greedy algorithms are very popular, due to their computational simplicity, and lead to a very good performance in a wide range of learning tasks. Greedy algorithms will also be discussed in the context of sparsity-aware learning in [Chapter 10](#).

Assume that we are currently at the i th iteration step; consider the partial sum of terms

$$F_i(\cdot) = \sum_{k=1}^i a_k \phi(\cdot; \boldsymbol{\theta}_k). \quad (7.75)$$

Then we can write the following recursion:

$$F_i(\cdot) = F_{i-1}(\cdot) + a_i \phi(\cdot; \boldsymbol{\theta}_i), \quad i = 1, 2, \dots, K, \quad (7.76)$$

starting from an initial condition. According to the greedy rationale, $F_{i-1}(\cdot)$ is assumed known and the goal is to optimize with respect to the set of parameters, $a_i, \boldsymbol{\theta}_i$. For optimization, a loss function has to be adopted. No doubt different options are available, giving different names to the derived algorithm. A popular loss function used for classification is the exponential loss, defined as

$\mathcal{L}(y, F(\mathbf{x})) = \exp(-yF(\mathbf{x}))$: Exponential Loss Function,

(7.77)

and it gives rise to the *Adaptive Boosting* (AdaBoost) algorithm. The exponential loss function is shown in [Figure 7.14](#), together with the 0-1 loss function. The former can be considered a (differentiable) upper bound of the (nondifferentiable) 0-1 loss function. Note that the exponential loss weighs misclassified ($yF(\mathbf{x}) < 0$) points more heavily compared to the correctly identified ones ($yF(\mathbf{x}) > 0$). Employing the

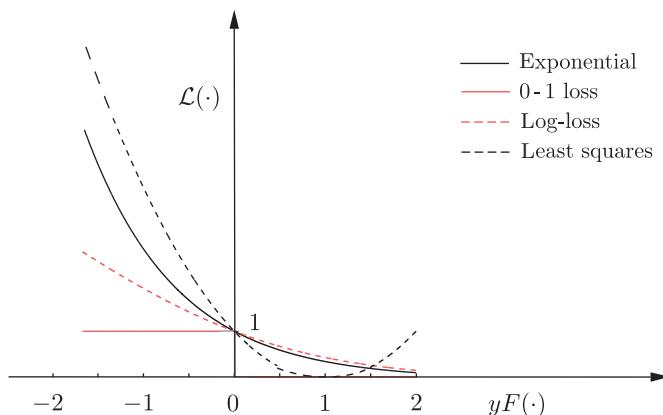


FIGURE 7.14

The 0-1, the exponential, the log-loss and the LS loss functions. They have all been normalized to cross the point $(0, 1)$. The horizontal axis for the squared error (LS) corresponds to $y - F(\mathbf{x})$.

exponential loss function, the set a_i, θ_i is obtained via the respective empirical cost function, in the following manner

$$(a_i, \theta_i) = \arg \min_{a, \theta} \sum_{n=1}^N \exp \left(-y_n (F_{i-1}(\mathbf{x}_n) + a\phi(\mathbf{x}_n; \theta)) \right). \quad (7.78)$$

This optimization is also performed in two steps. First, a is treated fixed and we optimize with respect to θ ,

$$\theta_i = \arg \min_{\theta} \sum_{n=1}^N w_n^{(i)} \exp (-y_n a \phi(\mathbf{x}_n; \theta)), \quad (7.79)$$

where

$$w_n^{(i)} := \exp (-y_n F_{i-1}(\mathbf{x}_n)), \quad n = 1, 2, \dots, N. \quad (7.80)$$

Observe that $w_n^{(i)}$ depends neither on a nor on $\phi(\mathbf{x}_n; \theta)$, hence it can be considered a weight associated with sample n . Moreover, its value depends entirely on the results obtained from the previous recursions.

We now turn our focus on the cost in (7.79). The optimization depends on the specific form of the base classifier. However, due to the exponential form of the loss, and the fact that the base classifier is a binary one, so that $\phi(\mathbf{x}; \theta) \in \{-1, 1\}$, optimizing (7.79) is readily seen to be equivalent with optimizing the following cost:

$$\theta_i = \arg \min_{\theta} P_i, \quad (7.81)$$

where

$$P_i := \sum_{n=1}^N w_n^{(i)} \chi_{(-\infty, 0]}(y_n \phi(\mathbf{x}_n; \theta)), \quad (7.82)$$

and $\chi_{[-\infty, 0]}(\cdot)$ is the 0-1 loss function.³ Note that P_i is the weighted empirical classification error. Obviously, when the misclassification error is minimized, the cost in (7.79) is also minimized, because the exponential loss weighs the misclassified points heavier. To guarantee that P_i remains in the $[0, 1]$ interval, the weights are normalized to unity by dividing by the respective sum; note that this does not affect the optimization process. In other words, θ_i can be computed in order to minimize the empirical misclassification error committed by the base classifier. For base classifiers of very simple structure, such a minimization is computationally feasible.

Having computed the optimal θ_i , the following are easily established from the respective definitions,

$$\sum_{y_n \phi(\mathbf{x}_n; \theta_i) < 0} w_n^{(i)} = P_i, \quad (7.83)$$

and

$$\sum_{y_n \phi(\mathbf{x}_n; \theta_i) > 0} w_n^{(i)} = 1 - P_i. \quad (7.84)$$

³ The characteristic function $\chi_A(x)$ is equal to one if $x \in A$ and zero otherwise.

Combining (7.83) and (7.84) with (7.78) and (7.80), it is readily shown that

$$a_i = \arg \min_a \left\{ \exp(-a)(1 - P_i) + \exp(a)P_i \right\}. \quad (7.85)$$

Taking the derivative with respect to a and equating to zero results in

$$a_i = \frac{1}{2} \ln \frac{1 - P_i}{P_i}. \quad (7.86)$$

Once a_i and θ_i have been estimated, the weights for the next iteration are readily given by

$$w_n^{(i+1)} = \frac{\exp(-y_n F_i(\mathbf{x}_n))}{Z_i} = \frac{w_n^{(i)} \exp(-y_n a_i \phi(\mathbf{x}_n; \theta_i))}{Z_i}, \quad (7.87)$$

where Z_i is the normalizing factor

$$Z_i := \sum_{n=1}^N w_n^{(i)} \exp(-y_n a_i \phi(\mathbf{x}_n; \theta_i)). \quad (7.88)$$

Looking at the way the weights are formed, one can grasp one of the major secrets underlying the AdaBoost algorithm: The weight associated with a training sample, \mathbf{x}_n , is increased (decreased) with respect to its value at the previous iteration, depending on whether the pattern has failed (succeeded) in being classified correctly. Moreover, the percentage of the decrease (increase) depends on the value of a_i , which controls the relative importance in the buildup of the final classifier. Hard samples, which keep failing over successive iterations, gain importance in their participation in the weighted empirical error value. For the case of the AdaBoost, it can be shown that the training error tends to zero exponentially fast ([Problem 7.18](#)). The scheme is summarized in [Algorithm 7.1](#).

Algorithm 7.1 (The AdaBoost algorithm).

- Initialize: $w_n^{(1)} = \frac{1}{N}$, $i = 1, 2, \dots, N$
- Initialize: $i = 1$
- Repeat
 - Compute the optimum θ_i in $\phi(\cdot; \theta_i)$ by minimizing P_i ; (7.81)
 - Compute the optimum P_i ; (7.82)
 - $a_i = \frac{1}{2} \ln \frac{1 - P_i}{P_i}$
 - $Z_i = 0$
 - **For** $n = 1$ to N **Do**
 - $w_n^{(i+1)} = w_n^{(i)} \exp(-y_n a_i \phi(\mathbf{x}_n; \theta_i))$
 - $Z_i = Z_i + w_n^{(i+1)}$
 - **End For**
 - **For** $n = 1$ to N **Do**
 - $w_n^{(i+1)} = w_n^{(i+1)} / Z_i$
 - **End For**
 - $K = i$
 - $i = i + 1$
- Until a termination criterion is met.
- $f(\cdot) = \text{sgn} \left(\sum_{k=1}^K a_k \phi(\cdot; \theta_k) \right)$

The AdaBoost was first derived in Ref. [20] in a different way. Our formulation follows that given in Ref. [21]. Yoav Freund and Robert Schapire received the prestigious Gödel award for this algorithm in 2003.

The log-loss function

In AdaBoost, the exponential loss function was employed. From a theoretical point of view, this can be justified by the following argument: Consider the mean value with respect to the binary label, y , of the exponential loss function,

$$\mathbb{E} [\exp(-yF(\mathbf{x}))] = P(y=1)\exp(-F(\mathbf{x}))+P(y=-1)\exp(F(\mathbf{x})). \quad (7.89)$$

Taking the derivative with respect to $F(\mathbf{x})$ and equating to zero, we readily obtain that the minimum of (7.89) occurs at

$$F_*(\mathbf{x}) = \arg \min_f \mathbb{E} [\exp(-yf)] = \frac{1}{2} \ln \frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})}. \quad (7.90)$$

The logarithm of the ratio on the right-hand side is known as the *log-odds* ratio. Hence, if one views the minimizing function in (7.78) as the empirical approximation of the mean value in (7.89), it fully justifies considering the sign of the function in (7.73) as the classification rule.

A major problem associated with the exponential loss function, as is readily seen in Figure (7.14), is that it weights heavily wrongly classified samples, depending on the value of the respective margin, defined as

$$m_x := |yF(\mathbf{x})|. \quad (7.91)$$

Note that the farther the point is from the decision surface ($F(\mathbf{x}) = 0$), the larger the value of $|F(\mathbf{x})|$. Thus, points that are located at the wrong side of the decision surface ($yF(\mathbf{x}) < 0$) and far away are weighted with (exponentially) large values, and their influence in the optimization process is large compared to the other points. Thus in the presence of outliers, the exponential loss is not the most appropriate one. As a matter of fact, in such environments, the performance of the AdaBoost can degrade dramatically.

An alternative loss function is the *log-loss* or *binomial deviance*, defined as

$$\boxed{\mathcal{L}(y, F(\mathbf{x})) := \ln(1 + \exp(-yF(\mathbf{x}))) : \text{ Log-loss Function,}} \quad (7.92)$$

which is also shown in Figure 7.14. Observe that its increase is almost linear for large negative values. Such a function leads to a more balanced influence of the loss among all the points. We will return to the issue of *robust loss functions*, that is, loss functions that are more immune to the presence of outliers, in Chapter 11. Note that the function that minimizes the mean of the log-loss, with respect to y , is the same as the one given in (7.90) (try it). However, if one employs the log-loss instead of the exponential, the optimization task gets more involved, and one has to resort to gradient descent or Newton-type schemes for optimization; see [22].

Remarks 7.7.

- For comparison reasons, in Figure 7.14, the LS loss is shown. The LS depends on the value $(y - F(\mathbf{x}))$, which is the equivalent of the margin defined above, $yF(\mathbf{x})$. Observe that, besides the relatively large influence that large values of error have, the error is also penalized for patterns whose label has been predicted correctly. This is one more justification that the LS criterion is not, in general, a good choice for classification.

- Multiclass generalizations of the Boosting scheme have been proposed in Refs. [19, 21]. In Ref. [16], regularized versions of the AdaBoost scheme have been derived in order to impose sparsity. Different regularization schemes are considered, including ℓ_1 , ℓ_2 , and ℓ_∞ . The end result is a family of coordinate-descent algorithms that integrate forward feature induction and back-pruning. In Ref. [47], a version is presented where a priori knowledge is brought into the scene. The so-called AdaBoost_p^{*} is introduced in Ref. [43], where the margin is explicitly taken into account.
- Note that the boosting rationale can be applied equally well to regression tasks involving respective loss functions, such as the LS. A robust alternative to the LS is the absolute error, instead of the square, loss function [22].
- The boosting technique has attracted a lot of attention among researchers in the field in order to justify its good performance in practice and its relative immunity to overfitting. While the training error may become zero, still this does not necessarily imply overfitting. A first explanation was attempted in terms of bounds, concerning the respective generalization performance. The derived bounds are independent of the number of iterations K , and they are expressed in terms of the margin Ref. [46]. However, these bounds tend to be very loose. Another explanation may lie in the fact that each time, optimization is carried out with only a single set of parameters. The interested reader may find the discussions following the papers [6, 8, 21] very enlightening on this issue.

Example 7.5. Consider a 20-dimensional two-class classification task. The data points from the first class (ω_1) stem from either of the two Gaussian distributions with means $\mu_{11} = [0, 0, \dots, 0]^T$, $\mu_{12} = [1, 1, \dots, 1]^T$, while the points of the second class (ω_2) stem from the Gaussian distribution

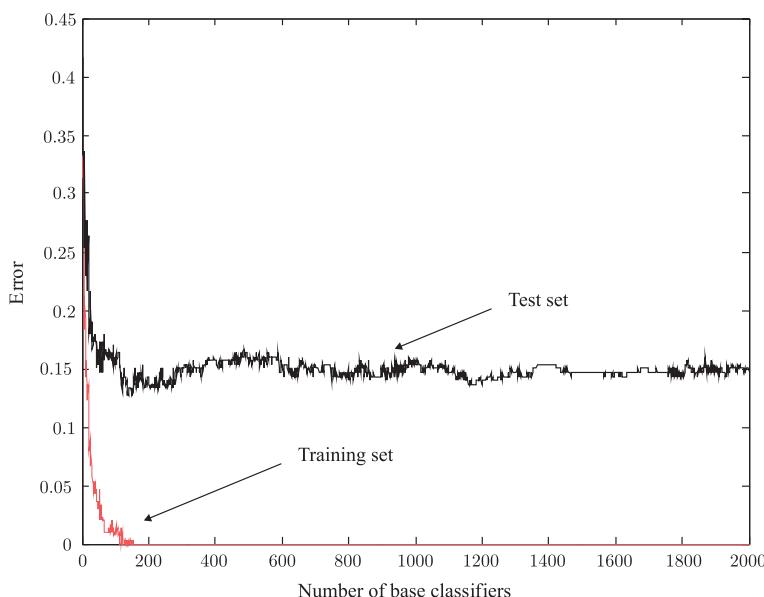


FIGURE 7.15

Training and test error rate curves as a function of the number of iterations, for the case of Example 7.5.

with mean $\mu_2 = [\overbrace{0, \dots, 0}^{10}, \overbrace{1, \dots, 1}^{10}]^\top$. The covariance matrices of all distributions are equal to the 20-dimensional identity matrix. Each one of the training and the test sets consists of 300 points, 200 from ω_1 (100 from each distribution) and 100 from ω_2 .

For the AdaBoost, the base classifier was selected to be a *stump*. This is a very naive type of tree, consisting of a single node, and classification of a feature vector x is achieved on the basis of the value of only one of its features, say, x_i . Thus, if $x_i < a$, where a is an appropriate threshold, x is assigned to class ω_1 . If $x_i > a$, it is assigned to class ω_2 . The decision about the choice of the specific feature, x_i , to be used in the classifier was randomly made. Such a classifier results in a training error rate slightly better than 0.5. The AdaBoost algorithm was run on the training data for 2000 iteration steps. Figure 7.15 verifies the fact that the training error rate converges to zero very fast. The test error rate keeps decreasing even after the training error rate becomes zero and then levels off at around 0.15.

7.11 BOOSTING TREES

In the discussion on experimental comparison of various methods in Section 7.9, it was stated the boosted trees are among the most powerful learning schemes for classification and data mining. Thus, it is worth spending some more time on this special type of boosting techniques.

Trees were introduced in Section 7.8. From the knowledge we have now acquired, it is not difficult to see that the output of a tree can be compactly written as

$$T(x; \Theta) = \sum_{j=1}^J \hat{y}_j \chi_{R_j}(x), \quad (7.93)$$

where J is the number of leaf nodes, R_j is the region associated with the j th leaf, after the space partition imposed by the tree, \hat{y}_j is the respective label associated with R_j (output/prediction value for regression) and χ is our familiar characteristic function. The set of parameters, Θ , consists of (\hat{y}_j, R_j) , $j = 1, 2, \dots, J$, which are estimated during training. These can be obtained by selecting an appropriate cost function. Also, suboptimal techniques are usually employed, in order to build up a tree, as the ones discussed in Section 7.8.

In a boosted tree model, the base classifier comprises a tree. For example, the stump used in Example 7.5 is a very special case of a boosted tree. In practice, one can employ trees of larger size. Of course, the size must not be very large, in order to be closer to a weak classifier. In practice, values of J between three and eight are advisable.

The boosted tree model can be written as

$$F(x) = \sum_{k=1}^K T(x; \Theta_k), \quad (7.94)$$

where

$$T(x; \Theta_k) = \sum_{j=1}^J \hat{y}_{kj} \chi_{R_{kj}}(x).$$

Equation (7.94) is basically the same as (7.74), with the a 's being equal to one. We have assumed the size of all the trees to be the same, although this may not be necessarily the case. Adopting a loss

function, \mathcal{L} , and the greedy rationale, used for the more general boosting approach, we arrive at the following recursive scheme of optimization:

$$\Theta_i = \arg \min_{\Theta} \sum_{n=1}^N \mathcal{L}(y_n, F_{i-1}(\mathbf{x}_n) + T(\mathbf{x}_n; \Theta)). \quad (7.95)$$

Optimization with respect to Θ takes place into two steps: one with respect to \hat{y}_{ij} , $j = 1, 2, \dots, J$, given R_{ij} , and then one with respect to the regions R_{ij} . The latter is a difficult task and only simplifies in very special cases. In practice, a number of approximations can be employed. Note that in the case of the exponential loss and the two-class classification task, the above is directly linked to the AdaBoost scheme.

For more general cases, numeric optimization schemes are mobilized; see [22]. The same rationale applies for regression trees, where now loss functions for regression, such as LS or the absolute error value, are used. Such schemes are also known as *multiple additive regression trees* (MARTs). A related implementation code for boosted trees is freely available in the R gbm package, [44].

There are two critical factors concerning boosted trees. One is the size of the trees, J , and the other is the choice of K . Concerning the size of the trees, usually one tries different sizes, $4 \leq J \leq 8$, and selects the best one. Concerning the number of iterations, for large values, the training error may get close to zero, but the test error can increase due to overfitting. Thus, one has to stop early enough, usually by monitoring the performance.

Another way to cope with overfitting is to employ *shrinkage* methods, which tend to be equivalent to regularization. For example, in the stage-wise expansion of $F_i(\mathbf{x})$ used in the optimization step (7.95), one can instead adopt the following:

$$F_i(\cdot) = F_{i-1}(\cdot) + \nu T(\cdot; \Theta_i).$$

The parameter ν takes small values and it can be considered as controlling the learning rate of the boosting procedure. Values smaller than $\nu < 0.1$ are advised. However, the smaller the value of ν , the larger the value K should be to guarantee good performance. For more on MARTs, the interested reader can peruse [26].

7.12 A CASE STUDY: PROTEIN FOLDING PREDICTION

One of the most challenging modalities in bioinformatics is that of genetic data, that is, DNA sequences that can be stored and used either as raw input for models (e.g., sequence alignment, statistical analysis) or as the basis for the discovery of higher-level intrinsic attributes (e.g., the 3-D structure of the protein they produce). Since the discovery of the DNA structure in 1953 by James Watson and Francis Crick, but especially after the completion of the Human Genome Project in 2003, the basic building blocks of all life can be traced down to simple chemical components. In terms of data storage and analysis, these components are no more than a sequence of symbols in a biometric signal: the DNA strand.

The most basic building element in a DNA sequence is the set of four *nucleotides* or *nucleobases*: adenine (A), cytosine (C), guanine (G), and thymine (T). These four bases are complementary in pairs, such that stable chemical bonds can be formed between guanine with cytosine (G-T) and adenine with thymine (A-T). These bonds produce the celebrated double helix in the DNA macromolecule and provide a redundant encoding mechanism for the genetic information. Each nonoverlapping triplet of such subsequent nucleotides in a DNA sequence is called a *codon* and corresponds to one of the

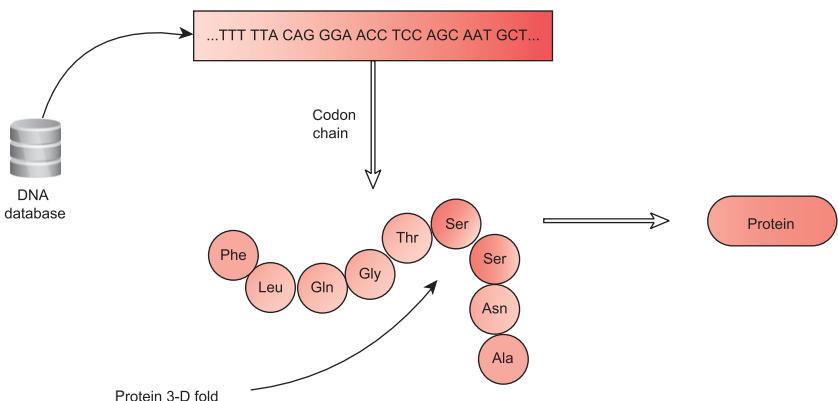
Table 7.1 The genetic code for the 20 standard proteinogenic amino acids. They are the building blocks of protein-coding strands of every DNA and they are formed as triples of the four nucleobases (T, C, A, G). Each triplet ('codon') is read left-up-right; for example, Alanine ("Ala") is encoded by the triplets "GCx" (x=any). The "(st)" codon signals the end-of-translation in a coding sequence [1].

	T	C	A	G	
T	Phe	Ser	Tyr	Cys	T
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	(st)	(st)	A
	Leu	Ser	(st)	Trp	G
C	Leu	Pro	His	Arg	T
	Leu	Pro	His	Arg	C
	Leu	Pro	Gln	Arg	A
	Leu	Pro	Gln	Arg	G
A	Ile	Thr	Asn	Ser	T
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	T
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

20 basic or *standard proteinogenic* amino-acids that are coded directly in DNA. Amino acids are the building blocks of proteins, the most important elements in the functionality of living cells. In fact, four different symbols combined in triplets produce 64 possible combinations; however, each of the 20 standard amino acids is encoded redundantly with a variable number of alternatives, and there are also three "stop" codons to signify the end-of-translation in a sequence; **Table 7.1**. There is also a "start" codon to signify the beginning; this is the "ATG" triplet, which coincides with the "Met." If this is met for a first time, signifies the beginning of a sequence, and if it is in the middle it corresponds to an amino acid.

Each possible protein is encoded by a variable number of amino acids, ranging typically from 80 to 170 codons; this corresponds to sequences of roughly 240-510 nucleotides, which essentially form the low-level encoding of the most useful information content of DNA (**Figure 7.16**).

The human DNA sequence comprises approximately 3.2×10^6 base pairs of nucleotides [1]. Regions in the sequence that have been identified as protein-encoding regions are limited to roughly 1-2% of the total length of the DNA sequence. Such regions are also known as *genes*. Early estimates of the number of human genes was around 5,0000-10,0000, however the analysis of the human genome in

**FIGURE 7.16**

The DNA genetic code translation process: from the sequence of nucleobases into a valid codon sequence of standard amino acids that can synthesize a specific protein.

recent years suggests that this number is limited to about 20,500 genes. The rest of the DNA sequence is known as noncoding. Some of the remaining (over 98% in humans) areas of the DNA sequence serve different functions, from sequence alignment to regulatory mechanisms (approximately 8.2%) and are known as the *functional* part; the rest of the DNA sequence (roughly 91.8%) is known as “junk” DNA; see [42]. Sometimes an organism’s DNA encodes different proteins with overlapping reading frames, producing different (valid) results for various start/end codon offsets. This problem is not much different than decoding a serial bit stream into bytes in an asynchronous mode of operation. In short, the DNA strand is an encoding scheme that employs (a) two complementary streams (double helix) and (b) redundancy in symbols and overlapping reading frames, with (c) additional information outside the actual “useful” coding blocks (“exons”).

Protein folding prediction as a classification task

Each standard amino acid sequence is an encoding scheme for a specific protein. The corresponding molecular 3-D structure of each protein is one of the most important aspects of proteomics. One of the main structural properties of a 3-D molecular structure is known as *fold*, and it is associated with the way that a protein is deployed in space. Each fold affects various chemical properties and inherent functionality of the corresponding gene in a DNA strand, which are of paramount importance to biologists. The prediction of the protein folding from the amino acid sequence has been one of the most challenging tasks since the creation of detailed genome databases, including the human genome project. Biologists have identified a number of possible folds that a protein can acquire, and each one of them is associated with certain properties. It is thus very important, once an unknown protein is identified (equivalently, once an encoding protein sequence is detected), to be able to find the folds associated with the 3-D structural form that the corresponding molecule acquires in space. This is achieved by trying to classify the new finding into one of the previously known classes, where each class is associated with a specific fold and its respective properties. This now becomes a typical classification task.

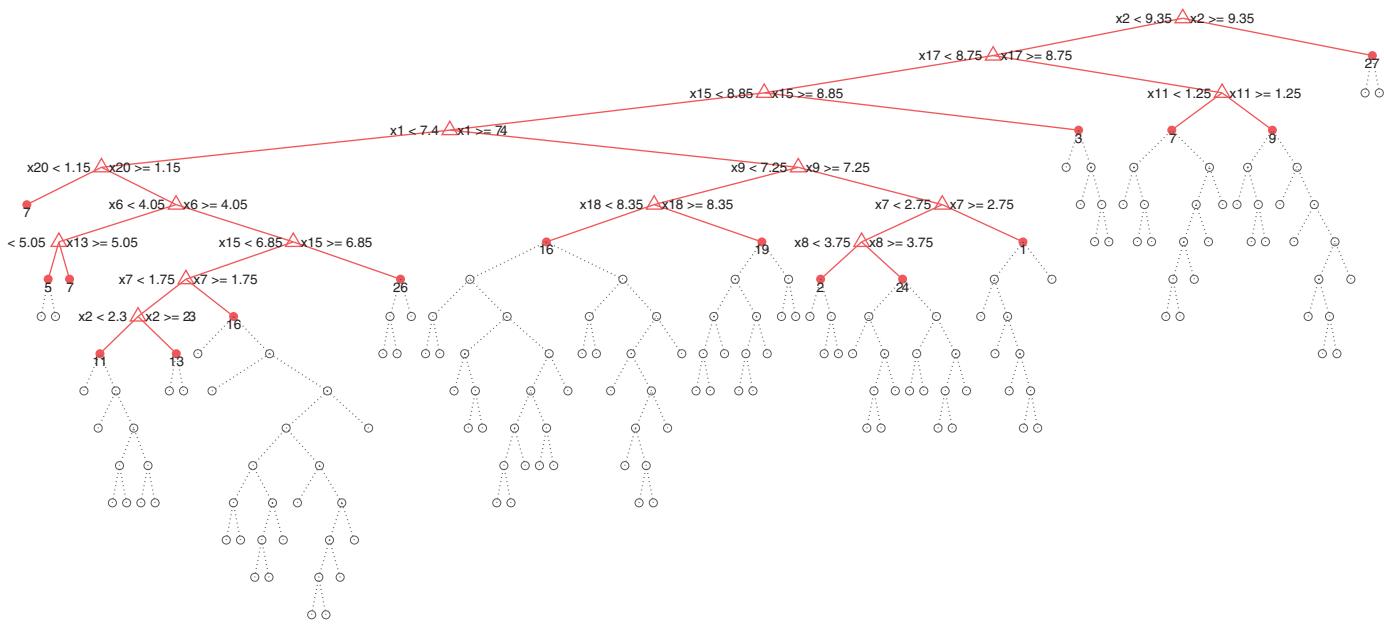


FIGURE 7.17

CART decision tree for the benchmark data set – Node details and thresholds for the first few levels.

Classification of folding prediction via decision trees

The data set used is available at UCSD-MKL repository [37] for protein fold prediction, based on a subset of the PDB-40D SCOP collection [41]. This data set contains two splits: a training subset with 311 samples and a testing subset with 383 samples. In the original data set, for every sample there are 12 different sets of feature vectors, each one giving rise to a different feature space. In this example, the 20-value “composition” vector was used: Once a DNA encoding sequence has been identified, the corresponding feature vector is constructed according to the relative frequencies (%) of the appearance of each one of the 20 standard amino acids in the sequence, which corresponds to the sample [39, 40]. Indeed, the composition vector has been established as a valid metric correlated with the 3-D properties of the corresponding protein molecule [2, 12]. Thus, the dimensionality of the feature space is equal to 20. The number of classes we are going to consider is 27, which is a selected subset comprising the most characteristic protein folds in the original PDB-40B database (from a total of more than 600 classes-folds).

The classifier model used in this example was a typical classification and regression tree - CART (Section 7.8) and it was carried out utilizing the Statistics toolbox of MATLAB (ver 8.0+).

Figure 7.17 illustrates the first few levels of the trained CART decision tree. Internal nodes include the (binary) decision threshold associated with it; for example, $x_1 < 7.4$ tests whether the relative frequency of the alphabetically first amino acid (Ala) in the considered sample protein sequence is lower than 7.4%. The overall classification accuracy in the testing subset is 31.85%, which is close to the performance of other more advanced classifiers, including multi-layered perceptrons (Chapter 18) and support vector machines (Chapter 11) applied on the same task, i.e., using only the composition vector as input; see [15].

Note that even though the classification accuracy is low for a fully automated procedure, these predictive models can be used as valuable tools for limiting the search scope and prioritizing the relevant experiments for the characterization of new proteins [34].

Our focus was to present an application area of immense interest, while the approach we followed was rather on the pedagogic side, by employing simple features and a single classifier. More state-of-the-art approaches employ much more descriptive statistics as feature vectors, instead of the composition vector as described above. For example, the 20×20 correlation matrix of subsequent amino acids can be calculated from the coding sequence and used as input in classification schemes [10, 34]. A current trend is to combine classifiers as well as feature spaces with additional information content (i.e., not only the coding sequence itself), and classification accuracy rates up to 70% have been reported [13, 24, 31].

PROBLEMS

- 7.1** Show that the Bayesian classifier is optimal, in the sense that it minimizes the probability of error.

Hint. Consider a classification task of M classes and start with the probability of correct label prediction, $P(C)$. Then the probability of error will be $P(e) = 1 - P(C)$.

- 7.2** Show that if the data follow the Gaussian distribution in an M class task, with equal covariance matrices in all classes, the regions formed by the Bayesian classifier are convex.
- 7.3** Derive the form of the Bayesian classifier for the case of two equiprobable classes, when the data follow the Gaussian distribution of the same covariance matrix. Furthermore, derive the equation that describes the LS linear classifier. Compare and comment on the results.

- 7.4** Show that the ML estimate of the covariance matrix of a Gaussian distribution, based on N i.i.d. observations, \mathbf{x}_n , $n = 1, 2, \dots, N$, is given by

$$\hat{\Sigma}_{ML} = \frac{1}{N} \sum_{n=1}^N (\mathbf{x}_n - \hat{\mu}_{ML})(\mathbf{x}_n - \hat{\mu}_{ML})^T,$$

where

$$\hat{\mu}_{ML} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n.$$

- 7.5** Prove that the covariance estimate

$$\hat{\Sigma} = \frac{1}{N-1} \sum_{k=1}^N (\mathbf{x}_k - \hat{\mu})(\mathbf{x}_k - \hat{\mu})^T$$

defines an unbiased estimator, where

$$\hat{\mu} = \frac{1}{N} \sum_{k=1}^N \mathbf{x}_k.$$

- 7.6** Show that the derivative of the logistic link function is given by

$$\frac{d\sigma(t)}{dt} = \sigma(t)(1 - \sigma(t)).$$

- 7.7** Derive the gradient of the negative log-likelihood function associated with the two-class logistic regression.
7.8 Derive the Hessian matrix of the negative log-likelihood function associated with the two-class logistic regression.
7.9 Show that the Hessian matrix of the negative log-likelihood function of the two-class logistic regression is a positive definite matrix.

- 7.10** Show that if

$$\phi_m = \frac{\exp(t_m)}{\sum_{j=1}^M \exp(t_j)},$$

the derivative with respect to t_j , $j = 1, 2, \dots, M$, is given by

$$\frac{\partial \phi_m}{\partial t_j} = \phi_m(\delta_{mj} - \phi_j).$$

- 7.11** Derive the gradient of the negative log-likelihood for the multiclass logistic regression case.
7.12 Derive the j, k block element of the Hessian matrix of the negative log-likelihood function for the multiclass logistic regression.
7.13 Consider the Rayleigh ratio,

$$R = \frac{\boldsymbol{\theta}^T A \boldsymbol{\theta}}{\|\boldsymbol{\theta}\|^2},$$

where A is a symmetric positive definite matrix. Show that R is maximized, with respect to $\boldsymbol{\theta}$, if $\boldsymbol{\theta}$ is the eigenvector corresponding to the maximum eigenvalue of A .

7.14 Consider the generalized Rayleigh quotient,

$$R_g = \frac{\boldsymbol{\theta}^T B \boldsymbol{\theta}}{\boldsymbol{\theta}^T A \boldsymbol{\theta}}.$$

where A and B are a symmetric positive definite matrices. Show that R_g is maximized with respect to $\boldsymbol{\theta}$, if $\boldsymbol{\theta}$ is the eigenvector that corresponds to the maximum eigenvalue of $A^{-1}B$, assuming that the inversion is possible.

- 7.15** Show that the between-class scatter matrix Σ_b for an M class problem is of rank $M - 1$.
- 7.16** Derive the arithmetic rule for combination, by minimizing the average KL distance.
- 7.17** Derive the product rule via the minimization of the Kullback-Leibler distance, as pointed out in the text.
- 7.18** Show that the error rate on the training set of the final classifier, obtained by boosting, tends to zero exponentially fast.

MATLAB Exercises

- 7.19** Consider a two-dimensional class problem that involves two classes, ω_1 and ω_2 , which are modeled by Gaussian distributions with means $\boldsymbol{\mu}_1 = [0, 0]^T$ and $\boldsymbol{\mu}_2 = [2, 2]^T$, respectively, and common covariance matrix $\Sigma = \begin{bmatrix} 1 & 0.25 \\ 0.25 & 1 \end{bmatrix}$.
 - (i) Form and plot a data set \mathcal{X} consisting from 500 points from ω_1 and another 500 points from ω_2 .
 - (ii) Assign each one of the points of \mathcal{X} to either ω_1 or ω_2 , according to the Bayes decision rule, and plot the points with different colors, depending on the class they are assigned to. Plot the corresponding classifier.
 - (iii) Based on (ii), estimate the error probability.
 - (iv) Let $L = \begin{bmatrix} 0 & 1 \\ 0.005 & 0 \end{bmatrix}$ be a loss matrix. Assign each one of the points of \mathcal{X} to either ω_1 or ω_2 , according to the average risk minimization rule (Eq. (7.9)), and plot the points with different colors, depending on the class they are assigned to.
 - (v) Based on (iv), estimate the average risk for the above loss matrix.
 - (vi) Comment on the results obtained by (ii)-(iii) and (iv)-(v) scenarios.
- 7.20** Consider a two-dimensional class problem that involves two classes, ω_1 and ω_2 , which are modeled by Gaussian distributions with means $\boldsymbol{\mu}_1 = [0, 2]^T$ and $\boldsymbol{\mu}_2 = [0, 0]^T$ and covariance matrices $\Sigma_1 = \begin{bmatrix} 4 & 1.8 \\ 1.8 & 1 \end{bmatrix}$ and $\Sigma_2 = \begin{bmatrix} 4 & 1.2 \\ 1.2 & 1 \end{bmatrix}$, respectively.
 - (i) Form and plot a data set \mathcal{X} consisting from 5000 points from ω_1 and another 500 points from ω_2 .
 - (ii) Assign each one of the points of \mathcal{X} to either ω_1 or ω_2 , according to the Bayes decision rule, and plot the points with different colors, according to the class they are assigned to.
 - (iii) Compute the error classification probability.
 - (iv) Assign each one of the points of \mathcal{X} to either ω_1 or ω_2 , according to the naive Bayes decision rule, and plot the points with different colors, according to the class they are assigned to.
 - (v) Compute the error classification probability, for the naive Bayes classifier.

(vii) Repeat steps (i)-(v) for the case where $\Sigma_1 = \Sigma_2 = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$.

(viii) Comment on the results.

Hint. Use the fact that the marginal distributions of $P(\omega_1|\mathbf{x})$, $P(\omega_1|x_1)$, and $P(\omega_1|x_2)$ are also Gaussians with means 0 and 2 and variances 4 and 1, respectively. Similarly, the marginal distributions of $P(\omega_2|\mathbf{x})$, $P(\omega_2|x_1)$, and $P(\omega_2|x_2)$ are also Gaussians with means 0 and 0 and variances 4 and 1, respectively.

- 7.21** Consider a two-class, two-dimensional classification problem, where the first class (ω_1) is modeled by a Gaussian distribution with mean $\boldsymbol{\mu}_1 = [0, 2]^T$ and covariance matrix

$\Sigma_1 = \begin{bmatrix} 4 & 1.8 \\ 1.8 & 1 \end{bmatrix}$, while the second class (ω_2) is modeled by a Gaussian distribution with mean $\boldsymbol{\mu}_2 = [0, 0]^T$ and covariance matrix $\Sigma_2 = \begin{bmatrix} 4 & 1.8 \\ 1.8 & 1 \end{bmatrix}$.

- (i) Generate and plot a training set \mathcal{X} and a test set \mathcal{X}_{test} , each one consisting of 1500 points from each distribution.
- (ii) Classify the data vectors of \mathcal{X}_{test} using the Bayesian classification rule.
- (iii) Perform logistic regression and use the data set \mathcal{X} to estimate the involved parameter vector $\boldsymbol{\theta}$. Evaluate the classification error of the resulting classifier based on \mathcal{X}_{test} .
- (iv) Comment on the results obtained by (ii) and (iii).
- (v) Repeat the previous steps (i)-(iv), for the case where $\Sigma_2 = \begin{bmatrix} 4 & -1.8 \\ -1.8 & 1 \end{bmatrix}$ and compare the obtained results with those produced by the previous setting. Draw your conclusions.

Hint. For the estimation of $\boldsymbol{\theta}$ in (iii), perform steepest descent (Eq. (7.37)) and set the learning parameter μ_i equal to 0.001.

- 7.22** Consider a two-dimensional classification problem involving three classes ω_1 , ω_2 , and ω_3 . The data vectors from ω_1 stem from either of the two Gaussian with means $\boldsymbol{\mu}_{11} = [0, 3]^T$,

$\boldsymbol{\mu}_{12} = [11, -2]^T$ and covariance matrices $\Sigma_{11} = \begin{bmatrix} 0.2 & 0 \\ 0 & 2 \end{bmatrix}$ and $\Sigma_{12} = \begin{bmatrix} 3 & 0 \\ 0 & 0.5 \end{bmatrix}$, respectively.

Similarly, the data vectors from ω_2 stem from either of the two Gaussians distributions with

means $\boldsymbol{\mu}_{21} = [3, -2]^T$, $\boldsymbol{\mu}_{22} = [7.5, 4]^T$ and covariance matrix $\Sigma_{21} = \begin{bmatrix} 5 & 0 \\ 0 & 0.5 \end{bmatrix}$ and

$\Sigma_{22} = \begin{bmatrix} 7 & 0 \\ 0 & 0.5 \end{bmatrix}$, respectively. Finally, ω_3 is modeled by a single Gaussian distribution with

mean $\boldsymbol{\mu}_3 = [7, 2]^T$ and covariance matrix $\Sigma_3 = \begin{bmatrix} 8 & 0 \\ 0 & 0.5 \end{bmatrix}$.

- (i) Generate and plot a training data set \mathcal{X} consisting of 1000 data points from ω_1 (500 from each distribution), 1000 data points from ω_2 (again 500 from each distribution), and 500 points from ω_3 (use 0 as the seed for the initialization of the Gaussian random number generator). In a similar manner, generate a test data set \mathcal{X}_{test} (use 100 as the seed for the initialization of the Gaussian random number generator).
- (ii) Generate and view a decision tree based on using \mathcal{X} as the training set.
- (iii) Compute the classification error on both the training and the test sets. Comment briefly on the results.

- (iv) Prune the produced tree at levels 0 (no actual pruning), $1, \dots, 11$ (In MATLAB, trees are pruned based on an optimal pruning scheme that first prunes branches giving less improvement in error cost). For each pruned tree compute the classification error based on the test set.
- (v) Plot the classification error versus the pruned levels and locate the pruned level that gives the minimum test classification error. What conclusions can be drawn by the inspection of this plot?
- (vi) View the original decision tree as well as the best pruned one.

Hint. The MATLAB functions that generate a decision tree (DT), display a DT, prune a DT, evaluate the performance of a DT on a given data set, are *classregtree*, *view*, *prune*, and *eval*, respectively.

- 7.23** Consider a two-class, two-dimensional classification problem where the classes are modeled as the first two classes in the previous exercise.

- (i) Generate and plot a training set \mathcal{X} , consisting of 100 data points from each distribution of each class (that is, \mathcal{X} contains 400 points in total, 200 points from each class). In a similar manner, generate a test set.
- (ii) Use the training set to built a boosting classifier, utilizing as weak classifier a single-node decision tree. Perform 12,000 iterations.
- (iii) Plot the training and the test error versus the number of iterations and comment on the results.

Hint. – For (i) use *randn('seed', 0)* and *randn('seed', 100)* to initialize the random number generator for the training and the test set, respectively.

- For (ii) use

$$\text{ens} = \text{fitensemble}(\mathbf{X}', \mathbf{y}', \text{'AdaBoostM1'}, \text{'no_of_base_classifiers', } \text{'Tree'})$$
where \mathbf{X}' has in its rows the data vectors, \mathbf{y} is an ordinal vector containing the class where each row vector of \mathbf{X}' belongs, *AdaBoostM1* is the boosting method used, *no_of_base_classifiers* is the number of base classifiers that will be used, and *Tree* denotes the weak classifier.
- For (iii) use $L = \text{loss}(\text{ens}, \mathbf{X}', \mathbf{y}', \text{'mode', } \text{'cumulative'})$, which for a given boosting classifier *ens*, returns the vector L of errors performed on \mathbf{X}' , such that $L(i)$ being the error committed when only the first i weak classifiers are taken into account.

- 7.24** Consider the classification task for protein folding prediction as described in [Section 7.12](#).

Using the same subset of the PDB-40D SCOP collection [41] from the UCSD-MKL repository [37], write a MATLAB program to reproduce these results.

- (i) Read the training subset \mathcal{X} , consisting of 311 data points, and the testing subset \mathcal{Y} , consisting of 383 data points. Each data point represents a sample “fold” described by the 20-value “composition” feature vector of amino acids and assigned to one of the 27 classes.
- (ii) Using the Statistics toolbox of MATLAB, create and train a standard CART in classification mode for this task. Evaluate the classifier using the testing subset and report the overall accuracy rate.

Hint. – For (ii) use the ‘ClassificationTree’ object from the Statistics toolbox in MATLAB (v8.1+).

- The tree object provides a “fit” method for training and a “predict” method for testing the constructed model.

REFERENCES

- [1] J.M. Berg, J.L. Tymoczko, L. Stryer, Biochemistry, fifth ed., Freedman, New York, 2002.
- [2] H. Bohr et al., A novel approach to prediction of the 3-dimensional structures of protein backbones by neural networks, FEBS Lett. 261 (1990) 43-46.
- [3] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, 2004.
- [4] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth, 1984.
- [5] L. Breiman, Bagging predictors, Machine Learn. 24 (1996) 123-140.
- [6] L. Breiman, Arcing classifiers, Ann. Stat. 26(3) (1998) 801-849.
- [7] L. Breiman, Random forests, Machine Learn. 45 (2001) 5-32.
- [8] P. Bühlman, T. Hothorn, Boosting algorithms: regularization, prediction and model fitting (with discussion), Stat. Sci. 22(4) (2007) 477-505.
- [9] A. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in International Conference on Machine Learning, 2006.
- [10] Y. Chen, F. Ding, H. Nie, et al., Protein folding: Then and now, Arch. Biochem. Biophys. 469(1) (2008) 4-19.
- [11] H. Chipman, E. George, R. McCulloch, BART: Bayesian additive regression trees, Ann. Appl. Stat. 4(1) (2010) 266-298.
- [12] F. Crick, The recent excitement about neural networks, Nature 337 (1989) 129-132.
- [13] A. Dehzangi, K. Paliwal, A. Sharma, O. Dehzangi, A. Sattar, A combination of feature extraction methods with an ensemble of different classifiers for protein structural class prediction problem, IEEE/ACM Trans. Comput. Biol. Bioinform. 10(3) (2013) 564-575.
- [14] L. Devroye, L. Gyorfi, G.A. Lugosi, A Probabilistic Theory of Pattern Recognition, Springer Verlag city, 1996.
- [15] C. Ding, I. Dubchak, Multi-class protein fold recognition using support vector machines and neural networks, Bioinformatics 17 (4) (2001) 349-358.
- [16] J. Duchi, Y. Singer, Boosting with structural sparsity, in: Proceedings of the 26th International Conference on Machine Learning, Montreal, Canada, 2009.
- [17] R. Duda, P. Hart, D. Stork, Pattern Classification, second ed., Wiley, New York, 2000.
- [18] B. Efron, The efficiency of logistic regression compared to normal discriminant analysis, J. Amer. Stat. Assoc. 70 (1975) 892-898.
- [19] G. Eibl, K.P. Pfeifer, Multiclass boosting for weak classifiers, J. Machine Learn. Res. 6 (2006) 189-210.
- [20] Y. Freund, R.E. Schapire, A decision theoretic generalization of on-line learning and an applications to boosting, J. Comput. Syst. Sci. 55(1) (1997) 119-139.
- [21] J. Friedman, T. Hastie, R. Tibshirani, Additive logistic regression: a statistical view of boosting, Ann. Stat. 28(2) (2000) 337-407.
- [22] J. Freidman, Greedy function approxiamtion: a gradient boosting machine, Ann. Stat. 29(5) (2001) 1189-1232.
- [23] K. Fukunaga, Introduction to Statistical Pattern Recognition, second ed., Academic Press, 1990.

- [24] P. Ghanty, N.R. Pal, Prediction of protein folds: extraction of new features, dimensionality reduction, and fusion of heterogeneous classifiers, *IEEE Trans. NanoBiosci.* 8(1) (2009) 100-110.
- [25] I. Guyon, S. Gunn, M. Nikravesh, L. Zadeh (Eds.), *Feature Extraction, Foundations and Applications*, Springer Verlag, New York, 2006.
- [26] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, second ed., Springer Verlag, 2009.
- [27] R. Hu, R.I. Damper, A no panacea theorem for classifier combination, *Pattern Recogn.* 41 (2008) 2665-2673.
- [28] A.K. Jain, P.W. Duin, J. Mao, Statistical pattern recognition: a review, *IEEE Trans. Pattern Anal. Machine Intell.* 22(1) (2000) 4-37.
- [29] N. Johnson, A study of the NIPS feature selection challenge, Technical Report, Stanford University, <http://statweb.stanford.edu/~tibs/ElemStatLearn/comp.pdf>, 2009.
- [30] M. Kearns, L.G. Valiant, Cryptographic limitations of learning Boolean formulae and finite automata, *J. ACM* 41(1) (1994) 67-95.
- [31] K.-L. Lin, C.-Y. Lin, C.-D. Huang, et al., Feature selection and combination criteria for improving accuracy in protein structure prediction, *IEEE Trans. NanoBiosci.* 6(2) (2007) 186-196.
- [32] J. Kittler, M. Hatef, R. Duin, J. Matas, On combining classifiers, *IEEE Trans. Pattern Anal. Machine Intell.* 20(3) (1998) 228-234.
- [33] I.L. Kuncheva, *Pattern Classifiers: Methods and Algorithms*, John Wiley, 2004.
- [34] L. Hunter (Ed.), *Artificial Intelligence in Molecular Biology*, AAAI/MIT Press, CA, 1993.
- [35] D. Meyer, F. Leisch, K. Hornik, The support vector machine under test, *Neurocomputing* 55 (2003) 169-186.
- [36] D. Michie, D.J. Spiegelhalter, C.C. Taylor (Eds.), *Machine Learning, Neural, and Statistical Classification*, Ellis Horwood, London, 1994.
- [37] <https://mldata.org/repository/data/viewslug/protein-fold-prediction-uucsdk-mkl>.
- [38] R. Neal, J. Zhang, High dimensional classification with Bayesian neural networks and Dirichlet diffusion trees, in: I. Guyon, S. Gunn, M. Nikravesh, L. Zadeh (Eds.), *Feature Extraction, Foundations and Applications*, Springer Verlag, New York, 2006, pp. 265-296.
- [39] K. Nishikawa, T. Ooi, Correlation of the amino acid composition of a protein to its structural and biological characteristics, *J. Biochem.* 91 (1982) 1821-1824.
- [40] K. Nishikawa, Y. Kubota, T. Ooi, Classification of proteins into groups based on amino acid composition and other characters, *J. Biochem.* 94 (1983) 981-995.
- [41] <http://scop.berkeley.edu>.
- [42] C.M. Rands, S. Meader, C.P. Ponting, G. Lunter, 8.2% of the human genome is constrained: Variation in rates of turnover across functional element classes in the human lineage, *PLOS Genet.* 10(7) (2014) 1-12.
- [43] G. Ratsch, M.K. Warmuth, Efficient margin maximizing with boosting, *J. Machine Learn. Res.* 6 (2005) 2131-2152.
- [44] G. Ridgeway, The state of boosting, *Comput. Sci. Stat.* 31 (1999) 172-181.
- [45] B.D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996.
- [46] R.E. Schapire, V. Freund, P. Bartlett, W.S. Lee, Boosting the margin: a new explanation for the effectiveness of voting methods, *Ann. Stat.* 26(5) (1998) 1651-1686.
- [47] R.E. Schapire, M. Rochery, M. Rahim, N. Gupta, Boosting with prior knowledge for call classification, *IEEE Trans. Speech Audio Process.* 13(2) (2005) 174-181.
- [48] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, A.A. Blake, Real-time human pose recognition in parts from single depth images, in: *Proceedings of the Conference on Computer Vision and Pattern Recognition, CVPR*, 2011.
- [49] R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, San Mateo, 1993.
- [50] D.B. Rubin, Iterative reweighted least squares, in: *Encyclopedia of Statistical Sciences*, vol. 4, John Wiley, city 1983, pp. 272-275.
- [51] S. Russell, P. Norvig, *Artificial Intelligence: A Modern Approach*, third ed., Pearson, 2010.

- [52] S. Theodoridis, K. Koutroumbas, Pattern Recognition, fourth ed., Academic Press, 2009.
- [53] S. Theodoridis, A. Pikrakis, K. Koutroumbas, D. Cavouras, An Introduction to Pattern Recognition: A MATLAB Approach, Academic Press, 2010.
- [54] L.G. Valiant, A theory of the learnable, Commun. ACM 27(11) (1984) 1134-1142.
- [55] A. Webb, Statistical Pattern Recognition, second ed., John Wiley, 2002.
- [56] D. Wolpert, Stacked generalization, Neural Networks 5 (1992) 241-259.
- [57] D. Wolpert, The lack of a priori distinctions between learning algorithms, Neural Comput. 8(7) (1996) 1341-1390.
- [58] Y. Wu, H. Tjelmeland, M. West, Bayesian CART: Prior structure and MCMC computations, J. Comput. Graph. Stat. 16(1) (2007) 44-66.

This page intentionally left blank

PARAMETER LEARNING: A CONVEX ANALYTIC PATH

8

CHAPTER OUTLINE

8.1	Introduction	332
8.2	Convex Sets and Functions	333
8.2.1	Convex Sets	333
8.2.2	Convex Functions	334
8.3	Projections onto Convex Sets	337
8.3.1	Properties of Projections	341
8.4	Fundamental Theorem of Projections onto Convex Sets	345
8.5	A Parallel Version of POCS	348
8.6	From Convex Sets to Parameter Estimation and Machine Learning	349
8.6.1	Regression	349
8.6.2	Classification	351
8.7	Infinite Many Closed Convex Sets: The Online Learning Case	353
8.7.1	Convergence of APSM	355
<i>Some Practical Hints .</i>		356
8.8	Constrained Learning	360
8.9	The Distributed APSM	361
8.10	Optimizing Nonsmooth Convex Cost Functions	362
8.10.1	Subgradients and Subdifferentials	363
8.10.2	Minimizing Nonsmooth Continuous Convex Loss Functions: The Batch Learning Case	366
<i>The Subgradient Method .</i>		367
<i>The Generic Projected Subgradient Scheme .</i>		369
<i>The Projected Gradient Method (PGM) .</i>		369
<i>Projected Subgradient Method .</i>		370
8.10.3	Online Learning for Convex Optimization	372
<i>The PEGASOS Algorithm .</i>		374
8.11	Regret Analysis	375
<i>Regret Analysis of the Subgradient Algorithm .</i>		377
8.12	Online Learning and Big Data Applications: A Discussion	379
<i>Approximation, Estimation and Optimization Errors .</i>		379
<i>Batch Versus Online Learning .</i>		381

8.13 Proximal Operators	384
8.13.1 Properties of the Proximal Operator	387
8.13.2 Proximal Minimization	388
<i>Resolvent of the Subdifferential Mapping</i>	389
8.14 Proximal Splitting Methods for Optimization	390
<i>The Proximal Forward-Backward Splitting Operator</i>	391
<i>Alternating Direction Method of Multipliers (ADMM)</i>	392
<i>Mirror Descent Algorithms</i>	393
Problems.....	394
<i>MATLAB Exercises</i>	397
8.15 Appendix to Chapter 8	398
References.....	403

8.1 INTRODUCTION

The theory of convex sets and functions has a rich history and has been the focus of intense study for over a century in mathematics. In the terrain of applied sciences and engineering, the revival of interest on convex functions and optimization is traced back in the early 1980s. In addition to the increased processing power that became available via the use of computers, certain theoretical developments were catalytic in demonstrating the power of such techniques. The advent of the so-called interior point methods opened a new path in solving the classical linear programming task. Moreover, it was increasingly realized that, despite its advantages, the least-squares cost function also has a number of drawbacks, particularly in the presence of non-Gaussian noise and the presence of outliers. It has been demonstrated that the use of alternative cost functions, which may not even be differentiable, can alleviate a number of problems that are associated with the least-squares methods. Furthermore, the increased interest in robust learning methods brought into the scene the need for nontrivial constraints, which the optimized solution has to respect. In the machine learning community, the discovery of support vector machines, to be treated in [Chapter 11](#), played an important role in popularizing convex optimization techniques.

The goal of this chapter is to present some basic notions and definitions related to convex analysis and optimization in the context of machine learning and signal processing. Convex optimization is a discipline in itself, and it cannot be summarized in a chapter. Our emphasis here is on computationally light techniques with a focus on online versions, which are gaining in importance in the context of big data applications. A related discussion is also part of this chapter.

The material revolves around two families of algorithms. One goes back to the classical work of Von Neumann on projections on convex sets, which is reviewed together with its more recent online versions. The notions of projection and related properties are treated in some detail. The method of projections, in the context of constrained optimization, is gaining in popularity recently.

The other family of algorithms, that is considered, builds around the notion of subgradient for optimizing nondifferentiable convex functions and generalizations of the gradient descent family, discussed in [Chapter 5](#). Further, we introduce a powerful tool for analyzing the performance of online

algorithms for convex optimization, known as regret analysis, and present a case study. We touch on a current trend in convex optimization, involving proximal and mirror descent methods.

8.2 CONVEX SETS AND FUNCTIONS

Although most of the algorithms we will discuss in this chapter refer to vector variables in Euclidean spaces, which is in line with what we have done so far in this book, the definitions and some of the fundamental theorems will be stated in the context of the more general case of Hilbert spaces.¹ This is because the current chapter will also serve the needs of subsequent chapters, whose setting is that of infinite dimensional Hilbert spaces. For those readers who are not interested in such spaces, all they need to know is that a Hilbert space is a generalization of the Euclidean one, allowing for infinite dimensions. To serve the needs of these readers, we will be careful in pointing out the differences between Euclidean and the more general Hilbert spaces in the theorems, whenever this is required.

8.2.1 CONVEX SETS

Definition 8.1. A nonempty subset C of a Hilbert space \mathbb{H} , $C \subseteq \mathbb{H}$, is called *convex*, if $\forall \mathbf{x}_1, \mathbf{x}_2 \in C$ and $\forall \lambda \in [0, 1]$, the following holds true²

$$\mathbf{x} := \lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2 \in C. \quad (8.1)$$

Note that if $\lambda = 1$, $\mathbf{x} = \mathbf{x}_1$, and if $\lambda = 0$, $\mathbf{x} = \mathbf{x}_2$. For any other value of λ in $[0, 1]$, \mathbf{x} lies in the line segment joining \mathbf{x}_1 and \mathbf{x}_2 . Indeed, from (8.1) we can write

$$\mathbf{x} - \mathbf{x}_2 = \lambda(\mathbf{x}_1 - \mathbf{x}_2), \quad 0 \leq \lambda \leq 1.$$

Figure 8.1 shows two examples of convex sets, in the two-dimensional Euclidean space, \mathbb{R}^2 . In Figure 8.1a, the set comprises all points whose Euclidean (ℓ_2) norm is less than or equal to one,

$$C_2 = \left\{ \mathbf{x} : \sqrt{x_1^2 + x_2^2} \leq 1 \right\}.$$

Sometimes we refer to C_2 as the ℓ_2 -ball of radius equal to one. Note that the set includes all the points *on and inside the circle*. The set in Figure 8.1b comprises all the points *on and inside the rhombus* defined by,

$$C_1 = \left\{ \mathbf{x} : |x_1| + |x_2| \leq 1 \right\}.$$

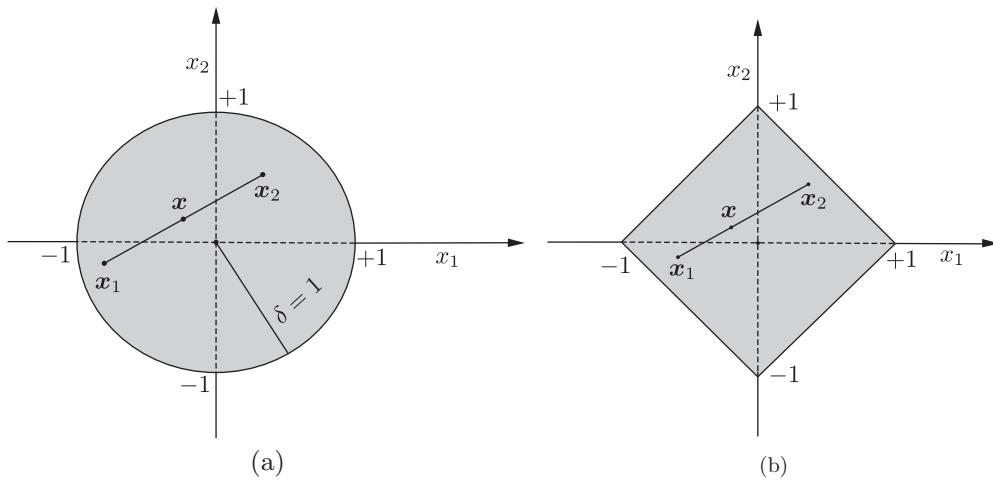
Because the sum of the absolute values of the components of a vector defines the ℓ_1 -norm, that is, $\|\mathbf{x}\|_1 := |x_1| + |x_2|$, in analogy to C_2 we call the set C_1 as the ℓ_1 -ball of radius equal to one. In contrast, the sets whose ℓ_2 and ℓ_1 norms are equal to one, or in other words,

$$\bar{C}_2 = \left\{ \mathbf{x} : x_1^2 + x_2^2 = 1 \right\}, \bar{C}_1 = \left\{ \mathbf{x} : |x_1| + |x_2| = 1 \right\},$$

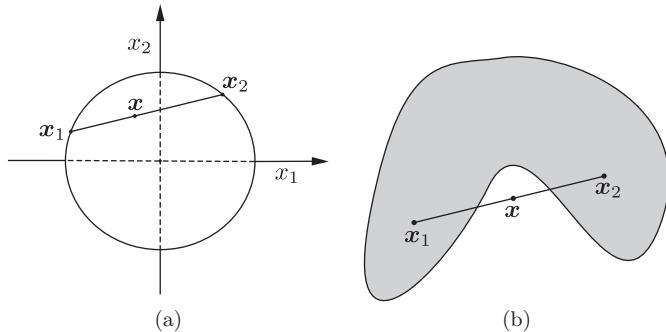
are not convex (Problem 8.2). Figure 8.2 shows two examples of nonconvex sets.

¹ The mathematical definition of a Hilbert space is provided in Section 8.15.

² In conformity with Euclidean vector spaces and for the sake of notational simplicity, we will keep the same notation and denote the elements of a Hilbert space with lowercase bold letters.

**FIGURE 8.1**

(a) The ℓ_2 -ball of radius $\delta = 1$ comprises all points with Euclidean norm less than or equal to $\delta = 1$. (b) The ℓ_1 -ball consists of all the points with ℓ_1 norm less than or equal to $\delta = 1$. Both are convex sets.

**FIGURE 8.2**

Examples of two nonconvex sets. In both cases, the point \mathbf{x} does not lie on the same set in which \mathbf{x}_1 and \mathbf{x}_2 belong. In (a) the set comprises all the points whose Euclidean norm is equal to one.

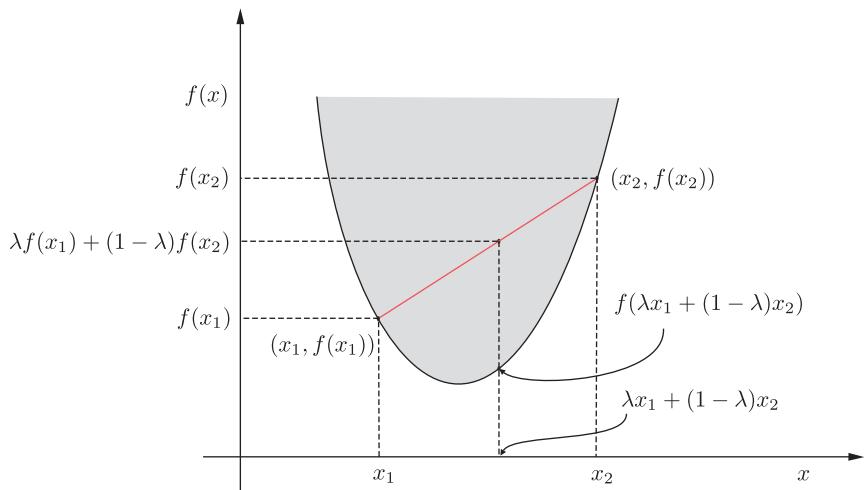
8.2.2 CONVEX FUNCTIONS

Definition 8.2. A function

$$f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R}$$

is called *convex* if \mathcal{X} is convex and if $\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ the following holds true:

$$f(\lambda \mathbf{x}_1 + (1 - \lambda) \mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1 - \lambda) f(\mathbf{x}_2), \quad \lambda \in [0, 1]. \quad (8.2)$$

**FIGURE 8.3**

The line segment joining the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$ lies above the graph of $f(x)$. The shaded region corresponds to the epigraph of the function.

The function is called *strictly convex* if (8.2) holds true with strict inequality when $\lambda \in (0, 1)$, $x_1 \neq x_2$. The geometric interpretation of (8.2) is that the line segment joining the points $(x_1, f(x_1))$ and $(x_2, f(x_2))$ lies above the graph of $f(x)$, as shown in Figure 8.3. We say that a function is *concave* (*strictly concave*) if the negative, $-f$, is convex (strictly convex). Next, we state three important theorems.

Theorem 8.1 (First order convexity condition). *Let $\mathcal{X} \subseteq \mathbb{R}^l$ be a convex set and*

$$f : \mathcal{X} \mapsto \mathbb{R},$$

be a differentiable function. Then, $f(\cdot)$ is convex if and only if $\forall x, y \in \mathcal{X}$,

$$f(y) \geq f(x) + \nabla^T f(x)(y - x). \quad (8.3)$$

The proof of the theorem is given in Problem 8.3. The theorem generalizes to nondifferentiable convex functions; it will be discussed in this context in Section 8.10.

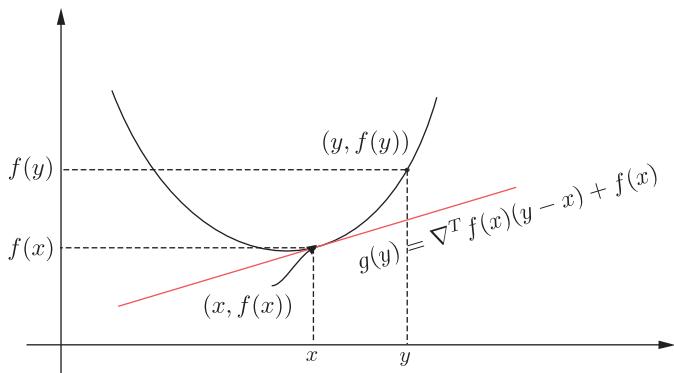
Figure 8.4 illustrates the geometric interpretation of this theorem. It means that the graph of the convex function is located above the graph of the affine function

$$g : y \mapsto \nabla^T f(x)(y - x) + f(x),$$

which defines the tangent hyperplane of the graph at the point $(x, f(x))$.

Theorem 8.2 (Second order convexity condition). *Let $\mathcal{X} \subseteq \mathbb{R}^l$ be a convex set. Then a twice differentiable function, $f : \mathcal{X} \mapsto \mathbb{R}$, is convex (strictly convex) if and only if the Hessian matrix is positive semidefinite (positive definite).*

The proof of the theorem is given in Problem 8.5. Recall that in previous chapters, when we dealt with the squared error loss function, we commented that it is a convex one. Now we are ready to justify this argument. Consider the quadratic function,

**FIGURE 8.4**

The graph of a convex function is above the tangent plane at any point of the respective graph.

$$f(\mathbf{x}) := \frac{1}{2} \mathbf{x}^T Q \mathbf{x} + \mathbf{b}^T \mathbf{x} + c,$$

where Q is a positive definite matrix. Taking the gradient, we have

$$\nabla f(\mathbf{x}) = Q\mathbf{x} + \mathbf{b},$$

and the Hessian matrix is equal to Q , which by assumption is positive definite, hence f is a (strictly) convex function.

In the sequel, two very important notions in convex analysis and optimization are defined.

Definition 8.3. The *epigraph* of a function, f , is defined as the set of points

$$\text{epi}(f) := \left\{ (\mathbf{x}, r) \in \mathcal{X} \times \mathbb{R} : f(\mathbf{x}) \leq r \right\} : \text{Epigraph.} \quad (8.4)$$

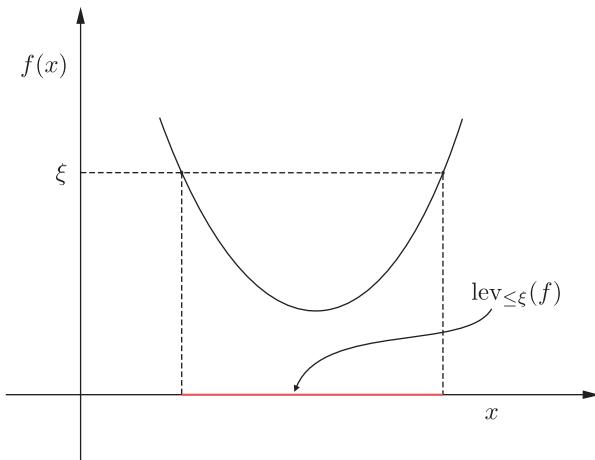
From a geometric point of view, the epigraph is the set of all points in $\mathbb{R}^l \times \mathbb{R}$ that lie on and above the graph of $f(\mathbf{x})$, as it is indicated by the gray shaded region in [Figure 8.3](#). It is important to note that a function is convex if and only if its epigraph is a convex set ([Problem 8.6](#)).

Definition 8.4. Given a real number ξ , the *lower level set* of function, $f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R}$, at height ξ , is defined as

$$\text{lev}_{\leq \xi}(f) := \left\{ \mathbf{x} \in \mathcal{X} : f(\mathbf{x}) \leq \xi \right\} : \text{Level Set at } \xi. \quad (8.5)$$

In words, it is the set of all points at which the function takes a value less than or equal to ξ . The geometric interpretation of the level set is shown in [Figure 8.5](#). It can easily be shown ([Problem 8.7](#)) that if a function, f , is convex then its lower level set is convex for any $\xi \in \mathbb{R}$. The converse is not true. We can easily check out that the function $f(x) = -\exp(x)$ is not convex (as a matter of fact it is concave) and all its lower level sets are convex.

Theorem 8.3 (Local and global minimizers). *Let a convex function, $f : \mathcal{X} \mapsto \mathbb{R}$. Then, if a point \mathbf{x}_* is a local minimizer; it is also a global one and the set of all minimizers is convex. Further, if the function is strictly convex, the minimizer is unique.*

**FIGURE 8.5**

The level set at height ξ comprises all the points in the interval denoted as the “red” segment on the x -axis.

Proof. Inasmuch as the function is convex we know that, $\forall \mathbf{x} \in \mathcal{X}$,

$$f(\mathbf{x}) \geq f(\mathbf{x}_*) + \nabla^T f(\mathbf{x}_*)(\mathbf{x} - \mathbf{x}_*),$$

and because at the minimizer the gradient is zero, we get

$$f(\mathbf{x}) \geq f(\mathbf{x}_*), \quad (8.6)$$

which proves the claim. Let us now denote as

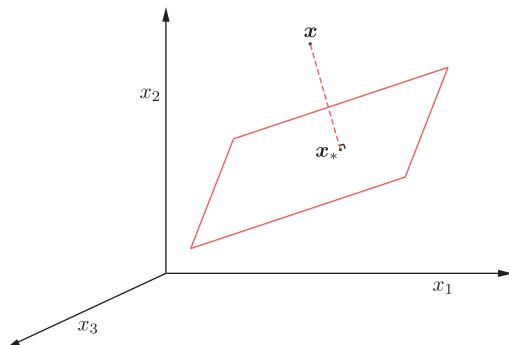
$$f_* = \min_{\mathbf{x}} f(\mathbf{x}). \quad (8.7)$$

Note that the set of all minimizers coincides with the level set at height f_* . Then because the function is convex, we know that the level set $\text{lev}_{f_*}(f)$ is convex, which verifies the convexity of the set of minimizers. Finally, for strictly convex functions, the inequality in (8.6) is a strict one, which proves the uniqueness of the (global) minimizer. The theorem is also true, even if the function is not differentiable ([Problem 8.10](#)). \square

8.3 PROJECTIONS ONTO CONVEX SETS

The projection onto a hyperplane in finite dimensional Euclidean spaces was discussed and used in the context of the affine projection algorithm (APA) algorithm in [Chapter 5](#). The notion of projection will now be generalized to include any closed convex set and also in the framework of general (infinite dimensional) Hilbert spaces.

The concept of projection is among the most fundamental concepts in mathematics, and everyone who has attended classes in basic geometry has used and studied it. What one may not have realized is that while performing a projection, for example, drawing a line segment from a point to a line or a plane, basically he or she solves an *optimization* task. The point \mathbf{x}_* in [Figure 8.6](#), that is, the projection of \mathbf{x} onto the plane, H , in the three-dimensional space, is that point, among all the points lying on the plane, whose (Euclidean) distance from $\mathbf{x} = [x_1, x_2, x_3]^T$ is minimum; in other words,

**FIGURE 8.6**

The projection \mathbf{x}_* of \mathbf{x} onto the plane, minimizes the distance of \mathbf{x} from all the points lying on the plane.

$$\mathbf{x}_* = \min_{y \in H} ((x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2). \quad (8.8)$$

As a matter of fact, what we have learned to do in our early days at school is to solve a *constrained optimization* task. Indeed, (8.8) can equivalently be written as,

$$\begin{aligned} \mathbf{x}_* &= \arg \min_y \|\mathbf{x} - \mathbf{y}\|^2, \\ \text{s.t. } &\theta^T \mathbf{y} + \theta_0 = 0, \end{aligned}$$

where the constraint is the equation describing the specific plane. Our goal herein focuses on generalizing the notion of projection, to employ it to attack more general and complex tasks.

Theorem 8.4. Let C be a nonempty closed³ convex set in a Hilbert space \mathbb{H} and $\mathbf{x} \in \mathbb{H}$. Then, there exists a unique point, denoted as $P_C(\mathbf{x}) \in C$, such as

$\|\mathbf{x} - P_C(\mathbf{x})\| = \min_{y \in C} \|\mathbf{x} - \mathbf{y}\| : \quad \text{Projection of } \mathbf{x} \text{ on } C.$

$P_C(\mathbf{x})$ is called the (metric) projection of \mathbf{x} onto C . Note that if $\mathbf{x} \in C$, then $P_C(\mathbf{x}) = \mathbf{x}$, since this makes the norm $\|\mathbf{x} - P_C(\mathbf{x})\| = 0$.

Proof. The proof comprises two paths. One is to establish *uniqueness* and the other to establish *existence*. Uniqueness is easier, and the proof will be given here. Existence is slightly more technical, and it is provided in [Problem 8.11](#).

To show *uniqueness*, assume that there are two points, namely $\mathbf{x}_{*,1}$ and $\mathbf{x}_{*,2}$, $\mathbf{x}_{*,1} \neq \mathbf{x}_{*,2}$, such as:

$$\|\mathbf{x} - \mathbf{x}_{*,1}\| = \|\mathbf{x} - \mathbf{x}_{*,2}\| = \min_{y \in C} \|\mathbf{x} - \mathbf{y}\|. \quad (8.9)$$

(a) If $\mathbf{x} \in C$, then $P_C(\mathbf{x}) = \mathbf{x}$ is unique, since any other point in C would make $\|\mathbf{x} - P_C(\mathbf{x})\| > 0$.

³ For the needs of this chapter, it suffices to say that a set C is closed if the limit point of any sequence of points in C , lies in C .

- (b) Let $\mathbf{x} \notin C$. Then, mobilizing the parallelogram law of the norm (Appendix 8.15, Eq. (8.149), Problem 8.8) we get

$$\|(\mathbf{x} - \mathbf{x}_{*,1}) + (\mathbf{x} - \mathbf{x}_{*,2})\|^2 + \|(\mathbf{x} - \mathbf{x}_{*,1}) - (\mathbf{x} - \mathbf{x}_{*,2})\|^2 = 2 \left(\|\mathbf{x} - \mathbf{x}_{*,1}\|^2 + \|\mathbf{x} - \mathbf{x}_{*,2}\|^2 \right),$$

or

$$\|2\mathbf{x} - (\mathbf{x}_{*,1} + \mathbf{x}_{*,2})\|^2 + \|\mathbf{x}_{*,1} - \mathbf{x}_{*,2}\|^2 = 2 \left(\|\mathbf{x} - \mathbf{x}_{*,1}\|^2 + \|\mathbf{x} - \mathbf{x}_{*,2}\|^2 \right),$$

and exploiting (8.9) and the fact that $\|\mathbf{x}_{*,1} - \mathbf{x}_{*,2}\| > 0$, we have

$$\left\| \mathbf{x} - \left(\frac{1}{2}\mathbf{x}_{*,1} + \frac{1}{2}\mathbf{x}_{*,2} \right) \right\|^2 < \|\mathbf{x} - \mathbf{x}_{*,1}\|^2. \quad (8.10)$$

However, due to the convexity of C , the point $\frac{1}{2}\mathbf{x}_{*,1} + \frac{1}{2}\mathbf{x}_{*,2}$ lies in C . Also, by the definition of projection, $\mathbf{x}_{*,1}$ is the point with the smallest distance, hence (8.10) cannot be valid. \square

For the existence, one has to use the property of closeness (every sequence in C has its limit in C) as well as the property of completeness of Hilbert spaces, which guarantees that every Cauchy sequence in \mathbb{H} has a limit (Appendix 8.15). The proof is given in Problem 8.11.

Remarks 8.1.

- Note that if $\mathbf{x} \notin C \subseteq \mathbb{H}$, then its projection onto C lies on the *boundary* of C (Problem 8.12).

Example 8.1. Derive analytical expressions for the projections of a point $\mathbf{x} \in \mathbb{H}$, where \mathbb{H} is a real Hilbert space, onto (a) a hyperplane, (b) a halfspace, and (c) the ℓ_2 -ball of radius δ .

- (a) A hyperplane, H , is defined as

$$H := \left\{ \mathbf{y} : \langle \boldsymbol{\theta}, \mathbf{y} \rangle + \theta_0 = 0 \right\},$$

for some $\boldsymbol{\theta} \in \mathbb{H}$ and $\theta_0 \in \mathbb{R}$. If \mathbb{H} breaks down to a Euclidean space, the projection is readily available by simple geometric arguments, and it is given by

$$P_C(\mathbf{x}) = \mathbf{x} - \frac{\langle \boldsymbol{\theta}, \mathbf{x} \rangle + \theta_0}{\|\boldsymbol{\theta}\|^2} \boldsymbol{\theta} : \quad \text{Projection onto a Hyperplane,} \quad (8.11)$$

and it is shown in Figure 8.7. For a general Hilbert space \mathbb{H} , the hyperplane is a closed convex subset of \mathbb{H} , and the projection is still given by the same formula (Problem 8.13).

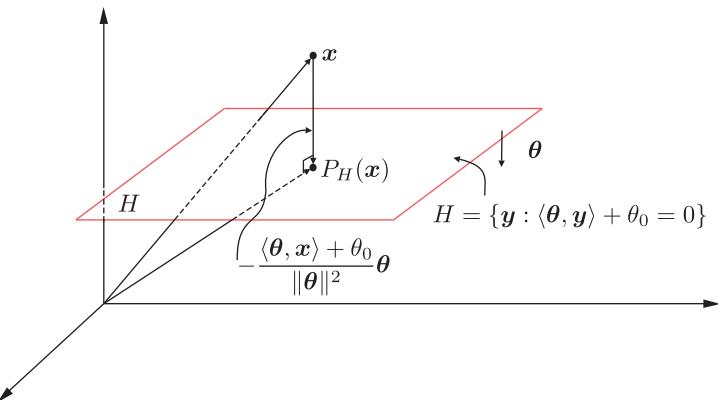
- (b) The definition of a halfspace, H^+ , is given by

$$H^+ = \left\{ \mathbf{y} : \langle \boldsymbol{\theta}, \mathbf{y} \rangle + \theta_0 \geq 0 \right\}, \quad (8.12)$$

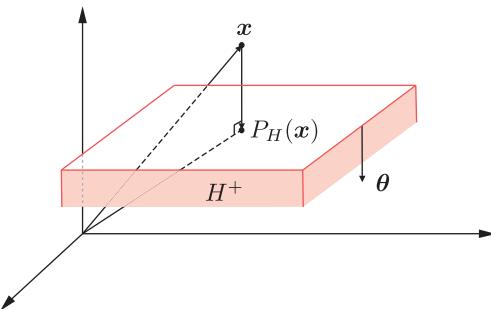
and it is shown in Figure 8.8, for the \mathbb{R}^3 case.

Because the projection lies on the boundary, if $\mathbf{x} \notin H^+$ its projection will lie onto the hyperplane defined by $\boldsymbol{\theta}$ and θ_0 , and it will be equal to \mathbf{x} if $\mathbf{x} \in H^+$; thus, the projection is easily checked out to be

$$P_{H^+}(\mathbf{x}) = \mathbf{x} - \frac{\min \{0, \langle \boldsymbol{\theta}, \mathbf{x} \rangle + \theta_0\}}{\|\boldsymbol{\theta}\|^2} \boldsymbol{\theta} : \quad \text{Projection onto a Halfspace.} \quad (8.13)$$

**FIGURE 8.7**

The projection onto a hyperplane in \mathbb{R}^3 .

**FIGURE 8.8**

Projection onto a halfspace.

- (c) The closed ball centered at $\mathbf{0}$ and of radius δ , denoted as $B[\mathbf{0}, \delta]$, in a general Hilbert space, \mathbb{H} , is defined as

$$B[\mathbf{0}, \delta] = \{y : \|y\| \leq \delta\}.$$

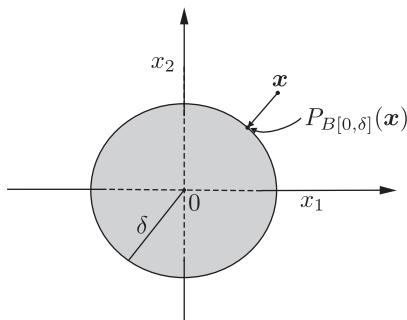
The projection of $x \notin B[\mathbf{0}, \delta]$ onto $B[\mathbf{0}, \delta]$ is given by

$$P_{B[\mathbf{0}, \delta]}(x) = \begin{cases} x, & \text{if } \|x\| \leq \delta, \\ \delta \frac{x}{\|x\|}, & \text{if } \|x\| > \delta, \end{cases} : \quad \text{Projection onto a Closed Ball,} \quad (8.14)$$

and it is geometrically illustrated in Figure 8.9, for the case of \mathbb{R}^2 (Problem 8.14).

Remarks 8.2.

- In the context of sparsity-aware learning, which is dealt with in Chapter 10, a key point is the projection of a point in \mathbb{R}^l (\mathbb{C}^l) onto the ℓ_1 -ball. There it is shown that, given the size of the ball, this projection corresponds to the so-called soft thresholding operation (see Example 8.10 for a definition).

**FIGURE 8.9**

The projection onto a closed ball of radius δ centered at $\mathbf{0}$.

It should be stressed that a linear space equipped with the ℓ_1 -norm is no more Euclidean (Hilbert), inasmuch as this norm *is not induced* by an inner product operation; moreover, uniqueness of the projection with respect to this norm is not guaranteed ([Problem 8.15](#)).

8.3.1 PROPERTIES OF PROJECTIONS

In this section, we summarize some basic properties of the projections. These properties are used to prove a number of theorems and convergence results, associated with algorithms that are developed around the notion of projection.

Readers who are interested only in the algorithms can bypass this section.

Proposition 8.1. *Let \mathbb{H} be a Hilbert space, $C \subseteq \mathbb{H}$ be a closed convex set, and $x \in \mathbb{H}$. Then the projection $P_C(x)$ satisfies the following two properties⁴:*

$$\text{Real}\{\langle x - P_C(x), y - P_C(x) \rangle\} \leq 0, \quad \forall y \in C, \quad (8.15)$$

and

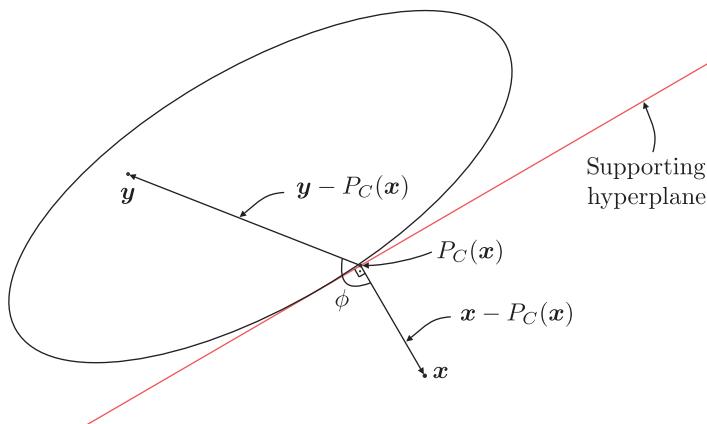
$$\|P_C(x) - P_C(y)\|^2 \leq \text{Real}\{\langle x - y, P_C(x) - P_C(y) \rangle\}, \quad \forall x, y \in \mathbb{H}. \quad (8.16)$$

The proof of the proposition is provided in [Problem 8.16](#). The geometric interpretation of (8.15) for the case of real Hilbert space is shown in [Figure 8.10](#). Note that for a real Hilbert space, the first property becomes,

$$\langle x - P_C(x), y - P_C(x) \rangle \leq 0, \quad \forall y \in C. \quad (8.17)$$

From the geometric point of view, (8.17) means that the angle formed by the two vectors, $x - P_C(x)$ and $y - P_C(x)$, is *obtuse*. The hyperplane that crosses $P_C(x)$ and is orthogonal to $x - P_C(x)$ is known as *supporting* hyperplane and it leaves all points in C on one side and x on the other. It can be shown that if C is closed and convex and $x \notin C$, there is always such a hyperplane; see, for example, [30].

⁴ The theorems are stated here for the general case of complex numbers.

**FIGURE 8.10**

The vectors $y - P_C(x)$ and $x - P_C(x)$ form an obtuse angle, ϕ .

Lemma 8.1. Let S be a closed subspace, $S \subseteq \mathbb{H}$ in a Hilbert space \mathbb{H} . Then $\forall x, y \in \mathbb{H}$, the following properties hold true:

$$\langle x, P_S(y) \rangle = \langle P_S(x), y \rangle = \langle P_S(x), P_S(y) \rangle, \quad (8.18)$$

and

$$P_S(ax + by) = aP_S(x) + bP_S(y), \quad (8.19)$$

where a and b are arbitrary scalars. In other words, the projection operation on a closed subspace is a linear one ([Problem 8.17](#)). Recall that in a Euclidean space all subspaces are closed, hence the linearity is always valid.

It can be shown ([Problem 8.18](#)) that, if S is a closed subspace in a Hilbert space \mathbb{H} , its orthogonal complement, S^\perp , is also a closed subspace, such as $S \cap S^\perp = \{\mathbf{0}\}$; by definition, the orthogonal compliment, S^\perp , is the set whose elements are orthogonal to each element of S . Moreover, $\mathbb{H} = S \oplus S^\perp$; that is, each element, $x \in \mathbb{H}$, can be uniquely decomposed as,

$$x = P_S(x) + P_{S^\perp}(x), \quad x \in \mathbb{H} : \quad \text{For Closed Subspaces,} \quad (8.20)$$

as this is demonstrated in [Figure 8.11](#).

Definition 8.5. Let a mapping

$$T : \mathbb{H} \mapsto \mathbb{H}.$$

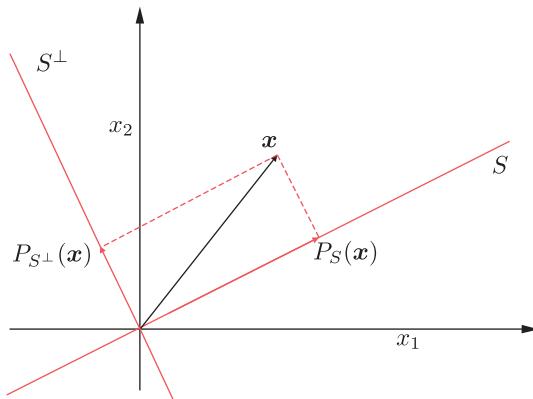
T is called *nonexpansive* if $\forall x, y \in \mathbb{H}$

$$\|T(x) - T(y)\| \leq \|x - y\| : \quad \text{Nonexpansive Mapping.} \quad (8.21)$$

Proposition 8.2. Let C be a closed convex set in a Hilbert space \mathbb{H} . Then, the associated projection operator

$$P_C : \mathbb{H} \mapsto C,$$

is nonexpansive.

**FIGURE 8.11**

Every point in a Hilbert space \mathbb{H} can be uniquely decomposed into the sum of its projections on any *closed* subspace S and its orthogonal complement S^\perp .

Proof. Let $x, y \in \mathbb{H}$. Recall property (8.16), that is,

$$\|P_C(x) - P_C(y)\|^2 \leq \text{Real} \{\langle x - y, P_C(x) - P_C(y) \rangle\}. \quad (8.22)$$

Moreover, by employing the Schwarz inequality (Appendix 8.15, Eq. (8.147)), we get

$$|\langle x - y, P_C(x) - P_C(y) \rangle| \leq \|x - y\| \|P_C(x) - P_C(y)\|. \quad (8.23)$$

Combining (8.22) and (8.23), we readily obtain that

$$\|P_C(x) - P_C(y)\| \leq \|x - y\|. \quad (8.24)$$

□

Figure 8.12 provides a geometric interpretation of (8.24). The property of nonexpansiveness, as well as a number of its variants, for example, [6, 18, 30, 78, 79], are of paramount importance in convex set theory and learning. It is the property that guarantees the convergence of an algorithm, which comprises a sequence of successive projections (mappings), to the so-called *fixed point set*; that is, to the set whose elements are left unaffected by the respective mapping, T , shown as,

$\text{Fix}(T) = \{x \in \mathbb{H} : T(x) = x\} : \quad \text{Fixed Point Set.}$

In the case of a projection operator on a closed convex set, we know that the respective fixed point set is the set C itself, since $P_C(x) = x$, $\forall x \in C$.

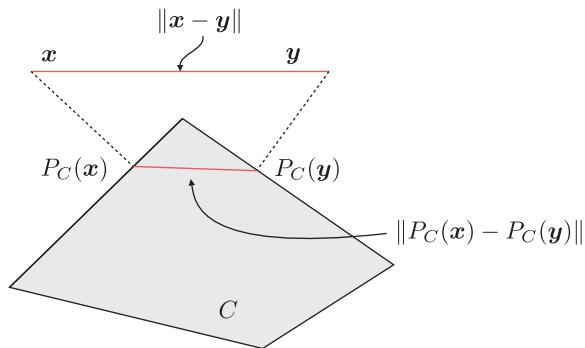
Definition 8.6. Let C be a closed convex set in a Hilbert space. An operator

$$T_C : \mathbb{H} \mapsto C$$

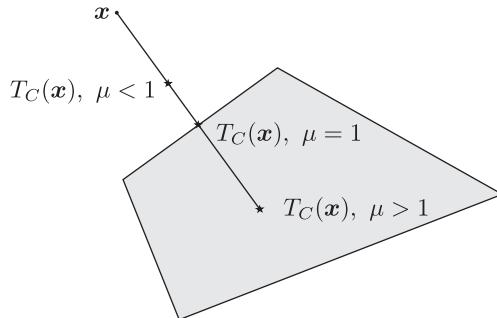
is called *relaxed projection* if

$$T_C := I + \mu(P_C - I), \quad \mu \in (0, 2),$$

or in other words, $\forall x \in \mathbb{H}$

**FIGURE 8.12**

The nonexpansiveness property of the projection operator, $P_C(\cdot)$, guarantees that the distance between two points can never be smaller than the distance between their respective projections on a closed convex set.

**FIGURE 8.13**

Geometric illustration of the relaxed projection operator.

$$T_C(x) = x + \mu(P_C(x) - x), \quad \mu \in (0, 2) : \quad \text{Relaxed Projection on } C.$$

We readily see that for $\mu = 1$, $T_C(x) = P_C(x)$. Figure 8.13 shows the geometric illustration of the relaxed projection. Observe that for different values of $\mu \in (0, 2)$, the relaxed projection traces all points in the line segment joining the points x and $x + 2(P_C(x) - x)$. Note that

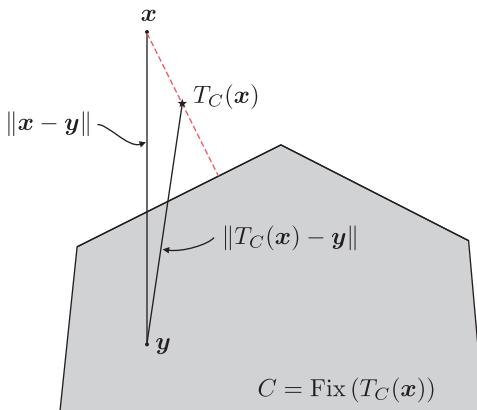
$$T_C(x) = x, \quad \forall x \in C,$$

that is, $\text{Fix}(T_C) = C$. Moreover, it can be shown that the relaxed projection operator is also nonexpansive, that is, (Problem 8.19)

$$\|T_C(x) - T_C(y)\| \leq \|x - y\|, \quad \forall \mu \in (0, 2).$$

A final property of the relaxed projection, which can also be easily shown (Problem 8.20), is the following:

$$\forall y \in C, \quad \|T_C(x) - y\|^2 \leq \|x - y\|^2 - \eta \|T_C(x) - x\|^2, \quad \eta = \frac{2 - \mu}{\mu}. \quad (8.25)$$

**FIGURE 8.14**

The relaxed projection is a strongly attracting mapping. $T_C(x)$ is closer to any point $y \in C = \text{Fix}(T_C)$ than the point x is.

Such mappings are known as *η -nonexpansive* or *strongly attracting* mappings; it is guaranteed that the distance $\|T_C(x) - y\|$ is *smaller* than $\|x - y\|$ at least by the positive quantity $\eta \|T_C(x) - x\|^2$; that is, the fixed point set $\text{Fix}(T_C) = C$ *strongly attracts* x . The geometric interpretation is given in Figure 8.14.

8.4 FUNDAMENTAL THEOREM OF PROJECTIONS ONTO CONVEX SETS

In this section, one of the most celebrated theorems in the theory of convex sets is stated; the fundamental theorem of projections onto convex sets (POCS). This theorem is at the heart of a number of powerful algorithms and methods, some of which are described in this book. The origin of the theorem is traced back to Von Neumann [93], who proposed the theorem for the case of two subspaces.

Von Neumann was a Hungarian-born American of Jewish descent. He was a child prodigy who earned his Ph.D. at age 22. It is difficult to summarize his numerous significant contributions, which range from pure mathematics, to economics (he is considered the founder of the game theory) and from quantum mechanics (he laid the foundations of the mathematical framework in quantum mechanics) to computer science (he was involved in the development of ENIAC, the first general-purpose electronic computer). He was heavily involved in the Manhattan project for the development of the hydrogen bomb.

Let C_k , $k = 1, 2, \dots, K$, be a *finite* number of *closed* convex sets in a Hilbert space \mathbb{H} , and assume that they share a *nonempty* intersection,

$$C = \bigcap_{k=1}^K C_k \neq \emptyset.$$

Let T_{C_k} , $k = 1, 2, \dots, K$, be the respective relaxed projection mappings

$$T_{C_k} = I + \mu_k(P_{C_k} - I), \quad \mu_k \in (0, 2), \quad k = 1, 2, \dots, K.$$

Form the concatenation of these relaxed projections,

$$T := T_{C_k} T_{C_{k-1}} \cdots T_{C_1},$$

where the specific order is not important. In words, T comprises a sequence of relaxed projections, starting from C_1 . In the sequel, the obtained point is projected onto C_2 , and so on.

Theorem 8.5. *Let $C_k, k = 1, 2, \dots, K$, be closed convex sets in a Hilbert space, \mathbb{H} , with nonempty intersection. Then, for any $x_0 \in \mathbb{H}$, the sequence $(T^n(x_0))$, $n = 1, 2, \dots$ converges weakly to a point in $C = \bigcap_{k=1}^K C_k$.*

The theorem [16, 41] states the notion of *weak convergence*. When \mathbb{H} becomes a Euclidean (finite dimensional) space, the notion of weak convergence coincides with the familiar “standard” definition of (strong) convergence. Weak convergence is a weaker version of the strong convergence, and it is met in infinite dimensional spaces. A sequence, $x_n \in \mathbb{H}$, is said to converge weakly to a point $x_* \in \mathbb{H}$, if $\forall y \in \mathbb{H}$,

$$\langle x_n, y \rangle \xrightarrow[n \rightarrow \infty]{w} \langle x_*, y \rangle,$$

and we write,

$$x_n \xrightarrow[n \rightarrow \infty]{w} x_*.$$

As already said, in Euclidean spaces, weak convergence implies strong convergence. This is not necessarily true for general Hilbert spaces. On the other hand, strong convergence always implies weak convergence, for example, [82] (Problem 8.21). Figure 8.15 gives the geometric illustration of the theorem.

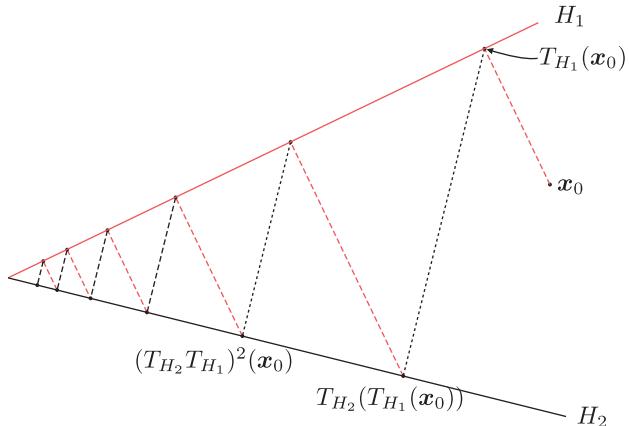


FIGURE 8.15

Geometric illustration of the fundamental theorem of projections onto convex sets (POCS), for $T_{C_i} = P_{C_i}$, $i = 1, 2$, ($\mu_{C_i} = 1$). The closed convex sets are the two straight lines in \mathbb{R}^2 . Observe that the sequence of projections tends to the intersection of H_1, H_2 .

The proof of the theorem is a bit technical for the general case, for example, [82]. However, it can be simplified for the case where the involved convex sets are closed subspaces (Problem 8.23). At the heart of the proof lie (a) the nonexpansiveness property of T_{C_k} , $k = 1, 2, \dots, K$, which is retained by T and (b) the fact that the fixed point set of T is $\text{Fix}(T) = \bigcap_{k=1}^K C_k$.

Remarks 8.3.

- In the special case where all C_k , $k = 1, 2, \dots, K$, are closed subspaces, then

$$T^n(x_0) \longrightarrow P_C(x_0).$$

In other words, the sequence of relaxed projections converges *strongly* to the projection of x_0 on C . Recall that if each C_k , $k = 1, 2, \dots, K$, is a closed subspace of \mathbb{H} , it can easily be shown that their intersection is also a closed subspace. As said before, in a Euclidean space \mathbb{R}^l , all subspaces are closed.

- The previous statement is also true for *linear varieties*. A linear variety is the translation of a subspace by a constant vector a . That is, if S is a subspace, and $a \in \mathbb{H}$, then the set of points

$$S_a = \{y : y = a + x, x \in S\}$$

is a linear variety. Hyperplanes are linear varieties: see, for example, Figure 8.16.

- The scheme resulting from the POCS theorem, employing the relaxed projection operator, is summarized in Algorithm 8.1,

Algorithm 8.1 (The POCS algorithm).

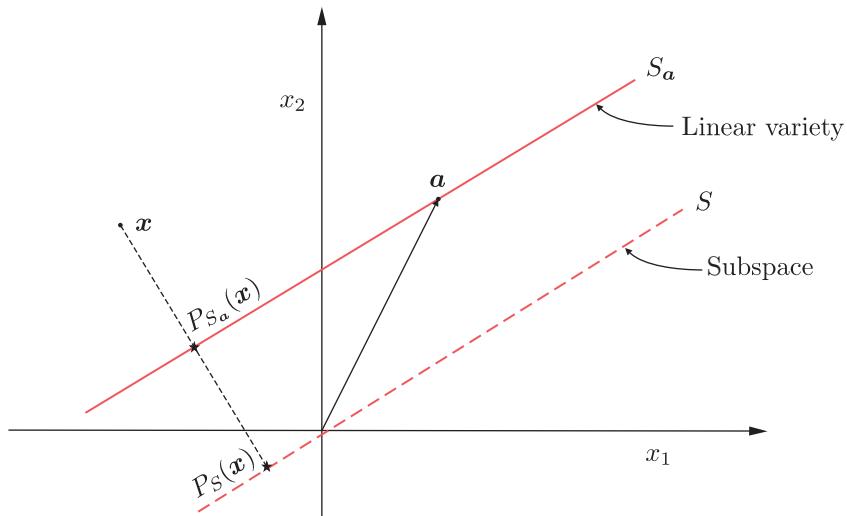


FIGURE 8.16

A hyperplane (not crossing the origin) is a linear variety. P_{S_a} and P_S are the projections of x onto S_a and S , respectively.

- Initialization.
 - Select $\mathbf{x}_0 \in \mathbb{H}$.
 - Select $\mu_k \in (0, 2)$, $k = 1, 2, \dots, K$
- **For** $n = 1, 2, \dots$, **Do**
 - $\hat{\mathbf{x}}_{0,n} = \mathbf{x}_{n-1}$
 - **For** $k = 1, 2, \dots, K$ **Do**
 - **End For**
 - $\mathbf{x}_n = \hat{\mathbf{x}}_K$.
- **End For**

$$\hat{\mathbf{x}}_{k,n} = \hat{\mathbf{x}}_{k-1,n} + \mu_k (P_{C_k}(\hat{\mathbf{x}}_{k-1,n}) - \hat{\mathbf{x}}_{k-1,n}) \quad (8.26)$$

8.5 A PARALLEL VERSION OF POCS

In [65], a parallel version of the POCS algorithm was stated. In addition to its computational advantages (when parallel processing can be exploited), this scheme will be our vehicle for generalizations to the online processing, where one can cope with the case where the number of convex sets becomes infinite (or very large in practice). The proof for the parallel POCS is slightly technical and relies heavily on the results stated in the previous section. The concept behind the proof is to construct appropriate product spaces, and this is the reason that the algorithm is also referred to as POCS in *product spaces*. For the detailed proof, the interested reader may consult [65].

Theorem 8.6. *Let $C_k, k = 1, 2, \dots, K$, be closed convex sets, in a Hilbert space, \mathbb{H} . Then, for any $\mathbf{x}_0 \in \mathbb{H}$, the sequence \mathbf{x}_n , defined as*

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \mu_n \left(\sum_{k=1}^K \omega_k P_{C_k}(\mathbf{x}_{n-1}) - \mathbf{x}_{n-1} \right), \quad (8.27)$$

weakly converges to a point in $\bigcap_{k=1}^K C_k$, if

$$0 < \mu_n \leq M_n,$$

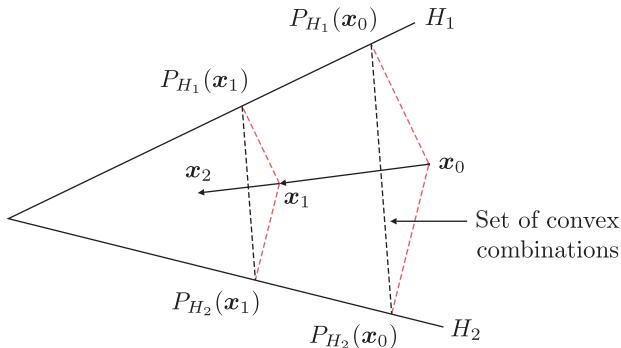
and

$$M_n := \sum_{k=1}^K \frac{\omega_k \|P_{C_k}(\mathbf{x}_{n-1}) - \mathbf{x}_{n-1}\|^2}{\|\sum_{k=1}^K \omega_k P_{C_k}(\mathbf{x}_{n-1}) - \mathbf{x}_{n-1}\|^2}, \quad (8.28)$$

where $\omega_k > 0$, $k = 1, 2, \dots, K$, such that

$$\sum_{k=1}^K \omega_k = 1.$$

Update recursion (8.27) says that at each iteration, all projections on the convex sets take place *concurrently*, and then they are *convexly* combined. The extrapolation parameter, μ_n , is chosen in interval $(0, M_n]$, where M_n is recursively computed in (8.28), so that convergence is guaranteed. Figure 8.17 illustrates the updating process.

**FIGURE 8.17**

The parallel POCS algorithm for the case of two (hyperplanes) lines in \mathbb{R}^2 . At each step, the projections on H_1 and H_2 are carried out in parallel and then they are convexly combined.

8.6 FROM CONVEX SETS TO PARAMETER ESTIMATION AND MACHINE LEARNING

Let us now see how this elegant theory can be turned into a useful tool for parameter estimation in machine learning. We will demonstrate the procedure using two examples.

8.6.1 REGRESSION

Consider the regression model, relating input-output observation points,

$$y_n = \boldsymbol{\theta}_o^T \mathbf{x}_n + \eta_n, \quad (\mathbf{y}_n, \mathbf{x}_n) \in \mathbb{R} \times \mathbb{R}^l, \quad n = 1, 2, \dots, N, \quad (8.29)$$

where $\boldsymbol{\theta}_o$ is the unknown parameter vector. Assume that η_n is a bounded noise sequence, that is,

$$|\eta_n| \leq \epsilon. \quad (8.30)$$

Then, (8.29), (8.30) guarantee that

$$|y_n - \mathbf{x}_n^T \boldsymbol{\theta}_o| \leq \epsilon. \quad (8.31)$$

Consider now the following set of points

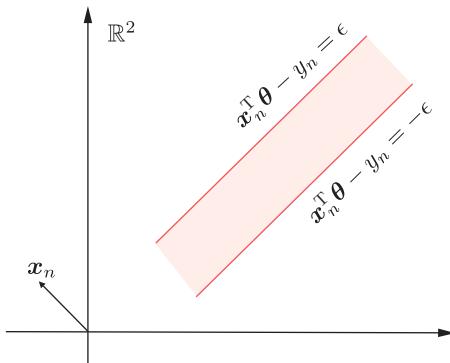
$S_\epsilon = \left\{ \boldsymbol{\theta} : |y_n - \mathbf{x}_n^T \boldsymbol{\theta}| \leq \epsilon \right\}$

(8.32)

This set is known as a *hyperslab*, and it is geometrically illustrated in Figure 8.18. The definition is generalized for any \mathbb{H} by replacing the inner product notation as $\langle \mathbf{x}_n, \boldsymbol{\theta} \rangle$. The set comprises all the points that lie in the region formed by the two hyperplanes

$$\begin{aligned} \mathbf{x}_n^T \boldsymbol{\theta} - y_n &= \epsilon, \\ \mathbf{x}_n^T \boldsymbol{\theta} - y_n &= -\epsilon. \end{aligned}$$

This region is trivially shown to be a *closed convex set*. Note that *every pair* of training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, defines a hyperslab of different orientation (depending on \mathbf{x}_n) and position

**FIGURE 8.18**

Each pair of training points, (y_n, \mathbf{x}_n) , defines a hyperslab in the parameters' space.

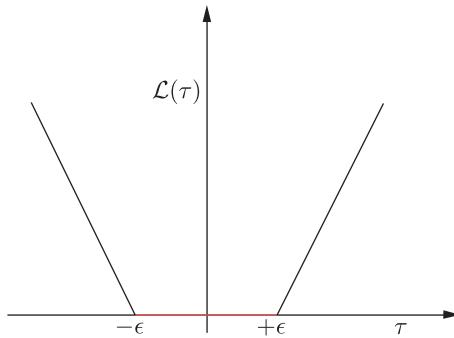
in space (determined by y_n). Moreover, (8.31) guarantees that the unknown, $\boldsymbol{\theta}_o$, lies within all these hyperslabs; hence, $\boldsymbol{\theta}_o$ lies in their *intersection*. All we need now is to derive the projection operator onto hyperslabs (we will do it soon), and use one of the POCS schemes to find a point in the intersection. Assuming that enough training points are available and that the intersection is “small” enough, then any point in this intersection will be “close” to $\boldsymbol{\theta}_o$. Note that such a procedure is not based on optimization arguments. Recall, however, that even in optimization techniques, iterative algorithms have to be used, and in practice, iterations have to stop after a finite number of steps. Thus, one can only approximately reach the optimal value. More on these issues and related convergence properties will be discussed later in this chapter.

The obvious question now is what happens if the noise is not bounded. There are two answers to this point. First, in any practical application where measurements are involved, the noise has to be bounded. Otherwise, the circuits will be burned out. So, at least conceptually, this assumption does not conflict with what happens in practice. It is a matter of selecting the right value for ϵ . The second answer is that one can choose ϵ to be a few times the standard deviation of the assumed noise model. Then, $\boldsymbol{\theta}_o$ will lie in these hyperslabs with high probability. We will discuss strategies for selecting ϵ , but our goal in this section is to discuss the main rationale in using the theory in practical applications. Needless to say there is nothing divine around hyperslabs. Other closed convex sets can also be used, if the nature of the noise in a specific application suggests a different type of convex sets.

It is now interesting to look at the set where the solution lies, in this case at the hyperslab, from a different perspective. Consider the loss function,

$$\mathcal{L}(y, \boldsymbol{\theta}^T \mathbf{x}) = \max(0, |y - \boldsymbol{\theta}^T \mathbf{x}| - \epsilon) : \text{Linear } \epsilon\text{-Insensitive Loss Function,} \quad (8.33)$$

which is illustrated in Figure 8.19 for the case $\boldsymbol{\theta} \in \mathbb{R}$. This is known as linear ϵ -insensitive loss function, and it has been popularized in the context of support vector regression (Chapter 11). For all $\boldsymbol{\theta}$ s, which lie within the hyperslab defined in (8.32), the loss function scores a zero. For points outside the hyperslab, there is a linear increase of its value. Thus, the hyperslab is the *zero level set* of the linear ϵ -insensitive loss function, defined *locally* according to the point (y_n, \mathbf{x}_n) . Thus, although no optimization concept is associated with POCS, the choice of the closed convex sets can be done to minimize “locally,” at each point, a convex loss function by selecting its zero level set.

**FIGURE 8.19**

The linear ϵ -insensitive loss function, $\tau = y - \theta^T x$. Its value is zero if $|\tau| < \epsilon$, and increases linearly for $|\tau| \geq \epsilon$.

We conclude our discussion by providing the projection operator of a hyperslab, S_ϵ . It is trivially shown that, given θ , its projection onto S_ϵ (defined by (y_n, x_n, ϵ)) is given by

$$P_{S_\epsilon} = \theta + \beta_\theta(y_n, x_n)x_n, \quad (8.34)$$

where

$$\beta_\theta(y_n, x_n) = \begin{cases} \frac{y_n - \langle x_n, \theta \rangle - \epsilon}{\|x_n\|^2}, & \text{if } \langle x_n, \theta \rangle - y_n < -\epsilon, \\ 0, & \text{if } |\langle x_n, \theta \rangle - y_n| \leq \epsilon, \\ \frac{y_n - \langle x_n, \theta \rangle + \epsilon}{\|x_n\|^2}, & \text{if } \langle x_n, \theta \rangle - y_n > \epsilon. \end{cases} \quad (8.35)$$

That is, if the point lies within the hyperslab, it coincides with its projection. Otherwise, the projection is on one of the two hyperplanes (depending on which side of the hyperslab the point lies), which define S_ϵ . Recall that the projection of a point lies on the boundary of the corresponding closed convex set.

8.6.2 CLASSIFICATION

Let us consider the two-class classification task, and assume that we are given the set of training points, $(y_n, x_n), n = 1, 2, \dots, N$.

Our goal now will be to design a linear classifier so that to score

$$\theta^T x_n \geq \rho, \text{ if } y_n = +1,$$

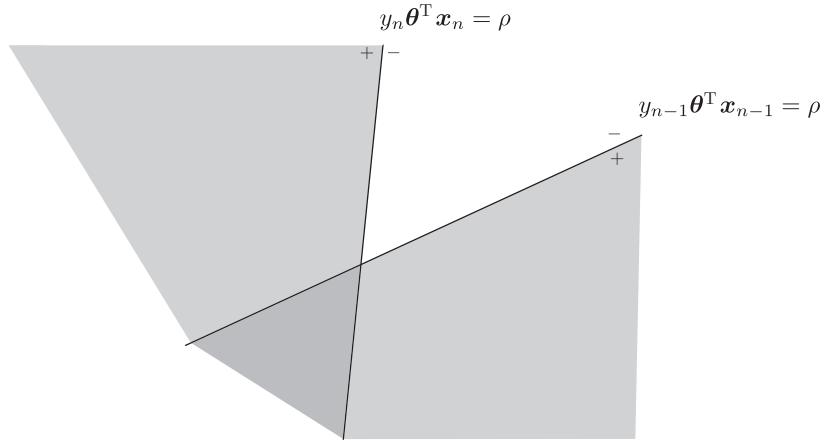
and

$$\theta^T x_n \leq -\rho, \text{ if } y_n = -1.$$

This requirement can be expressed as: Given $(y_n, x_n) \in \{-1, 1\} \times \mathbb{R}^{l+1}$, design a linear classifier,⁵ $\theta \in \mathbb{R}^{l+1}$, such that

$$y_n \theta^T x_n \geq \rho > 0. \quad (8.36)$$

⁵ Recall from Chapter 3, that this formulation covers the general case where a bias term is involved, by increasing the dimensionality of x_n and adding 1 as its last element.

**FIGURE 8.20**

Each training point, (y_n, \mathbf{x}_n) , defines a halfspace in the parameters $\boldsymbol{\theta}$ -space, and the linear classifier will be searched in the intersection of all these halfspaces.

Note that, given y_n, \mathbf{x}_n , and ρ , (8.36) defines a *halfspace* (Example 8.1); this is the reason that we used “ $\geq \rho$ ” rather than a strict inequality. In other words, all $\boldsymbol{\theta}$ ’s, which satisfy the desired inequality (8.36) lie in this halfspace. Since each pair, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, defines a single halfspace, our goal now becomes that of trying to find a point at the intersection of all these halfspaces. This intersection is guaranteed to be nonempty if the classes are linearly separable. Figure 8.20 illustrates the concept. The more realistic case, of nonlinearly separable classes, will be treated in Chapter 11, where a mapping in a high dimensional (kernel) space makes the probability of two classes being linearly separable to tend to 1, as the dimensionality of the kernel space goes to infinity.

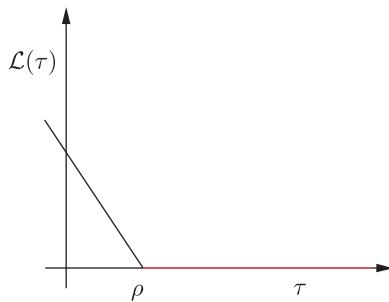
The halfspace associated with a training pair, (y_n, \mathbf{x}_n) , can be seen as the level set of height zero of the so-called *hinge loss* function, defined as

$$\mathcal{L}_\rho(y, \boldsymbol{\theta}^T \mathbf{x}) = \max(0, \rho - y \boldsymbol{\theta}^T \mathbf{x}) : \quad \text{Hinge Loss Function,} \quad (8.37)$$

whose graph is shown in Figure 8.21. Thus, choosing the halfspace as the closed convex set to represent (y_n, \mathbf{x}_n) , is equivalent with selecting the zero level set of the hinge loss, “adjusted” for the point (y_n, \mathbf{x}_n) .

Remarks 8.4.

- In addition to the two applications typical of the machine learning point of view, POCS has been applied in a number of other applications; see, for example, [18, 24, 82, 84], for further reading.
- If the involved sets do not intersect, that is, $\bigcap_{k=1}^K C_k = \emptyset$, then it has been shown [25] that, the parallel version of POCS in (8.27) converges to a point whose weighted squared distance from each one of the convex sets (defined as the distance of the point from its respective projection) is minimized.
- Attempts to generalize the theory to nonconvex sets have also been made, for example, [82] and more recently in the context of sparse modeling in [80].

**FIGURE 8.21**

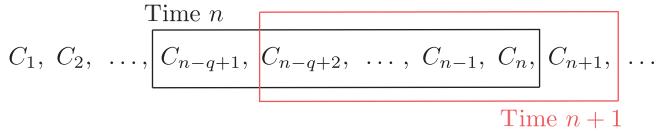
The hinge loss function. For the classification task, $\tau = y\theta^T x$, its value is zero if $\tau \geq \rho$, and increases linearly for $\tau < \rho$.

- When $C := \bigcap_{k=1}^K C_k \neq \emptyset$, we say that the problem is *feasible* and the intersection C is known as the *feasibility set*. The closed convex sets, C_k , $k = 1, 2, \dots, K$, are sometimes called the *property sets*, for obvious reasons. In both previous examples, namely the regression and the classification, we commented that the involved property sets resulted as the 0-level sets of a loss function \mathcal{L} . Hence, assuming that the problem is feasible (the cases of bounded noise in regression and linearly separable classes in classification), any solution in the feasible set C , will also be a *minimizer* of the respective loss functions in (8.33), (8.37), respectively. Thus, although optimization did not enter into our discussion, there can be an optimizing flavor in the POCS method. Moreover, note that in this case, the loss functions need *not* be differentiable and the techniques we discussed so far in the previous chapters are not applicable. We will return to this issue later on in Section 8.10.

8.7 INFINITE MANY CLOSED CONVEX SETS: THE ONLINE LEARNING CASE

In our discussion so far, we have assumed a finite number, K , of closed convex (property) sets. To land at their intersection (feasibility set) one has to cyclically project onto all of them or to perform the projections in parallel. Such a strategy is not appealing for the online processing scenario. At every time instant, a new pair of observations becomes available, which defines a new property set. Hence, in this case, the number of the available convex sets gets increased. Visiting all the available sets makes the complexity time dependent and after some time the required computational resources will become unmanageable.

An alternative viewpoint was suggested in [96–98], and later on extended in [73, 99, 100]. The main idea here is that at each time instant, n , a pair of output-input training data is received and a (property) closed convex set, C_n , is constructed. The time index, n , is left to grow unbounded. However, at each time instant, the q (a user-defined parameter) most recently constructed property sets are considered. In other words, the parameter q defines a *sliding window* in time. At each time instant, projections/relaxed projections are performed within this time window. The rationale is illustrated in Figure 8.22. Thus, the number of sets onto which projections are performed does not grow with time, their number remains finite, and it is fixed by the user. The developed algorithm is an offspring of the parallel version of

**FIGURE 8.22**

At time n , the property sets C_{n-q+1}, \dots, C_n are used, while at time $n + 1$, the sets $C_{n-q+2}, \dots, C_{n+1}$ are considered. Thus, the required number of projections does not grow with time.

POCS and it is known as *adaptive projected subgradient method* (APSM), for reasons that will become clear later on in [Section 8.10.3](#). We will describe the algorithm in the context of regression. Following the discussion in [Section 8.6](#), as each pair, $(y_n, \mathbf{x}_n) \in \mathbb{R} \times \mathbb{R}^l$, becomes available, a hyperslab, $S_{\epsilon,n}, n = 1, 2, \dots$, is constructed and the goal is to find a $\boldsymbol{\theta} \in \mathbb{R}^l$, that lies in the intersection of all these property sets, starting from an arbitrary value, $\boldsymbol{\theta}_0 \in \mathbb{R}^l$.

Algorithm 8.2 (The APSM algorithm).

- Initialization
 - Choose $\boldsymbol{\theta}_0 \in \mathbb{R}^l$.
 - Choose q ; The number of property sets to be processed at each time instant.
- **For** $n = 1, 2, \dots, q - 1$, **Do**; Initial period, that is, $n < q$.
 - Choose $\omega_1, \dots, \omega_n : \sum_{k=1}^n \omega_k = 1, \omega_k \geq 0$
 - Select μ_n

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \left(\sum_{k=1}^n \omega_k P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right) \quad (8.38)$$

- **End For**
- **For** $n = q, q + 1, \dots$, **Do**
 - Choose $\omega_n, \dots, \omega_{n-q+1}$; usually $\omega_k = \frac{1}{q}, k = n - q + 1, \dots, n$.
 - Select μ_n

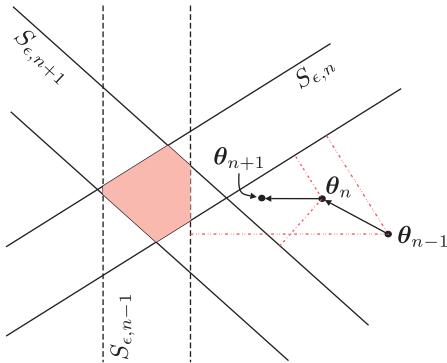
$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu_n \left(\sum_{k=n-q+1}^n \omega_k P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right) \quad (8.39)$$

- **End For**

The extrapolation parameter can now be chosen in the interval $(0, 2M_n)$ in order for convergence to be guaranteed. For the case of (8.39)

$$M_n = \sum_{k=n-q+1}^n \frac{\omega_k \|P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1}\|^2}{\left\| \sum_{k=n-q+1}^n \omega_k P_{S_{\epsilon,k}}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right\|^2}. \quad (8.40)$$

Note that this interval differs from that reported in the case of a finite number of sets, in Eq. (8.27). For the first iteration steps associated with Eq. (8.38), the summations in the above formula starts from $k = 1$ instead of $k = n - q + 1$.

**FIGURE 8.23**

At time, n , $q = 2$ hyperslabs are processed, namely, $S_{\epsilon,n}$, $S_{\epsilon,n-1}$. θ_{n-1} is concurrently projected onto both of them and the projections are convexly combined. The new estimate is θ_n . Next $S_{\epsilon,n+1}$ “arrives” and the process is repeated. Note that at every time, the estimate gets closer to the intersection; the latter will become smaller and smaller, as more hyperslabs arrive.

Recall that, $P_{S_{\epsilon,k}}$ is the projection operation given in (8.34)-(8.35). Note that this is a generic scheme and can be applied with different property sets. All that is needed is to change the projection operator. For example, if classification is considered, all we have to do is to replace $S_{\epsilon,n}$ by the halfspace H_n^+ , defined by the pair $(y_n, \mathbf{x}_n) \in \{-1, 1\} \times \mathbb{R}^{l+1}$, as explained in Section 8.6.2 and use the projection from (8.13); see [75, 76]. At this point, it must be emphasized that the original APSM form (e.g., [98, 99]) is more general and can cover a wide range of convex sets and functions. We will come back to this in Remarks 8.8.

Figure 8.23 illustrates geometrically the APSM algorithm. We have assumed that the number of hyperslabs that are considered for projection at each time instant is $q = 2$. Each iteration comprises:

- q projections, which can be carried out in parallel,
- their convex combination, and
- the update step.

8.7.1 CONVERGENCE OF APSM

The proof of the convergence of the APSM is a bit technical and the interested reader can consult the related references. Here, we can be content with a geometric illustration that intuitively justifies the convergence, under certain assumptions. This geometric interpretation is at the heart of a stochastic approach to the APSM convergence, which was presented in [23].

Assume the noise to be bounded and that there is a true θ_o that generates the data, that is,

$$y_n = \mathbf{x}_n^T \theta_o + \eta_n. \quad (8.41)$$

By assumption,

$$|\eta_n| \leq \epsilon,$$

hence,

$$|\mathbf{x}_n^T \boldsymbol{\theta}_o - y_n| \leq \epsilon.$$

Thus, $\boldsymbol{\theta}_o$ does lie in the intersection of all the hyperslabs of the form

$$|\mathbf{x}_n^T \boldsymbol{\theta} - y_n| \leq \epsilon,$$

and in this case the problem is feasible. The question that is raised is how close one can go, even asymptotically as $n \rightarrow \infty$, to $\boldsymbol{\theta}_o$. For example, if the volume of the intersection is large, even if the algorithm converges to a point in the boundary of this intersection does not necessarily say much about how close the solution is to the true value $\boldsymbol{\theta}_o$. The proof in [23] establishes that the algorithm brings the estimate *arbitrarily close* to $\boldsymbol{\theta}_o$, under some general assumptions concerning the sequence of observations and that the noise is bounded.

To understand what is behind the technicalities of the proof, recall that there are two main geometric issues concerning a hyperslab: (a) its orientation, which is determined by \mathbf{x}_n and (b) its width. In finite dimensional spaces, it is a matter of simple geometry to show that the width of a hyperslab is equal to

$$d = \frac{2\epsilon}{\|\mathbf{x}_n\|}. \quad (8.42)$$

This is a direct consequence of the fact that the distance⁶ of a point, say $\tilde{\boldsymbol{\theta}}$, from the hyperplane defined by the pair (y, \mathbf{x}) , that is,

$$\mathbf{x}^T \boldsymbol{\theta} - y = 0,$$

is equal to

$$\frac{|\mathbf{x}^T \tilde{\boldsymbol{\theta}} - y|}{\|\mathbf{x}\|}.$$

Indeed, let $\bar{\boldsymbol{\theta}}$ be a point on one of the two boundary hyperplanes (e.g., $\mathbf{x}_n^T \boldsymbol{\theta} - y_n = \epsilon$), which define the hyperslab and consider its distance from the other one ($\mathbf{x}_n^T \boldsymbol{\theta} - y_n = -\epsilon$); then, (8.42) is readily obtained.

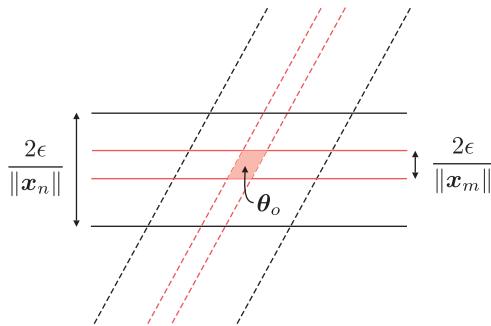
Figure 8.24 shows four hyperslabs in two different directions (one for the full lines and one for the dotted lines). The red hyperslabs are narrower than the black ones. Moreover, all four necessarily include $\boldsymbol{\theta}_o$. If \mathbf{x}_n is left to vary randomly so that any orientation will occur, with high probability and for any orientation, the norm can also take small as well as arbitrarily large values, then intuition says that the volume of the intersection around $\boldsymbol{\theta}_o$ will become arbitrarily small.

Some practical hints

The APSM algorithm needs the setting of three parameters, namely, ϵ , μ_n , and q . It turns out that the algorithm is not particularly sensitive in their choice:

- The choice of the parameter μ_n is similar in concept to the choice of the step-size in the LMS algorithm. In particular, the larger the μ_n the faster the convergence speed, at the expense of a higher steady-state error floor. In practice, a step-size approximately equal to $0.5M_n$ will lead to a

⁶ For Euclidean spaces, this can be easily established by simple geometric arguments; see also, Section 11.10.1.

**FIGURE 8.24**

For each direction, the width of a hyperslab varies inversely proportional to $\|\mathbf{x}_n\|$. In this figure, $\|\mathbf{x}_n\| < \|\mathbf{x}_m\|$ although both vectors point to the same direction. The intersection of hyperslabs of different directions and widths renders the volume of their intersection arbitrarily small around θ_0 .

low steady-state error, albeit the convergence speed will be relatively slow. On the contrary, if one chooses a larger step-size, $1.5M_n$ approximately, then the algorithm enjoys a faster convergence speed, although the steady-state error after convergence is increased.

- Regarding the parameter ϵ , a typical choice is to set $\epsilon \approx \sqrt{2}\sigma$, where σ is the standard deviation of the noise. In practice (see, e.g. [46]), it has been shown that the algorithm is rather insensitive to this parameter. Hence, one needs only a *rough* estimate of the standard deviation.
- Concerning the choice of q , this is analogous to the q used for the APA in [Chapter 5](#). The larger the q is, the faster the convergence becomes; however, large values of q increase complexity as well as the error floor after convergence. In practice, relatively small values of q , for example, a small fraction of the l , can significantly improve the convergence speed compared to the normalized least-mean-squares algorithm (NLMS). Sometimes, one can start with a relatively large value of q , and once the error decreases, q can be given smaller values to achieve lower error floors.

It is important to note that the past data reuse, within the sliding window of length q in the APA algorithm is implemented via the inversion of a $q \times q$ matrix. In the APSM, this is achieved via a sequence of q projections, leading to a complexity of linear dependence on q ; moreover, these projections can be performed in parallel. Furthermore, the APA tends to be more sensitive to the presence of noise, since the projections are carried out on hyperplanes. In contrast, for the APSM case, projections are performed on hyperslabs, which implicitly care for the noise, for example, [97].

Remarks 8.5.

- If the hyperslabs collapse to hyperplanes ($\epsilon = 0$) and $q = 1$, the algorithm becomes the NLMS. Indeed, for this case the projection in (8.39) becomes the projection on the hyperplane, H , defined by (y_n, \mathbf{x}_n) , that is,

$$\mathbf{x}_n^T \boldsymbol{\theta} = y_n,$$

and from (8.11), after making the appropriate notational adjustments, we have

$$P_H(\boldsymbol{\theta}_{n-1}) = \boldsymbol{\theta}_{n-1} - \frac{\mathbf{x}_n^T \boldsymbol{\theta}_{n-1} - y_n}{\|\mathbf{x}_n\|^2} \mathbf{x}_n. \quad (8.43)$$

Plugging (8.43) into (8.39), we get

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \frac{\mu_n}{\|\mathbf{x}_n\|^2} e_n \mathbf{x}_n,$$

$$e_n = y_n - \mathbf{x}_n^T \boldsymbol{\theta}_{n-1},$$

which is the normalized LMS, introduced in Section 5.6.1.

- Closely related to the APSM algorithmic family are the *set-membership* algorithms, for example, [29, 32–34, 60]. This family can be seen as a special case of the APSM philosophy, where only special types of convex sets are used, for example, hyperslabs. Also, at each iteration step, a single projection is performed onto the set associated with the most recent pair of observations. For example, in [34, 94] the update recursion of a set-membership APA is given by

$$\boldsymbol{\theta}_n = \begin{cases} \boldsymbol{\theta}_{n-1} + X_n(X_n^T X_n)^{-1}(e_n - y_n), & \text{if } |e_n| > \epsilon, \\ \boldsymbol{\theta}_{n-1}, & \text{otherwise,} \end{cases} \quad (8.44)$$

where $X_n = [\mathbf{x}_n, \mathbf{x}_{n-1}, \dots, \mathbf{x}_{n-q+1}]$, $y_n = [y_n, y_{n-1}, \dots, y_{n-q+1}]^T$, $e_n = [e_n, e_{n-1}, \dots, e_{n-q+1}]^T$, with $e_n = y_n - \mathbf{x}_n^T \boldsymbol{\theta}_{n-1}$. The stochastic analysis of the set-membership APA [34] establishes a mean-square error (MSE) performance, and the analysis is carried out by adopting energy conservation arguments (Chapter 5).

Example 8.2. The goal of this example is to demonstrate the comparative convergence performance of the NLMS, the APA, the APSM, and the recursive least-squares RLS algorithms. The experiments were performed in two different noise settings, one for low and one for high noise levels, to demonstrate the sensitivity of the APA algorithm compared to the APSM. Data were generated according to our familiar model

$$y_n = \boldsymbol{\theta}_o^T \mathbf{x}_n + \eta_n.$$

The parameters $\boldsymbol{\theta}_o \in \mathbb{R}^{200}$ were randomly chosen from a $\mathcal{N}(0, 1)$ and then fixed. The input vectors were formed by a white noise sequence with samples i.i.d. drawn from a $\mathcal{N}(0, 1)$.

In the first experiment, the white noise sequence was chosen to have variance $\sigma^2 = 0.01$. The parameters for the three algorithms were chosen as $\mu = 1.2$ and $\delta = 0.001$ for the NLMS, $q = 30$, $\mu = 0.2$, and $\delta = 0.001$ for the APA and $\epsilon = \sqrt{2}\sigma$, $q = 30$, $\mu_n = 0.5 * M_n$ for the APSM. These parameters lead the algorithms to settle at the same error floor. Figure 8.25 shows the obtained squared error, averaged over 100 realizations, in dBs ($10 \log_{10}(e_n^2)$). For comparison, the RLS convergence curve is given for $\beta = 1$, which converges faster and at the same time settles at a lower error floor. If β is modified to a smaller value so that the RLS settles at the same error floor as the other algorithms, then its convergence gets even faster. However, this improved performance of the RLS is achieved at higher complexity, which becomes a problem for large values of l . Observe the faster convergence achieved by the APA and APSM, compared to the NLMS.

For the high-level noise, the corresponding variance was increased to 0.3. The obtained MSE curves are shown in Figure 8.26. Observe that now, the APA shows an inferior performance compared to APSM in spite of its higher complexity, due to the involved matrix inversion.

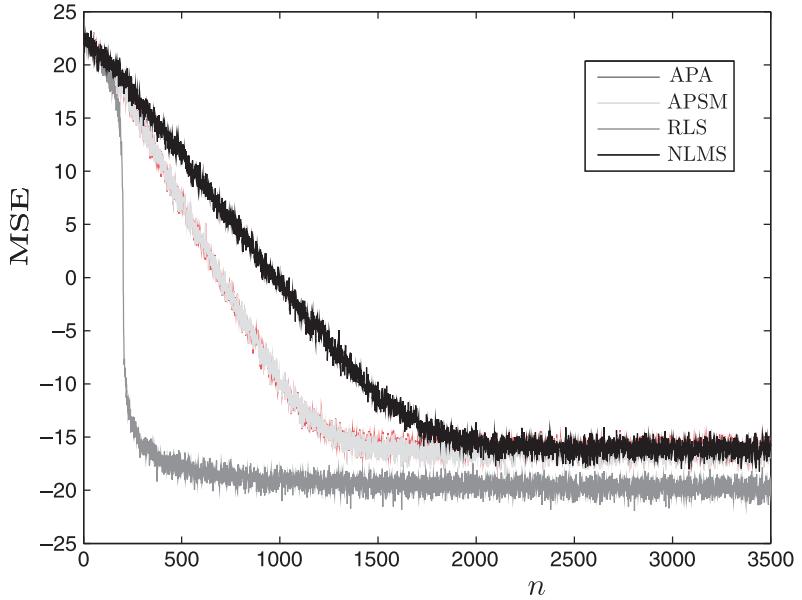


FIGURE 8.25

Mean-square error in dBs as a function of iterations. The data reuse ($q = 30$), associated with the APA and APSM, offer a significant improvement in convergence speed compared to the NLMS. The curves for the APA and APSM almost coincide in this low noise scenario.

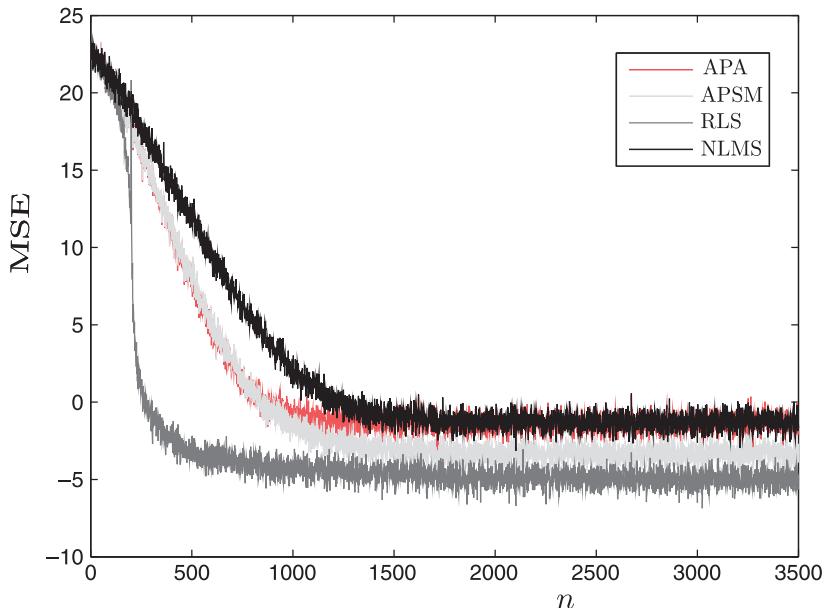


FIGURE 8.26

Mean-square error in dBs as a function of iterations for a high noise scenario. Compared to Figure 8.25, all curves settled at higher noise levels. Moreover, notice that now the APA settles at higher error floor than the corresponding APSM algorithm, for the same convergence rate.

8.8 CONSTRAINED LEARNING

Learning under a set of constraints is of significant importance in signal processing and machine learning, in general. We have already discussed a number of such learning tasks. Beamforming, discussed in [Chapters 5 and 4](#), is a typical one. In [Chapter 3](#), while introducing the concept of overfitting, we discussed the notion of regularization, which is another form of constraining the norm of the unknown parameter vector. In some other cases, we have available a priori information concerning the unknown parameters; this extra information can be given in the form of a set of constraints.

For example, if one is interested in obtaining estimates of the pixels in an image, then the values must be nonnegative. More recently, the unknown parameter vector may be known to be sparse; that is, only a few of its components are nonzero. In this case, constraining the respective ℓ_1 norm can significantly improve the accuracy as well as the convergence speed of an iterative scheme toward the solution. Schemes that explicitly take into consideration the underlying sparsity are known as *sparsity-promoting* algorithms, and they will be considered in more detail in [Chapter 10](#).

Algorithms that spring from the POCS theory are particularly suited to treat constraints in an elegant, robust, and rather straightforward way. Note that the goal of each constraint is to define a region in the solution space, where the required estimate is “forced” to lie. For the rest of this section, we will assume that the required estimate must satisfy M constraints, each one defining a *convex* set of points, $C_m, m = 1, 2, \dots, M$. Moreover,

$$\bigcap_{m=1}^M C_m \neq \emptyset,$$

which means that the constraints are consistent (there are also methods where this condition can be relaxed). Then, it can be shown that the mapping, T , defined as

$$T := P_{C_M} \dots P_{C_1},$$

is a *strongly attracting nonexpansive* mapping, [\(8.25\)](#), for example, [\[6, 18\]](#). Note that the same holds true if instead of the concatenation of the projection operators, one could convexly combine them.

In the presence of a set of constraints, the only difference in the APSM in [Algorithm 8.2](#) is that the update recursion [\(8.39\)](#) is now replaced by

$$\theta_n = T \left(\theta_{n-1} + \mu_n \left(\sum_{k=n-q+1}^n \omega_k P_{S_{\epsilon,k}}(\theta_{n-1}) - \theta_{n-1} \right) \right). \quad (8.45)$$

In other words, for M constraints, M extra projection operations have to be performed. The same applies to [\(8.38\)](#) with the difference being in the summation term in the brackets.

Remarks 8.6.

- The constrained form of the APSM has been successfully applied in the beamforming task, and in particular in treating nontrivial constraints, as it is required in the robust beamforming case [\[74, 77, 98, 99\]](#). The constrained APSM has also been efficiently used for sparsity-aware learning, for example, [\[46, 80\]](#) (see also [Chapter 10](#)). A more detailed review of related techniques is presented in [\[84\]](#).

8.9 THE DISTRIBUTED APSM

Distributed algorithms were discussed in [Chapter 5](#). In [Section 5.13.2](#), two versions of the diffusion LMS were introduced, namely the adapt-then-combine and the combine-then-adapt schemes. Diffusion versions of the APSM algorithm have also appeared in both configurations [20, 22]. For the APSM case, both schemes result in very similar performance.

Following the discussion in [Section 5.13.2](#), let the most recently received data pair at node $k = 1, 2, \dots, K$, be $(y_k(n), \mathbf{x}_k(n)) \in \mathbb{R} \times \mathbb{R}^l$. For the regression task, a corresponding hyperslab is constructed, that is,

$$S_{\epsilon,n}^{(k)} = \left\{ \boldsymbol{\theta} : |y_k(n) - \mathbf{x}_k^T(n)\boldsymbol{\theta}| \leq \epsilon_k \right\}.$$

The goal is the computation of a point that lies in the intersection of all these sets, for $n = 1, 2, \dots$. Following similar arguments as those employed for the diffusion LMS, the combine-then-adapt version of the APSM, given in [Algorithm 8.3](#), is obtained.

Algorithm 8.3 (The combine-then-adapt diffusion APSM).

- Initialization
 - **For** $k = 1, 2, \dots, K$, **Do**
 - $\boldsymbol{\theta}_k(0) = \mathbf{0} \in \mathbb{R}^l$; or any other value.
 - **End For**
 - Select A : $A^T \mathbf{1} = \mathbf{1}$
 - Select q ; The number of property sets to be processed at each time instant.
- **For** $n = 1, 2, \dots, q-1$, **Do**; Initial period, that is, $n < q$.
 - **For** $k = 1, 2, \dots, K$, **Do**
 - $\boldsymbol{\psi}_k(n-1) = \sum_{m \in \mathcal{N}_k} a_{mk} \boldsymbol{\theta}_m(n-1)$; \mathcal{N}_k the neighborhood of node k .
 - **End For**
 - **For** $k = 1, 2, \dots, K$, **Do**
 - Choose $\omega_1, \dots, \omega_n$: $\sum_{j=1}^n \omega_j = 1$, $\omega_j > 0$
 - Select $\mu_k(n) \in (0, 2M_k(n))$.

$$\boldsymbol{\theta}_k(n) = \boldsymbol{\psi}_k(n-1) + \mu_k(n) \left(\sum_{j=1}^n \omega_j P_{S_{\epsilon,j}^{(k)}}(\boldsymbol{\psi}_k(n-1)) - \boldsymbol{\psi}_k(n-1) \right)$$

- **End For**
- **For** $n = q, q+1, \dots$, **Do**
 - **For** $k = 1, 2, \dots, K$, **Do**
 - $\boldsymbol{\psi}_k(n-1) = \sum_{m \in \mathcal{N}_k} a_{mk} \boldsymbol{\theta}_m(n-1)$
 - **End For**
 - **For** $k = 1, 2, \dots, K$, **Do**
 - Choose $\omega_n, \dots, \omega_{n-q+1}$: $\sum_{j=n-q+1}^n \omega_j = 1$, $\omega_j > 0$.
 - Select $\mu_k(n) \in (0, 2M_k(n))$.

$$\boldsymbol{\theta}_k(n) = \boldsymbol{\psi}_k(n-1) + \mu_k(n) \left(\sum_{j=n-q+1}^n \omega_j P_{S_{\epsilon,j}^{(k)}}(\boldsymbol{\psi}_k(n-1)) - \boldsymbol{\psi}_k(n-1) \right)$$

- **End For**
- **End For**

The interval $M_{k,n}$ is defined as

$$M_k(n) = \sum_{j=n-q+1}^n \frac{\omega_j \|P_{S_{\epsilon,j}^{(k)}}(\psi_k(n-1)) - \psi_k(n-1)\|^2}{\|\sum_{j=n-q+1}^n \omega_j P_{S_{\epsilon,j}^{(k)}}(\psi_k(n-1)) - \psi_k(n-1)\|^2},$$

and similarly for the initial period.

Remarks 8.7.

- An important theoretical property of the APSM-based diffusion algorithms is that they enjoy *asymptotic consensus*. In other words, the nodes converge, asymptotically, to the *same* estimate. This asymptotic consensus is not in the mean, as it is the case with the diffusion LMS. This is interesting, since no explicit consensus constraints are employed.
- In [22], an extra projection step is used after the combination and prior to the adaptation step. The goal of this extra step is to “harmonize” the local information, which comprises the input/output measurements, with the information coming from the neighborhood, that is, the estimates obtained from the neighboring nodes. This speeds up convergence, at the cost of only one extra projection.
- A scenario in which some of the nodes are damaged and the associated observations are very noisy is also treated in [22]. To deal with such a scenario, instead of the hyperslab, the APSM algorithm is rephrased around the Huber loss function, developed in the context of robust statistics, to deal with cases where outliers are present (see also [Chapter 11](#)).

Example 8.3. The goal of this example is to demonstrate the comparative performance of the diffusion LMS and APSM. A network of $K = 10$ nodes is considered, and there are 32 connections among the nodes. In each node, data are generated according to a regression model, using the same vector $\theta_o \in \mathbb{R}^{60}$. The latter was randomly generated via a normal $\mathcal{N}(0, 1)$. The input vectors were i.i.d. generated according to the normal $\mathcal{N}(0, 1)$. The noise level at each node varied between 20 and 25 dBs. The parameters for the algorithms were chosen for optimized performance (after experimentation) and for similar convergence rate. For the LMS, $\mu = 0.035$ and for the APSM $\epsilon = \sqrt{2}\sigma$, $q = 20$, $\mu_k(n) = 0.2M_k(n)$. The combination weights were chosen according to the Metropolis rule and the data combination matrix was the identity one (no observations are exchanged). [Figure 8.27](#) shows the benefits of the data reuse offered by the APSM. The curves show the mean-square deviation ($\text{MSD} = \frac{1}{K} \sum_{k=1}^K \|\theta_k(n) - \theta_o\|^2$) as a function of the number of iterations.

8.10 OPTIMIZING NONSMOOTH CONVEX COST FUNCTIONS

Estimating parameters via the use of convex loss functions in the presence of a set of constraints is an established and well-researched field in optimization, with numerous applications in a wide range of disciplines. The mainstream of the methods follow either the Lagrange multipliers’ philosophy [10, 14] or the rationale behind the so-called *interior point* methods [14, 85]. In this section, we will focus on an alternative path and consider iterative schemes, which can be considered as the generalization of the gradient descent method, discussed in [Chapter 5](#). The reason is that such techniques give rise to variants that scale well with the dimensionality and have inspired a number of algorithms, which have

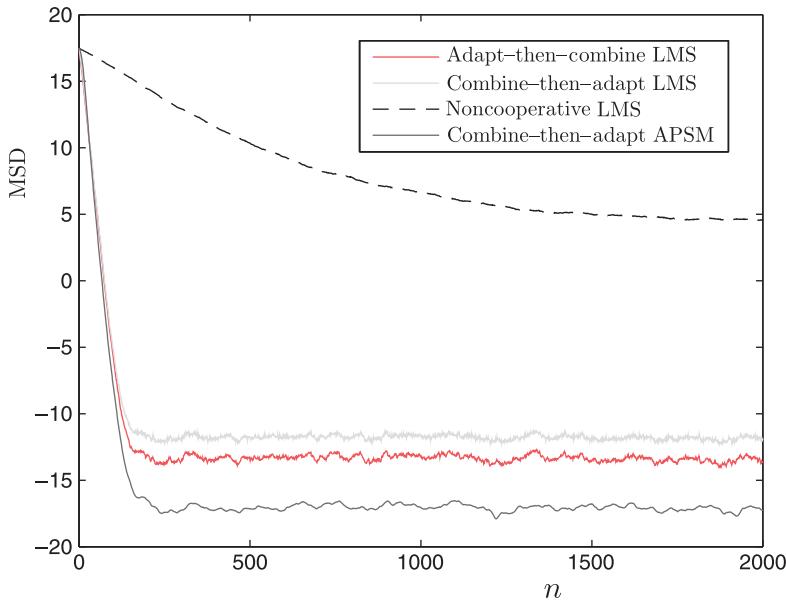


FIGURE 8.27

The MSD as a function of the number of iterations. The improved performance due to the data reuse offered by the diffusion APSM is readily observed. Moreover, observe the significant performance improvement offered by all cooperation schemes, compared to the noncooperative LMS; for the latter, only one node is used.

been suggested for online learning within the machine learning and signal processing communities. Later on, we will move to more advanced techniques that build on the so-called *operator/mapping* and *fixed point* theoretic framework.

Although the stage of our discussion will be that of Euclidean spaces, \mathbb{R}^l , everything that will be said can be generalized to infinite dimensional Hilbert spaces; we will consider such cases in Chapter 11.

8.10.1 SUBGRADIENTS AND SUBDIFFERENTIALS

We have already met the first order convexity condition in (8.3) and it was shown that this is a sufficient and necessary condition for convexity, provided, of course, that the gradient exists. The condition basically states that the graph of the convex function lies above the hyperplanes, which are tangent at any point, $(\mathbf{x}, f(\mathbf{x}))$, that lies on this graph.

Let us now move a step forward and assume a function

$$f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R},$$

to be convex, continuous but *nonsmooth*. This means that there are points where the gradient is not defined. Our goal now becomes that of generalizing the notion of gradient, for the case of convex functions.

Definition 8.7. A vector $\mathbf{g} \in \mathbb{R}^l$ is said to be the *subgradient* of a convex function, f , at a point, $\mathbf{x} \in \mathcal{X}$, if the following is true

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}), \quad \forall \mathbf{y} \in \mathcal{X} : \quad \text{Subgradient.} \quad (8.46)$$

It turns out that this vector is *not* unique. All the subgradients of a (convex) function at a point comprise a set.

Definition 8.8. The *subdifferential* of a convex function, f , at $\mathbf{x} \in \mathcal{X}$, denoted as $\partial f(\mathbf{x})$, is defined as the set

$$\partial f(\mathbf{x}) := \left\{ \mathbf{g} \in \mathbb{R}^l : f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}), \forall \mathbf{y} \in \mathcal{X} \right\} : \quad \text{Subdifferential.} \quad (8.47)$$

If f is differentiable at a point \mathbf{x} , then $\partial f(\mathbf{x})$ becomes a singleton, that is,

$$\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}.$$

Note that if $f(\mathbf{x})$ is convex, then the set $\partial f(\mathbf{x})$ is *nonempty and convex*. Moreover, $f(\mathbf{x})$ is differentiable at a point, \mathbf{x} , if and only if it has a unique subgradient [10]. From now on, we will denote a subgradient of f at a point, \mathbf{x} , as $f'(\mathbf{x})$.

Figure 8.28 gives a geometric interpretation of the notion of the subgradient. Each one of the subgradients at the point \mathbf{x}_0 defines a hyperplane that *supports* the graph of f . At \mathbf{x}_0 , there is an infinity of subgradients, which comprise the subdifferential (set) at \mathbf{x}_0 . At \mathbf{x}_1 , the function is differentiable and there is a *unique* subgradient that coincides with the gradient at \mathbf{x}_1 .

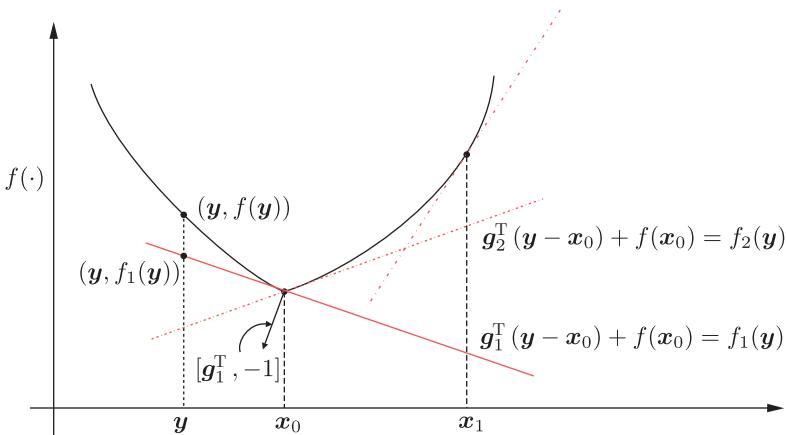


FIGURE 8.28

At \mathbf{x}_0 , there is an infinity of subgradients, each one defining a hyperplane in the extended $(\mathbf{x}, f(\mathbf{x}))$ space. All these hyperplanes pass through the point $(\mathbf{x}_0, f(\mathbf{x}_0))$ and support the graph of $f(\cdot)$. At the point \mathbf{x}_1 , there is a unique subgradient that coincides with the gradient and defines the unique tangent hyperplane at the respective point of the graph.

Example 8.4. Let $x \in \mathbb{R}$ and

$$f(x) = |x|.$$

Then, show that

$$\partial f(x) = \begin{cases} \text{sgn}(x), & \text{if } x \neq 0, \\ g \in [-1, 1], & \text{if } x = 0, \end{cases}$$

where $\text{sgn}(\cdot)$ is the sign function, being equal to 1 if its argument is positive and -1 if the argument is negative.

Indeed, if $x > 0$, then

$$g = \frac{dx}{dx} = 1,$$

and similarly $g = -1$, if $x < 0$. For $x = 0$, any $g \in [-1, 1]$ satisfies

$$g(y - 0) + 0 = gy \leq |y|,$$

and it is a subgradient. This is illustrated in Figure 8.29.

Lemma 8.2. Given a convex function $f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R}$, a point $\mathbf{x}_* \in \mathcal{X}$ is a minimizer of f , if and only if the zero vector belongs to its subdifferential set, that is,

$$\mathbf{0} \in \partial f(\mathbf{x}_*) : \quad \text{Condition for a Minimizer.} \quad (8.48)$$

Proof. The proof is straightforward from the definition of a subgradient. Indeed, assume that $\mathbf{0} \in \partial f(\mathbf{x}_*)$. Then, the following is valid

$$f(\mathbf{y}) \geq f(\mathbf{x}_*) + \mathbf{0}^T(\mathbf{y} - \mathbf{x}_*), \quad \forall \mathbf{y} \in \mathcal{X}$$

and \mathbf{x}_* is a minimizer. If now \mathbf{x}_* is a minimizer, then we have that

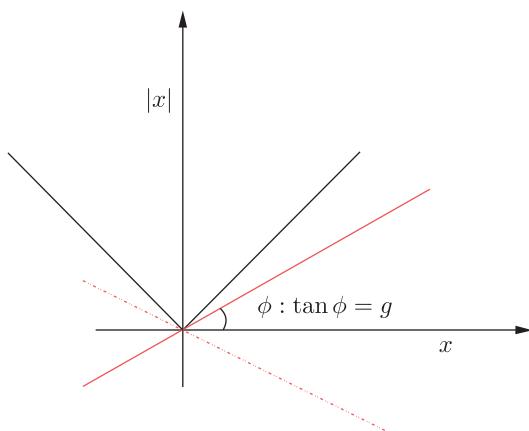


FIGURE 8.29

All lines with $\in [-1, 1]$ comprise the subdifferential at $x = 0$.

$$f(\mathbf{y}) \geq f(\mathbf{x}_*) = f(\mathbf{x}_*) + \mathbf{0}^T(\mathbf{y} - \mathbf{x}_*),$$

hence $\mathbf{0} \in \partial f(\mathbf{x}_*)$. □

Example 8.5. Let the metric *distance* function

$$d_C(\mathbf{x}) := \min_{\mathbf{y} \in C} \|\mathbf{x} - \mathbf{y}\|.$$

This is the Euclidean distance of a point from its projection on a closed convex set, C , as defined in [Section 8.3](#). Then show ([Problem 8.24](#)) that the subdifferential is given by

$$\partial d_C(\mathbf{x}) = \begin{cases} \frac{\mathbf{x} - P_C(\mathbf{x})}{\|\mathbf{x} - P_C(\mathbf{x})\|}, & \mathbf{x} \notin C, \\ N_C(\mathbf{x}) \cap B[0, 1], & \mathbf{x} \in C, \end{cases} \quad (8.49)$$

where

$$N_C(\mathbf{x}) := \left\{ \mathbf{g} \in \mathbb{R}^l : \mathbf{g}^T(\mathbf{y} - \mathbf{x}) \leq 0, \forall \mathbf{y} \in C \right\},$$

and

$$B[0, 1] := \left\{ \mathbf{x} \in \mathbb{R}^l : \|\mathbf{x}\| \leq 1 \right\}.$$

Moreover, if \mathbf{x} is an *interior* point of C , then

$$\partial d_C(\mathbf{x}) = \{\mathbf{0}\}.$$

Observe that for all points $\mathbf{x} \notin C$ as well as for all interior points of C , the subgradient is a singleton, which means that $d_C(\mathbf{x})$ is differentiable. Recall that the function $d_C(\cdot)$ is nonnegative, convex, and continuous [[43](#)].

Note that (8.49) is also generalized to infinite dimensional Hilbert spaces.

8.10.2 MINIMIZING NONSMOOTH CONTINUOUS CONVEX LOSS FUNCTIONS: THE BATCH LEARNING CASE

Let J be a cost function⁷

$$J : \mathbb{R}^l \mapsto [0, +\infty),$$

and C a closed convex set, $C \subseteq \mathbb{R}^l$. Our task is to compute a minimizer with respect to an unknown parameter vector, that is,

$$\begin{aligned} \theta_* &= \arg \min_{\theta} J(\theta), \\ \text{s.t.} \quad \theta &\in C, \end{aligned} \quad (8.50)$$

and we will assume that the set of solutions is *nonempty*. J is assumed to be convex, continuous, but not necessarily differentiable at all points. We have already seen examples of such loss function, such as the ϵ -insensitive linear function in (8.33) and the hinge one (8.37). The ℓ_1 -norm function is another example, and it will be treated in [Chapters 9 and 10](#).

⁷ Recall what we have already said, that all the methods to be reported can be extended to general Hilbert spaces, \mathbb{H} .

The subgradient method

Our starting point is the simplest of the cases, where $C = \mathbb{R}^l$; that is, the minimizing task is unconstrained. The first thought that comes into mind is to consider the generalization of the gradient descent method, which was introduced in [Chapter 5](#), and replace the gradient by the subgradient operation. The resulting scheme is known as the *subgradient algorithm* [71, 72].

Starting from an arbitrary estimate, $\theta^{(0)} \in \mathbb{R}^l$, the update recursions become

$$\boxed{\theta^{(i)} = \theta^{(i-1)} - \mu_i J'(\theta^{(i-1)}) : \quad \text{Subgradient Algorithm},} \quad (8.51)$$

where $J'(\cdot)$ denotes *any* subgradient of the cost function, and μ_i is a step-size sequence judiciously chosen so that convergence is guaranteed. In spite of the similarity in the appearance with our familiar gradient descent scheme, there are some major differences. The reader may have noticed that the new algorithm was not called subgradient “descent.” This is because the update in (8.51) is not necessarily performed in the descent direction. Thus, during the operation of the algorithm, the value of the cost function may increase. Recall that in the gradient descent methods, the value of the cost function is guaranteed to decrease with each iteration step, which also led to a linear convergence rate, as we have pointed out in [Chapter 5](#).

In contrast here, concerning the subgradient method, such comments cannot be stated. To establish convergence, a different route has to be adopted. To this end, let us define

$$J_*^{(i)} := \min \left\{ J(\theta^{(i)}), J(\theta^{(i-1)}), \dots, J(\theta^{(0)}) \right\}, \quad (8.52)$$

which can also be recursively obtained by

$$J_*^{(i)} = \min \left\{ J_*^{(i-1)}, J(\theta^{(i)}) \right\}.$$

Then the following holds true.

Proposition 8.3. *Let J be a convex cost function. Assume that the subgradients at all points are bounded, that is,*

$$\|J'(\mathbf{x})\| \leq G, \quad \forall \mathbf{x} \in \mathbb{R}^l.$$

Let us also assume that the step-size sequence be a diminishing one, such as

$$\sum_{i=1}^{\infty} \mu_i = \infty, \quad \sum_{i=1}^{\infty} \mu_i^2 < \infty.$$

Then

$$\lim_{i \rightarrow \infty} J_*^{(i)} = J(\theta_*),$$

where θ_ is a minimizer, assuming that the set of minimizers is not empty.*

Proof. We have that

$$\begin{aligned} \|\theta^{(i)} - \theta_*\|^2 &= \|\theta^{(i-1)} - \mu_i J'(\theta^{(i-1)}) - \theta_*\|^2 \\ &= \|\theta^{(i-1)} - \theta_*\|^2 - 2\mu_i J'^T(\theta^{(i-1)})(\theta^{(i-1)} - \theta_*) \\ &\quad + \mu_i^2 \|J'(\theta^{(i-1)})\|^2. \end{aligned} \quad (8.53)$$

By the definition of the subgradient, we have

$$J(\boldsymbol{\theta}_*) - J(\boldsymbol{\theta}^{(i-1)}) \geq \mathbf{J}'^T(\boldsymbol{\theta}^{(i-1)})(\boldsymbol{\theta}_* - \boldsymbol{\theta}^{(i-1)}). \quad (8.54)$$

Plugging (8.54) in (8.53) and after some algebraic manipulations, by applying the resulting inequality recursively (Problem 8.25), we finally obtain that

$$J_*^{(i)} - J(\boldsymbol{\theta}_*) \leq \frac{||\boldsymbol{\theta}^{(0)} - \boldsymbol{\theta}_*||^2}{2 \sum_{k=1}^i \mu_k} + \frac{\sum_{k=1}^i \mu_k^2}{2 \sum_{k=1}^i \mu_k} G^2. \quad (8.55)$$

Leaving i to grow to infinity and taking into account the assumptions, the claim is proved. \square

There are a number of variants of this proof. Also, other choices for the diminishing sequence can also guarantee convergence, such as $\mu_i = 1/\sqrt{i}$. Moreover, in certain cases, some of the assumptions may be relaxed. Note that the assumption of the subgradient being bounded is guaranteed, if J is γ -Lipschitz continuous (Problem 8.26), that is, there is $\gamma > 0$, such as

$$|J(\mathbf{y}) - J(\mathbf{x})| \leq \gamma \|\mathbf{y} - \mathbf{x}\|, \quad \forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^l.$$

Interpreting the proposition from a slightly different angle, we can say that the algorithm generates a *subsequence* of estimates, $\boldsymbol{\theta}_{i_*}$, which corresponds to the values of J_* , shown as, $J(\boldsymbol{\theta}_{i_*}) \leq J(\boldsymbol{\theta}_i)$, $i \leq i_*$, which converges to $\boldsymbol{\theta}_*$. The *best possible* convergence rate that may be achieved is of the order of $\mathcal{O}(\frac{1}{\sqrt{i}})$, if one optimizes the bound in (8.55), with respect to μ_k , [61], which can be obtained if $\mu_i = \frac{c}{\sqrt{i}}$, where c is a constant. In any case, it is readily noticed that the convergence speed of such methods is rather slow. Yet due to their computational simplicity, they are still in use, especially in cases where the number of data samples is large. The interested reader can obtain more on the subgradient method from [9, 72].

Example 8.6 (The perceptron algorithm). Recall the hinge loss function with $\rho = 0$, defined in (8.37),

$$\mathcal{L}(y, \boldsymbol{\theta}^T \mathbf{x}) = \max(0, -y\boldsymbol{\theta}^T \mathbf{x}).$$

In a two-class classification task, we are given a set of training samples, $(y_n, \mathbf{x}_n) \in \{-1, +1\} \times \mathbb{R}^{l+1}$, $n = 1, 2, \dots, N$, and the goal is to compute a linear classifier to minimize the empirical loss function

$$J(\boldsymbol{\theta}) = \sum_{n=1}^N \mathcal{L}(y_n, \boldsymbol{\theta}^T \mathbf{x}_n). \quad (8.56)$$

We will assume the classes to be *linearly separable*, which guarantees that there is a solution; that is, there exists a hyperplane that classifies correctly all data points. Obviously, such a hyperplane will score a zero for the cost function in (8.56). We have assumed that the dimension of our input data space has been increased by one, to account for the bias term for hyperplanes not crossing the origin.

The subdifferential of the hinge loss function is easily checked out to be (e.g., use geometric arguments, which relate a subgradient with a support hyperplane of the respective function graph),

$$\partial \mathcal{L}(y_n, \boldsymbol{\theta}^T \mathbf{x}_n) = \begin{cases} 0, & y_n \boldsymbol{\theta}^T \mathbf{x}_n > 0, \\ -y_n \mathbf{x}_n, & y_n \boldsymbol{\theta}^T \mathbf{x}_n < 0, \\ \mathbf{g} \in [-y_n \mathbf{x}_n, 0], & y_n \boldsymbol{\theta}^T \mathbf{x}_n = 0. \end{cases} \quad (8.57)$$

We choose to work with the following subgradient

$$\mathcal{L}'(y_n, \boldsymbol{\theta}^T \mathbf{x}_n) = -y_n \mathbf{x}_n \chi_{(-\infty, 0]}(y_n \boldsymbol{\theta}^T \mathbf{x}_n), \quad (8.58)$$

where $\chi_A(\tau)$ is the characteristic function, defined as

$$\chi_A(\tau) = \begin{cases} 1, & \tau \in A, \\ 0, & \tau \notin A. \end{cases} \quad (8.59)$$

The subgradient algorithm now becomes,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \sum_{n=1}^N y_n \mathbf{x}_n \chi_{(-\infty, 0]}(y_n \boldsymbol{\theta}^{(i-1)T} \mathbf{x}_n). \quad (8.60)$$

This is the celebrated *perceptron algorithm*, which we are going to see in more detail in [Chapter 18](#). Basically, what the algorithm in (8.60) says is the following. Starting from an arbitrary $\boldsymbol{\theta}^{(0)}$, test all training vectors with $\boldsymbol{\theta}^{(i-1)}$. Select all those vectors that fail to predict the correct class (for the correct class, $y_n \boldsymbol{\theta}^{(i-1)T} \mathbf{x}_n > 0$), and update the current estimate toward the direction of the weighted (by the corresponding label) average of the *misclassified* patterns. It turns out that the algorithm converges in a *finite* number of steps, even if the step-size sequence is not a diminishing one. This is what it was said before, that in certain cases, convergence of the subgradient algorithm is guaranteed even if some of the assumptions in [Proposition 8.3](#) do not hold.

The generic projected subgradient scheme

The generic scheme on which a number of variants draw their origin is summarized as follows. Select $\boldsymbol{\theta}^{(0)} \in \mathbb{R}^l$, arbitrarily. Then the iterative scheme

$$\boxed{\boldsymbol{\theta}^{(i)} = P_C(\boldsymbol{\theta}^{(i-1)} - \mu_i J'(\boldsymbol{\theta}^{(i-1)})) : \quad \text{GPS Scheme},} \quad (8.61)$$

where $J'(\cdot)$ denotes a respective subgradient and P_C is the projection operator onto C , converges (converges weakly in the more general case) to a solution of the constrained task in (8.50). The sequence of nonnegative real numbers, μ_i , is judiciously selected. It is readily seen that this scheme is a generalization of the gradient descent scheme, discussed in [Chapter 5](#), if we set $C = \mathbb{R}^l$ and J is differentiable.

The projected gradient method (PGM)

This method is a special case of (8.61), if J is *smooth* and we set $\mu_i = \mu$. That is,

$$\boxed{\boldsymbol{\theta}^{(i)} = P_C(\boldsymbol{\theta}^{(i-1)} - \mu \nabla J(\boldsymbol{\theta}^{(i-1)})) : \quad \text{PGM Scheme}.} \quad (8.62)$$

It turns out that if the gradient is γ -Lipschitz, that is,

$$\|\nabla J(\boldsymbol{\theta}) - \nabla J(\mathbf{h})\| \leq \gamma \|\boldsymbol{\theta} - \mathbf{h}\|, \quad \gamma > 0, \quad \forall \boldsymbol{\theta}, \mathbf{h} \in \mathbb{R}^l,$$

and

$$\mu \in \left(0, \frac{2}{\gamma}\right),$$

then starting from an arbitrary point $\boldsymbol{\theta}^{(0)}$, the sequence in (8.62) converges (weakly in a general Hilbert space) to a solution of (8.50) [40, 51].

Example 8.7. *Projected Landweber Method:*

Let our optimization task be

$$\begin{aligned} & \text{minimize } \frac{1}{2} \|y - X\theta\|^2, \\ & \text{subject to } \theta \in C. \end{aligned}$$

where $X \in \mathbb{R}^{m \times l}$, $y \in \mathbb{R}^m$. Expanding and taking the gradient we get,

$$\begin{aligned} J(\theta) &= \frac{1}{2} \theta^T X^T X \theta - y^T X \theta + \frac{1}{2} y^T y, \\ \nabla J(\theta) &= X^T X \theta - X^T y. \end{aligned}$$

First we check that $\nabla J(\theta)$ is γ -Lipschitz. To this end, we have

$$\|X^T X(\theta - h)\| \leq \|X^T X\| \|\theta - h\| \leq \lambda_{\max} \|\theta - h\|,$$

where the spectral norm of a matrix has been used (Section 6.4) and λ_{\max} denotes the maximum eigenvalue $X^T X$. Thus, if

$$\mu \in \left(0, \frac{2}{\lambda_{\max}}\right)$$

the corresponding iterations in (8.62) converge to a solution of (8.50). The scheme has been used in the context of compressed sensing where (as we will see in Chapter 10) the task of interest is

$$\begin{aligned} & \text{minimize } \frac{1}{2} \|y - X\theta\|^2, \\ & \text{subject to } \|\theta\|_1 \leq \rho. \end{aligned}$$

Then, it turns out that projecting on the ℓ_1 -ball (corresponding to C) is equivalent to a soft thresholding operation⁸ [35]. A variant occurs, if a projection on a weighted ℓ_1 ball is used, to speed up convergence (Chapter 10). Projection on a weighted ℓ_1 ball has been developed in [46], via fully geometric arguments, and it also results in soft-thresholding operations.

Projected subgradient method

Starting from an arbitrary point $\theta^{(0)}$, then for the following recursion [2, 54],

$$\theta^{(i)} = P_C \left(\theta^{(i-1)} - \frac{\mu_i}{\max\{1, \|J'(\theta^{(i-1)})\|\}} J'(\theta^{(i-1)}) \right) : \quad \text{PSMa}, \quad (8.63)$$

- either a solution of (8.50) is achieved in a finite number of steps,
- or the iterations converge (weakly in the general case) to a point in the set of solutions of (8.50),

provided that

$$\sum_{i=1}^{\infty} \mu_i = \infty, \quad \sum_{i=1}^{\infty} \mu_i^2 < \infty.$$

Another version of the projected subgradient algorithm was presented in [67]. Let $J_* = \min_{\theta} J(\theta)$ be the minimum (strictly speaking the infimum) of a cost function, whose set of minimizers is assumed to be nonempty. Then, the following iterative algorithm,

⁸ See Chapter 10 and Example 8.10.

$$\boldsymbol{\theta}^{(i)} = \begin{cases} P_C \left(\boldsymbol{\theta}^{(i-1)} - \mu_i \frac{J(\boldsymbol{\theta}^{(i-1)}) - J_*}{\|J'(\boldsymbol{\theta}^{(i-1)})\|^2} J'(\boldsymbol{\theta}^{(i-1)}) \right), & \text{if } J'(\boldsymbol{\theta}^{(i-1)}) \neq \mathbf{0}, \\ P_C(\boldsymbol{\theta}^{(i-1)}), & \text{if } J'(\boldsymbol{\theta}^{(i-1)}) = \mathbf{0}, \end{cases} \quad \text{PSMb} \quad (8.64)$$

converges (weakly in infinite dimensional spaces) for $\mu_i \in (0, 2)$ and under some general conditions and assuming that the subgradient is bounded. The proof is a bit technical and the interested reader can obtain it from, for example, [67, 79].

Needless to say that, besides the previously reported major schemes discussed, there is a number of variants; for a related review see [79].

8.10.3 ONLINE LEARNING FOR CONVEX OPTIMIZATION

Online learning in the framework of the squared error loss function has been the focus in Chapters 5 and 6. One of the reasons that online learning was introduced was to give the potential to the algorithm to track time variations in the underlying statistics. Another reason was to cope with the unknown statistics when the cost function involved expectations, in the context of the stochastic approximation theory. Moreover, online algorithms are of particular interest when the number of the available training data as well as the dimensionality of the input space become very large, compared to the load that today's storage, processing, and networking devices can cope with. Exchanging information has now become cheap and databases have been populated with a massive number of data. This has rendered batch processing techniques, for learning tasks with huge datasets, impractical. Online algorithms that process one data point at a time have now become an indispensable algorithmic tool.

Recall from Section 3.14 that the ultimate goal of a machine learning task, given a loss function \mathcal{L} , is to minimize the expected loss/risk, which in the context of parametric modeling can be written as

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathbb{E} [\mathcal{L}(\mathbf{y}, f_{\boldsymbol{\theta}}(\mathbf{x}))] \\ &:= \mathbb{E} [\mathcal{L}(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})]. \end{aligned} \quad (8.65)$$

Instead, the corresponding empirical loss function is minimized, given a set of N training points,

$$J_N(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}(\boldsymbol{\theta}, y_n, \mathbf{x}_n). \quad (8.66)$$

In this context, the subgradient scheme would take the form

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \frac{\mu_i}{N} \sum_{n=1}^N \mathcal{L}'_n(\boldsymbol{\theta}^{(i-1)}),$$

where for notational simplicity we used

$$\mathcal{L}_n(\boldsymbol{\theta}) := \mathcal{L}(\boldsymbol{\theta}, y_n, \mathbf{x}_n). \quad (8.67)$$

Thus, at each iteration, one has to compute N subgradient values, which for large values of N is computationally cumbersome. One way out is to adopt stochastic approximation arguments, as explained in Chapter 5, and come with a corresponding online version,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \mathcal{L}'_n(\boldsymbol{\theta}_{n-1}), \quad (8.68)$$

where now the iteration index, i , coincides with the time index, n . There are two different ways to view (8.68). Either n takes values in the interval $[1, N]$, and one cycles periodically until convergence, or n is left to grow unbounded. The latter is very natural for very large values of N , and we focus on this scenario from now on. Moreover, such strategy can cope with slow time variations, if this is the case. Note that in the online formulation, at each time instant a *different* loss function is involved and our task becomes that of an *asymptotic* minimization. Furthermore, one has to study the asymptotic *convergence properties*, as well as the respected *convergence conditions*. Soon, we are going to introduce a relatively recent tool, for analyzing the performance of online algorithms, namely, the *regret analysis*.

It turns out that for each one of the optimization schemes discussed in Subsection 8.10.2, we can write its online version. Given the sequence of loss functions, \mathcal{L}_n , $n = 1, 2, \dots$, the online version of the generic projected subgradient scheme becomes

$$\boldsymbol{\theta}_n = P_C(\boldsymbol{\theta}_{n-1} - \mu_n \mathcal{L}'_n(\boldsymbol{\theta}_{n-1})), \quad n = 1, 2, 3, \dots \quad (8.69)$$

In a more general setting, the constraint-related convex sets can be left to be time varying, too; in other words, we can write C_n . For example, such schemes with time-varying constraints have been developed in the context of sparsity-aware learning, where in place of the ℓ_1 -ball, a weighted ℓ_1 -ball is being used [46]. This has a really drastic effect in speeding up the convergence of the algorithm, see Chapter 10.

Another example is the so-called *adaptive gradient* (ADAGRAD) algorithm [38]. The projection operator is defined in a more general context, in terms of the Mahalanobis distance, that is,

$$P_C^G(\mathbf{x}) = \min_{\mathbf{z} \in C} (\mathbf{x} - \mathbf{z})^T G(\mathbf{x} - \mathbf{z}), \quad \forall \mathbf{x} \in \mathbb{R}^I. \quad (8.70)$$

In place of G , the square root of the average outer product of the computed subgradients is used, that is,

$$G_n = \left(\frac{1}{n} \sum_{k=1}^n \mathbf{g}_k \mathbf{g}_k^T \right)^{1/2},$$

where, $\mathbf{g}_k = \mathcal{L}'_k(\boldsymbol{\theta}_{k-1})$ denotes the subgradient at time instant k . Also, the same matrix is used to weigh the gradient correction and the scheme has the form,

$$\boldsymbol{\theta}_n = P_C^{G_n}(\boldsymbol{\theta}_{n-1} - \mu_n G_n^{-1} \mathbf{g}_n).$$

The use of the (time-varying) weighting matrix accounts for the geometry of the data observed in earlier iterations, which leads to a more informative gradient-based learning. For the sake of computational savings, the structure of G_n is taken to be diagonal. Different algorithmic settings are discussed in [38], alongside the study of the converging properties of the algorithm.

Example 8.8. *The LMS Algorithm:* Let us assume that

$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{2} (y_n - \boldsymbol{\theta}^T \mathbf{x}_n)^2,$$

and also set $C = \mathbb{R}^I$ and $\mu_n = \mu$. Then (8.69) becomes our familiar LMS recursion,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} + \mu (y_n - \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n) \mathbf{x}_n,$$

whose convergence properties have been discussed in Chapter 5.

The PEGASOS algorithm

The *primal estimated subgradient solver for SVM* (PEGASOS) algorithm is an online scheme built around the hinge loss function *regularized* by the squared Euclidean norm of the parameters vector, [70]. From this point of view, it is an instance of the online version of the projected subgradient algorithm. This algorithm results if we set in (8.69),

$$\mathcal{L}_n(\boldsymbol{\theta}) = \max(0, 1 - y_n \boldsymbol{\theta}^T \mathbf{x}_n) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2, \quad (8.71)$$

where in this case, ρ in the hinge loss function has been set equal to one. The associated empirical cost function is

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^N \max(0, 1 - y_n \boldsymbol{\theta}^T \mathbf{x}_n) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2, \quad (8.72)$$

whose minimization results in the celebrated *support vector machine* (SVM). Note that the only differences with the perceptron algorithm is the presence of the regularizer and the nonzero value of ρ . These seemingly minor differences have important implications in practice, and we are going to say more on this in [Chapter 11](#), where nonlinear extensions treated in the more general context of Hilbert spaces will be considered.

The subgradient adopted by the PEGASOS is

$$\mathcal{L}'_n(\boldsymbol{\theta}) = \lambda \boldsymbol{\theta} - y_n \mathbf{x}_n \chi_{(-\infty, 0]}(y_n \boldsymbol{\theta}^T \mathbf{x}_n - 1). \quad (8.73)$$

The step-size is chosen as $\mu_n = \frac{1}{\lambda n}$. Furthermore in its more general formulation, at each iteration step, an (optional) projection on the $\frac{1}{\sqrt{\lambda}}$ length ℓ_2 ball, $B[\mathbf{0}, \frac{1}{\sqrt{\lambda}}]$, is performed. The update recursion then becomes,

$$\boldsymbol{\theta}_n = P_{B[\mathbf{0}, \frac{1}{\sqrt{\lambda}}]} \left((1 - \mu_n \lambda) \boldsymbol{\theta}_{n-1} + \mu_n y_n \mathbf{x}_n \chi_{(-\infty, 0]}(y_n \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n - 1) \right), \quad (8.74)$$

where $P_{B[\mathbf{0}, \frac{1}{\sqrt{\lambda}}]}$ is the projection on the respective ℓ_2 ball given in (8.14). In (8.74), note that the effect of the regularization is to smooth out the contribution of $\boldsymbol{\theta}_{n-1}$. A variant of the algorithm for fixed number of points, N , suggests to average out a number of m subgradient values in an index set, $A_n \subseteq \{1, 2, \dots, N\}$, such as $k \in A_n$ if $y_k \boldsymbol{\theta}_{n-1}^T \mathbf{x}_k < 1$. Different scenarios for the choice of the m indices can be employed, with the random one being a possibility. The scheme is summarized in [Algorithm 8.4](#).

Algorithm 8.4 (The PEGASOS algorithm).

- Initialization
 - Select $\boldsymbol{\theta}^{(0)}$; Usually set to zero.
 - Select λ
 - Select m ; Number of subgradient values to be averaged.
- **For** $n = 1, 2, \dots, N$, **Do**
 - Select $A_n \subseteq \{1, 2, \dots, N\}$: $|A_n| = m$, uniformly at random.
 - $\mu_n = \frac{1}{\lambda n}$
 - $\boldsymbol{\theta}_n = (1 - \mu_n \lambda) \boldsymbol{\theta}_{n-1} + \frac{\mu_n}{m} \sum_{k \in A_n} y_k \mathbf{x}_k$
 - $\boldsymbol{\theta}_n = \min \left(1, \frac{1}{\sqrt{\lambda} \|\boldsymbol{\theta}_n\|} \right) \boldsymbol{\theta}_n$; Optional.
- **End For**

Application of regret analysis arguments point out the required number of iterations for obtaining a solution of accuracy ϵ is $\mathcal{O}(1/\epsilon)$, when each iteration operates on a single training sample. The algorithm is very similar with the algorithms proposed in [44, 105]. The difference being in the choice of the step-size. We will come back to these algorithms in Chapter 11. There, we are going to see that the online learning in infinite dimensional spaces is more tricky. In [70], a number of comparative tests against well-established SVM algorithms have been performed using standard datasets. The main advantage of the algorithm is its computational simplicity, and it achieves comparable performance rates at lower computational costs.

Remarks 8.8.

- *The APSM revisited:* It can be seen that the APSM algorithm can be re-derived in a more general setting as an online version of the PSGb in (8.64), which also justifies its name. It suffices to consider as a loss function the weighted average of the distances of a point θ from the q most recently formed, by the data, property sets, Problem 8.29.

Moreover, such a derivation paves the way for employing the algorithm even if the projection onto the constraint set is not analytically available. It turns out that the mapping

$$T(\theta) = \theta - \mu \frac{\mathcal{L}_n(\theta)}{\|\mathcal{L}'_n(\theta)\|^2} \mathcal{L}'_n(\theta),$$

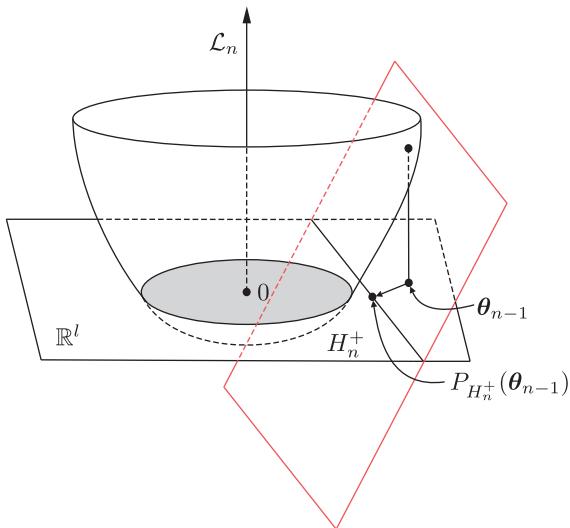
maps θ onto the hyperplane, H , which (in the extended space \mathbb{R}^{l+1}) is the intersection of \mathbb{R}^l with the hyperplane that is defined by the respective subgradient of the loss function at θ . This is a *separating* hyperplane, that separates θ from the zero level set of \mathcal{L}_n . Hence, after the mapping, θ is mapped to a point closer to this zero level set.

It can be shown that the algorithm monotonically converges into the intersection of all the zero level sets of \mathcal{L}_n , $n = 1, 2, \dots$, assuming that it is nonempty and the problem is feasible. This version of the APSM does not need the projection onto the property convex sets to be given analytically. In the case the loss functions are chosen to be the ϵ -insensitive loss or the hinge loss, the algorithm results in the APSM of Algorithm 8.2. Figure 8.30 shows the geometry of the mapping for a quadratic loss function; note that in this case, the zero level set is an ellipse and the projection is not analytically defined. The interested reader can obtain more on these issues, as well as on full convergence proofs, from [79, 84, 98, 99].

8.11 REGRET ANALYSIS

A major effort when dealing with iterative learning algorithms is dedicated to the issue of convergence; where the algorithm converges, under which conditions it converges and how fast it converges to its steady-state. A large part of Chapter 5 was focused on the convergence properties of the LMS. Furthermore, in the current chapter, when we discussed the various subgradient-based algorithms, convergence properties were also reported.

In general, analyzing the convergence properties of online algorithms tends to be quite a formidable task and classical approaches have to adopt a number of assumptions, sometimes rather strong. Typical assumptions refer to the statistical nature of the data (e.g., being i.i.d. or the noise being white), and/or that the true model, which generates the data, is assumed to be known, and/or that the algorithm has reached a region in the parameter's space that is close to a minimizer.

**FIGURE 8.30**

At every time instant, the subgradient of the loss function \mathcal{L}_n at θ_{n-1} , defines a hyperplane that intersects the input space. The intersection is a hyperplane, which separates θ_{n-1} from the zero level set of \mathcal{L}_n . The APSM performs a projection on the halfspace that contains the zero level set and brings the update closer to it. For the case of ϵ -insensitive or the hinge loss functions, the separating hyperplane is a support hyperplane and the projection coincides with the projection on the respective hyperslab or halfspace, respectively.

More recently, an alternative methodology has been developed which bypasses the need for such assumptions. The methodology evolves around the concept of *cumulative loss*, which has already been introduced in [Chapter 5, Section 5.5.2](#). The method is known as *regret analysis*, and its birth is due to developments in the interplay between game and learning theories; see, for example, [21].

Let us assume that the training samples, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots$, arrive sequentially and that an adopted online algorithm makes the corresponding predictions, \hat{y}_n . The quality of the prediction, for each time instant, is tested against a loss function, $\mathcal{L}(y_n, \hat{y}_n)$. The cumulative loss up to time N is given by

$$\mathcal{L}_{\text{cum}}(N) := \sum_{n=1}^N \mathcal{L}(y_n, \hat{y}_n). \quad (8.75)$$

Let f be a *fixed* predictor. Then the *regret* of the online algorithm relative to f , when running up to time instant N , is defined as

$$\text{Regret}_N(f) := \sum_{n=1}^N \mathcal{L}(y_n, \hat{y}_n) - \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) : \quad \text{Regret Relative to } f. \quad (8.76)$$

The name regret is inherited from the game theory and it means how “sorry” the algorithm or the learner (in the ML jargon) is, in retrospect, not to have followed the prediction of the fixed predictor, f . The predictor f is also known as the *hypothesis*. Also, if f is chosen from a set of functions, \mathcal{F} , this set is called the *hypothesis class*.

The regret relative to the family of functions, \mathcal{F} , when the algorithm runs over N time instants, is defined as

$$\text{Regret}_N(\mathcal{F}) := \max_{f \in \mathcal{F}} \text{Regret}_N(f). \quad (8.77)$$

In the context of regret analysis, the goal becomes that of designing an online learning rule so that the resulting regret with respect to an optimal fixed predictor to be small; that is, the regret associated with the learner should grow *sublinearly* (slower than linearly) with the number of iterations, N . Sublinear growth guarantees that the difference between the *average* loss suffered by the learner and the average loss of the optimal predictor will tend to zero asymptotically.

For the linear class of functions, we have

$$\hat{y}_n = \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n,$$

and the loss can be written as

$$\mathcal{L}(y_n, \hat{y}_n) = \mathcal{L}(y_n, \boldsymbol{\theta}_{n-1}^T \mathbf{x}_n) := \mathcal{L}_n(\boldsymbol{\theta}_{n-1}).$$

Adapting (8.76) to the previous notation, we can write

$$\text{Regret}_N(\mathbf{h}) = \sum_{n=1}^N \mathcal{L}_n(\boldsymbol{\theta}_{n-1}) - \sum_{n=1}^N \mathcal{L}_n(\mathbf{h}), \quad (8.78)$$

where $\mathbf{h} \in C \subseteq \mathbb{R}^l$ is a *fixed* parameter vector in the set C where solutions are sought.

Before proceeding further, it is interesting to note that the cumulative loss is based on the loss suffered by the learner, against y_n, \mathbf{x}_n , using the estimate, $\boldsymbol{\theta}_{n-1}$, which has been trained on data up to and including time instant $n-1$. The pair (y_n, \mathbf{x}_n) is not involved in its training. From this point of view, the cumulative loss is in line with our desire to guard against overfitting.

In the framework of regret analysis, the path to follow is to derive an upper bound for the regret, exploiting the convexity of the employed loss function. We will demonstrate the technique via a case study; that of the online version of the simple subgradient algorithm.

Regret analysis of the subgradient algorithm

The online version of (8.68), for minimizing the expected loss, $\mathbb{E} [\mathcal{L}(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})]$, is written as

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \mathbf{g}_n, \quad (8.79)$$

where for notational convenience, the subgradient is denoted as

$$\mathbf{g}_n := \mathcal{L}'_n(\boldsymbol{\theta}_{n-1}).$$

Proposition 8.4. *Assume that the subgradients of the loss function are bounded, shown as*

$$\|\mathbf{g}_n\| \leq G, \quad \forall n. \quad (8.80)$$

Furthermore, assume that the set of solutions, \mathcal{S} , is bounded; that is, $\forall \boldsymbol{\theta}, \mathbf{h} \in \mathcal{S}$, there exists a bound F , such that

$$\|\boldsymbol{\theta} - \mathbf{h}\| \leq F. \quad (8.81)$$

Let $\boldsymbol{\theta}_$ be an optimal (desired) predictor. Then, if $\mu_n = \frac{1}{\sqrt{n}}$,*

$$\frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(\boldsymbol{\theta}_{n-1}) \leq \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(\boldsymbol{\theta}_*) + \frac{F^2}{2\sqrt{N}} + \frac{G^2}{\sqrt{N}}. \quad (8.82)$$

In words, as $N \rightarrow \infty$ the average cumulative loss tends to the average loss of the optimal predictor.

Proof. Since the adopted loss function is assumed to be convex and by the definition of the subgradient, we have

$$\mathcal{L}_n(\mathbf{h}) \geq \mathcal{L}_n(\boldsymbol{\theta}_{n-1}) + \mathbf{g}_n^T(\mathbf{h} - \boldsymbol{\theta}_{n-1}), \quad \forall \mathbf{h} \in \mathbb{R}^l, \quad (8.83)$$

or

$$\mathcal{L}_n(\boldsymbol{\theta}_{n-1}) - \mathcal{L}_n(\mathbf{h}) \leq \mathbf{g}_n^T(\boldsymbol{\theta}_{n-1} - \mathbf{h}). \quad (8.84)$$

However, recalling (8.79), we can write that

$$\boldsymbol{\theta}_n - \mathbf{h} = \boldsymbol{\theta}_{n-1} - \mathbf{h} - \mu_n \mathbf{g}_n, \quad (8.85)$$

which results in

$$\begin{aligned} \|\boldsymbol{\theta}_n - \mathbf{h}\|^2 &= \|\boldsymbol{\theta}_{n-1} - \mathbf{h}\|^2 + \mu_n^2 \|\mathbf{g}_n\|^2 \\ &\quad - 2\mu_n \mathbf{g}_n^T(\boldsymbol{\theta}_{n-1} - \mathbf{h}). \end{aligned} \quad (8.86)$$

Taking into account the bound of the subgradient, Eq. (8.86) leads to the inequality,

$$\mathbf{g}_n^T(\boldsymbol{\theta}_{n-1} - \mathbf{h}) \leq \frac{1}{2\mu_n} \left(\|\boldsymbol{\theta}_{n-1} - \mathbf{h}\|^2 - \|\boldsymbol{\theta}_n - \mathbf{h}\|^2 \right) + \frac{\mu_n}{2} G^2. \quad (8.87)$$

Summing up both sides of (8.87), taking into account inequality (8.84) and after a bit of algebra ([Problem 8.30](#)) results in

$$\sum_{n=1}^N \mathcal{L}_n(\boldsymbol{\theta}_{n-1}) - \sum_{n=1}^N \mathcal{L}_n(\mathbf{h}) \leq \frac{1}{2\mu_N} F^2 + \frac{G^2}{2} \sum_{n=1}^N \mu_n. \quad (8.88)$$

Setting $\mu_n = \frac{1}{\sqrt{n}}$, using the obvious bound

$$\sum_{n=1}^N \frac{1}{\sqrt{n}} \leq 1 + \int_1^N \frac{1}{\sqrt{t}} dt = 2\sqrt{N} - 1, \quad (8.89)$$

and dividing both sides of (8.88) by N , the proposition is proved for any \mathbf{h} . Hence, it will also be true for $\boldsymbol{\theta}_*$. \square

The previous proof follows the one given in [106]; this was the first paper to adopt the notion of “regret” for the analysis of convex online algorithms. Proofs given later for more complex algorithms have borrowed, in one way or another, the arguments used there.

Remarks 8.9.

- Tighter regret bounds can be derived when the loss function is strongly convex, [42]. A function $f : \mathcal{X} \subseteq \mathbb{R}^l \mapsto \mathbb{R}$ is said to be σ -strongly convex, if, $\forall \mathbf{y}, \mathbf{x} \in \mathcal{X}$,

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}) + \frac{\sigma}{2} \|\mathbf{y} - \mathbf{x}\|^2, \quad (8.90)$$

for any subgradient \mathbf{g} at \mathbf{x} . It also turns out that a function $f(\mathbf{x})$ is strongly convex if $f(\mathbf{x}) - \frac{\sigma}{2} \|\mathbf{x}\|^2$ is convex ([Problem 8.31](#)).

For σ -strongly convex loss functions, if the step-size of the subgradient algorithm is diminishing at a rate $\mathcal{O}(\frac{1}{\sigma n})$ then the average cumulative loss is approaching the average loss of the optimal predictor at a rate $\mathcal{O}(\frac{\ln N}{N})$ ([Problem 8.32](#)). This is the case, for example, for the PEGASOS algorithm, discussed in [Section 8.10.3](#).

- In [4, 5], $\mathcal{O}(1/N)$ convergence rates are derived for a set of not strongly convex smooth loss functions (squared error and logistic regression) even for the case of constant step-sizes. The analysis method follows statistical arguments.

8.12 ONLINE LEARNING AND BIG DATA APPLICATIONS: A DISCUSSION

Online learning algorithms have been treated in [Chapters 4, 5](#) and [6](#). The purpose of this section is first to summarize some of the findings and at the same time to present a discussion related to the performance of online schemes compared to their batch relatives.

Recall that the ultimate goal in obtaining a parametric predictor,

$$\hat{y} = f_{\theta}(\mathbf{x}),$$

is to select θ so that to optimize the *expected loss/risk* function, [\(8.65\)](#). For practical reasons, the corresponding empirical formulation in [\(8.66\)](#) is most often adopted instead. From the learning theory's point of view, this is justified provided the respective class of functions is sufficiently restrictive [89]. The available literature is quite rich in obtaining performance bounds that measure how close the optimal value obtained via the expected risk is to that obtained via the empirical one, as a function of the number of points N . Note that as $N \rightarrow \infty$, and recalling well-known arguments from probability theory and statistics, the empirical risk tends to the expected risk (under general assumptions). Thus, for very large training data sets, adopting the empirical risk may not be that different from using the expected risk. However, for data sets of shorter lengths, a number of issues occur. Besides the value of N , another critical factor enters the scene; this is the *complexity* of the family of the functions, in which we search a solution. In other words, the generalization performance critically depends not only on N but also on how large or small this set of functions is. A related discussion for the specific case of the MSE was presented in [Chapter 3](#), in the context of the bias-variance trade-off. The roots of the more general theory go back to the pioneering work of Vapnik-Chernovenkis; see [31, 90, 91], and [83] for a less mathematical summary of the major points.

In the sequel, we will summarize some of the available results tailored to the needs of our current discussion.

Approximation, estimation and optimization errors

Recall that all we are given in a machine learning task is the available training set of examples. To set up the “game,” the designer has to decide on the selection of: (a) the loss function, $\mathcal{L}(\cdot, \cdot)$, which measures the deviation (error) between predicted and desired values and (b) the set of (parametric) functions \mathcal{F} ,

$$\mathcal{F} = \left\{ f_{\theta}(\cdot) : \theta \in \mathbb{R}^K \right\}.$$

Based on the choice of $\mathcal{L}(\cdot, \cdot)$, the *benchmark* function, denoted as f_* , is the one that minimizes the expected risk (see also [Chapter 3](#)), that is,

$$f_*(\cdot) = \arg \min_f \mathbb{E} [\mathcal{L}(y, f(\mathbf{x}))],$$

or equivalently

$$f_*(\mathbf{x}) = \arg \min_{\hat{y}} \mathbb{E} [\mathcal{L}(y, \hat{y}) | \mathbf{x}]. \quad (8.91)$$

Let also f_{θ_*} denote the optimal function that results by minimizing the expected risk constrained within the parametric family \mathcal{F} , that is,

$$f_{\theta_*}(\cdot) : \theta_* = \arg \min_{\theta} \mathbb{E} [\mathcal{L}(y, f_{\theta}(\mathbf{x}))]. \quad (8.92)$$

However, instead of f_{θ_*} , we obtain another function, denoted as f_N , by minimizing the empirical risk, $J_N(\theta)$,

$$f_N(\mathbf{x}) := f_{\theta_*(N)}(\mathbf{x}) : \theta_*(N) = \arg \min_{\theta} J_N(\theta). \quad (8.93)$$

Once f_N has been obtained, we are interested in evaluating its generalization performance; that is, to compute the value of the expected risk at f_N , $\mathbb{E}[\mathcal{L}(y, f_N(\mathbf{x}))]$. The excess error with respect to the optimal value can then be decomposed as [\[12\]](#),

$$\mathcal{E} = \mathbb{E} [\mathcal{L}(y, f_N(\mathbf{x}))] - \mathbb{E} [\mathcal{L}(y, f_*(\mathbf{x}))] = \mathcal{E}_{\text{appr}} + \mathcal{E}_{\text{est}} \quad (8.94)$$

where,

$$\mathcal{E}_{\text{appr}} := \mathbb{E} [\mathcal{L}(y, f_{\theta_*}(\mathbf{x}))] - \mathbb{E} [\mathcal{L}(y, f_*(\mathbf{x}))] : \text{Approximation Error,}$$

$$\mathcal{E}_{\text{est}} := \mathbb{E} [\mathcal{L}(y, f_N(\mathbf{x}))] - \mathbb{E} [\mathcal{L}(y, f_{\theta_*}(\mathbf{x}))] : \text{Estimation Error,}$$

where $\mathcal{E}_{\text{appr}}$ is known as the *approximation* error and \mathcal{E}_{est} is known as the *estimation* error. The former measures how well the chosen family of functions can perform compared to the optimal/benchmark value and the latter measures the performance loss within the family \mathcal{F} , due to the fact that optimization is performed via the empirical loss function. Large families of functions lead to low approximation error but higher estimation error and vice versa. A way to improve upon the estimation error, while keeping the approximation error small, is to increase N . The size/complexity of the family \mathcal{F} is measured by its *capacity*, which may depend on the number of parameters, but this is not always the whole story; see, for example, [\[83, 90\]](#). For example, the use of regularization, while minimizing the empirical risk, can have a decisive effect on the approximation-estimation error trade-off.

In practice, while optimizing the (regularized) empirical risk, one has to adopt an iterative minimization or an online algorithm, which leads to an approximate solution, denoted as \tilde{f}_N . Then the excess error in [\(8.94\)](#) involves a third term, [\[12, 13\]](#),

$$\mathcal{E} = \mathcal{E}_{\text{appr}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}}, \quad (8.95)$$

where

$$\mathcal{E}_{\text{opt}} := \mathbb{E} [\mathcal{L}(y, \tilde{f}_N(\mathbf{x}))] - \mathbb{E} [\mathcal{L}(y, f_N(\mathbf{x}))] : \text{Optimization Error.}$$

The literature is rich in deriving bounds concerning the excess error. More detailed treatment is beyond the scope of this book. As a case study, we will follow the treatment given in [13].

Let the computation of \tilde{f}_N be associated with a predefined accuracy

$$\mathbb{E} [\mathcal{L}(y, \tilde{f}_N(\mathbf{x}))] \leq \mathbb{E} [\mathcal{L}(y, f_N(\mathbf{x}))] + \rho.$$

Then, for a class of functions that are often met in practice, for example, under strong convexity of the loss function [50] or under certain assumptions on the data distribution [87], the following equivalence relation can be established,

$$\mathcal{E}_{\text{appr}} + \mathcal{E}_{\text{est}} + \mathcal{E}_{\text{opt}} \sim \mathcal{E}_{\text{appr}} + \left(\frac{\ln N}{N} \right)^a + \rho, \quad a \in \left[\frac{1}{2}, 1 \right], \quad (8.96)$$

which verifies the fact that as $N \rightarrow \infty$ the estimation error decreases and provides a rule for the respective convergence rate. The excess error \mathcal{E} , besides the approximation component, on which we have no access to control (given the family of functions, \mathcal{F}), it depends on (a) the number of data and (b) on the accuracy, ρ , associated with the algorithm used. How one can control these parameters depends on the type of learning task at hand.

- *Small scale tasks:* These types of tasks are constrained by the number of training points N . In this case, one can reduce the optimization error, since computational load is not a problem, and achieve the minimum possible estimation error, as this is allowed by the number of available training points. In this case, one achieves the approximation-estimation trade-off.
- *Large scale/big data tasks:* These types of tasks are constrained by the computational resources. Thus, a computationally cheap and less accurate algorithm may end up with lower excess error, since it has the luxury of exploiting more data, compared to a more accurate yet computationally more complex algorithm, given the maximum allowed computational load.

Batch versus online learning

Our interest in this subsection lies in investigating whether there is a performance loss if in place of a batch algorithm an online one is used instead. There is a very subtle issue involved here, which turns out to be very important from a practical point of view. We will restrict our discussion to differentiable convex loss functions.

Two major factors associated with the performance of an algorithm (in a stationary environment) are its convergence rate and its accuracy after convergence. The general form of a batch algorithm in minimizing (8.66) is written as

$$\begin{aligned} \boldsymbol{\theta}^{(i)} &= \boldsymbol{\theta}^{(i-1)} - \mu_i \Phi_i \nabla J_N(\boldsymbol{\theta}^{(i-1)}) \\ &= \boldsymbol{\theta}^{(i-1)} - \frac{\mu_i}{N} \Phi_i \sum_{n=1}^N \mathcal{L}'(\boldsymbol{\theta}^{(i-1)}, y_n, \mathbf{x}_n). \end{aligned} \quad (8.97)$$

For gradient descent, $\Phi_i = I$, and for Newton-type recursions, Φ_i is the inverse Hessian matrix of the loss function (Chapter 6).

Note that these are not the only possible choices for matrix Φ . For example, in the Levenberg-Marquardt method, the square Jacobian is employed, that is,

$$\Phi_i = \left[\nabla J(\boldsymbol{\theta}^{(i-1)}) \nabla^T J(\boldsymbol{\theta}^{(i-1)}) + \lambda I \right]^{-1},$$

where λ is a regularization parameter. In [3], the *natural gradient* is proposed, which is based on the Fisher information matrix associated with the noisy distribution implied by the adopted prediction model, $f_{\theta}(\mathbf{x})$. In both cases, the involved matrices asymptotically behave like the Hessian, yet they may provide improved performance during the initial convergence phase. For a further discussion, the interested reader may consult [48, 55].

As it has already been mentioned in [Chapters 5 and 6](#) ([Section 6.47](#)), the convergence rate to the respective optimal value of the simple gradient descent method is *linear*, that is,

$$\ln \frac{1}{\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*(N)\|^2} \propto i,$$

and the corresponding rate for a Newton-type algorithm is (approximately) quadratic, that is,

$$\ln \ln \frac{1}{\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}_*(N)\|^2} \propto i.$$

In contrast, the online version of [\(8.97\)](#), that is,

$$\boldsymbol{\theta}_n = \boldsymbol{\theta}_{n-1} - \mu_n \Phi_n \mathcal{L}'(\boldsymbol{\theta}_{n-1}, y_n, \mathbf{x}_n), \quad (8.98)$$

is based on a *noisy* estimation of the gradient, using the current sample point, (y_n, \mathbf{x}_n) , only. The effect of this is to slow down convergence, in particular when the algorithm gets close to the solution. Moreover, the estimate of the parameter vector fluctuates around the optimal value. We have extensively studied this phenomenon in the case of the LMS, when μ_n is assigned a constant value. This is the reason that in the stochastic gradient rationale, μ_n must be a decreasing sequence. However, it *must not* decrease very fast, which is guaranteed by the condition $\sum_n \mu_n \rightarrow \infty$, [Section 5.4](#). Furthermore, recall from our discussion there that the rate of convergence toward $\boldsymbol{\theta}_*$ is, on average, $\mathcal{O}(1/n)$. This result also covers the more general case of online algorithms given in [\(8.98\)](#), see, for example, [55]. Note, however, that all these results have been derived under a number of assumptions, for example, that the algorithm is close enough to a solution.

Our major interest now turns on comparing the rate at which a batch and a corresponding online algorithm converge to $\boldsymbol{\theta}_*$; that is, the value that minimizes the expected risk, which is the ultimate goal of our learning task. Since the aim is to compare performances, given the same number of training samples, let us use the same number, both for n in the online and N for the batch. Following [11] and applying a second order Taylor expansion on $J_n(\boldsymbol{\theta})$, it can be shown ([Problem 8.28](#)) that

$$\boldsymbol{\theta}_*(n) = \boldsymbol{\theta}_*(n-1) - \frac{1}{n} \Psi_n^{-1} \mathcal{L}'(\boldsymbol{\theta}_*(n-1), y_n, \mathbf{x}_n), \quad (8.99)$$

where

$$\Psi_n = \left(\frac{1}{n} \sum_{k=1}^n \nabla^2 \mathcal{L}(\boldsymbol{\theta}_*(n-1), y_k, \mathbf{x}_k) \right).$$

Note that [\(8.99\)](#) is similar in structure with [\(8.98\)](#). Also, as $n \rightarrow \infty$, Ψ_n converges to the Hessian matrix, H , of the expected risk function. Hence, for appropriate choices of the involved weighting matrices and setting $\mu_n = 1/n$, [\(8.98\)](#) and [\(8.99\)](#) can converge to $\boldsymbol{\theta}_*$ at similar rates; thus, in both cases, the critical factor that determines how close to the optimal, $\boldsymbol{\theta}_*$, the resulting estimates are, is the number of data points used. It can be shown [11, 55, 88], that

$$\mathbb{E} [||\theta_n - \theta_*||^2] + \mathcal{O}\left(\frac{1}{n}\right) = \mathbb{E} [||\theta_*(n) - \theta_*||^2] + \mathcal{O}\left(\frac{1}{n}\right) = \frac{C}{n},$$

where C is a constant depending on the specific form of the associated expected loss function used. *Thus, batch algorithms and their online versions can be made to converge at similar rates to θ_* , after appropriate fine-tuning of the involved parameters.* Once more, since the critical factor in big data applications is not data but computational resources, a cheap online algorithm can achieve enhanced performance (lower excess error) compared to a batch, yet computationally more thirsty scheme. This is because for a given computational load, the online algorithm can process more data points ([Problem 8.33](#)). More importantly, an online algorithm needs not to store the data, which can be processed on the fly as they arrive. For a more detailed treatment of the topic, the interested reader may consult [[13](#)].

In [[13](#)], two forms of batch linear support vector machines ([Chapter 11](#)) were tested against their online stochastic gradient counterparts. The tests were carried out on the RCV1 data basis [[52](#)], and the training set comprised 781,265 documents represented by (relatively) sparse feature vectors consisting of 47,152 feature values. The stochastic gradient online versions, appropriately tuned with a diminishing step-size, achieved comparable error rates at substantially lower computational times (less than one tenth) compared to their batch processing relatives.

Remarks 8.10.

- Most of our discussion on the online versions has been focused on the simplest version, given in ([8.98](#)) for $\Phi_n = I$. However, the topic of stochastic gradient descent schemes, especially in the context of smooth loss functions, has a very rich history of over 60 years, and many algorithmic variants have been “born.” In [Chapter 5](#), a number of variations of the basic LMS scheme were discussed. Some more notable examples, which are still popular are:

Stochastic gradient descent with momentum: The basic iteration of this variant is

$$\theta_n = \theta_{n-1} - \mu_n \mathcal{L}'_n(\theta_{n-1}) + \beta_n(\theta_{n-1} - \theta_{n-2}). \quad (8.100)$$

Very often, $\beta_n = \beta$ is chosen to be a constant; see, for example, [[86](#)].

Gradient averaging: Another widely used version results if the place of the single gradient is taken by an average estimate, that is,

$$\theta_n = \theta_{n-1} - \frac{\mu_n}{n} \sum_{k=1}^n \mathcal{L}'_k(\theta_{n-1}). \quad (8.101)$$

Variants with different averaging scenarios (e.g., random selection instead of using all previously points) are also around. Such an averaging has a smoothing effect on the convergence of the algorithm. We have already seen this rationale in the context of the PEGASOS algorithm ([Section 8.10.3](#)). The general trend of all the variants of the basic stochastic gradient scheme is to improve upon the constants, but the convergence rate still remains to be $\mathcal{O}(1/n)$.

In [[49](#)], the online learning rationale was used in the context of data sets of fixed size, N . Instead of using the gradient descent scheme in ([8.97](#)), the following version is proposed,

$$\theta^{(i)} = \theta^{(i-1)} - \frac{\mu_i}{N} \sum_{k=1}^N g_k^{(i)}, \quad (8.102)$$

where

$$\mathbf{g}_k^{(i)} = \begin{cases} \mathcal{L}'_k(\boldsymbol{\theta}^{(i-1)}), & \text{if } k = i_k, \\ \mathbf{g}_k^{(i-1)}, & \text{otherwise.} \end{cases} \quad (8.103)$$

The index i_k is randomly chosen every time from $\{1, 2, \dots, N\}$. Thus, in each iteration only one gradient is computed and the rest are drawn from the memory. It turns out that, for strongly convex smooth loss functions, the algorithm exhibits linear convergence to the solution of the empirical loss in 8.56. Of course, compared to the basic online schemes, an $\mathcal{O}(N)$ memory is required, for keeping track of the gradient computations.

- The literature on deriving performance bounds concerning online algorithms is very rich, both in numbers as well as in ideas. For example, another line of research involves bounds for *arbitrary* online algorithms; see, [1, 19, 66] and references therein.

8.13 PROXIMAL OPERATORS

So far in the chapter, we have devoted a lot of space to the notion of the projection operator. In this section, we go one step further and we will introduce an elegant generalization of the notion of projection. Just to establish a clear understanding, when we refer to an operator, we mean a mapping from $\mathbb{R}^l \mapsto \mathbb{R}^l$, in contrast to a function which is a mapping $\mathbb{R}^l \mapsto \mathbb{R}$.

Definition 8.9. Let

$$f : \mathbb{R}^l \mapsto \mathbb{R},$$

be a convex function and $\lambda > 0$. The corresponding *proximal* or *proximity* operator of index λ [59, 68],

$$\text{Prox}_{\lambda f} : \mathbb{R}^l \mapsto \mathbb{R}^l, \quad (8.104)$$

is defined such as,

$$\text{Prox}_{\lambda f}(\mathbf{x}) := \arg \min_{\mathbf{v} \in \mathbb{R}^l} \left\{ f(\mathbf{v}) + \frac{1}{2\lambda} \|\mathbf{x} - \mathbf{v}\|^2 \right\} : \quad \text{Proximal Operator.} \quad (8.105)$$

We stress that the proximal operator is a point in \mathbb{R}^l . The definition can also be extended to include functions defined as $f : \mathbb{R}^l \mapsto \mathbb{R} \cup \{+\infty\}$. A closely related notion to the proximal operator is the following.

Definition 8.10. Let f be a convex function as in the previous definition. We call the *Moreau envelope*, the function

$$e_{\lambda f}(\mathbf{x}) := \min_{\mathbf{v} \in \mathbb{R}^l} \left\{ f(\mathbf{v}) + \frac{1}{2\lambda} \|\mathbf{x} - \mathbf{v}\|^2 \right\} : \quad \text{Moreau Envelope.} \quad (8.106)$$

Note that the Moreau envelope [58] is a *function* related to the proximal *operator* as

$$e_{\lambda f}(\mathbf{x}) = f(\text{Prox}_{\lambda f}(\mathbf{x})) + \frac{1}{2\lambda} \|\mathbf{x} - \text{Prox}_{\lambda f}(\mathbf{x})\|^2. \quad (8.107)$$

The Moreau envelope can also be thought of as a regularized minimization, and it is also known as the *Moreau-Yosida regularization* [104].

A first point to clarify is whether the minimum in (8.105) exists. Note that the two terms in the brackets are both convex; namely, $f(\mathbf{v})$, and the quadratic term $\|\mathbf{x} - \mathbf{v}\|^2$. Hence, as it can easily be shown by recalling the definition of convexity, their sum is also convex. Moreover, the latter of the two terms is strictly convex, hence their sum is also strictly convex, which guarantees a *unique* minimum.

Example 8.9. Let us calculate $\text{Prox}_{\lambda\iota_C}$, where $\iota_C : \mathbb{R}^l \mapsto \mathbb{R} \cup \{+\infty\}$ stands for the indicator function of a nonempty closed convex subset $C \subset \mathbb{R}^l$, defined as

$$\iota_C(\mathbf{x}) := \begin{cases} 0, & \text{if } \mathbf{x} \in C, \\ +\infty, & \text{if } \mathbf{x} \notin C. \end{cases}$$

It is not difficult to verify that

$$\begin{aligned} \text{Prox}_{\lambda\iota_C}(\mathbf{x}) &= \arg \min_{\mathbf{v} \in \mathbb{R}^l} \left\{ \iota_C(\mathbf{v}) + \frac{1}{2\lambda} \|\mathbf{x} - \mathbf{v}\|^2 \right\} \\ &= \arg \min_{\mathbf{v} \in C} \|\mathbf{x} - \mathbf{v}\|^2 = P_C(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbb{R}^l, \forall \lambda > 0, \end{aligned}$$

where P_C is the (metric) projection mapping onto C .

Moreover,

$$\begin{aligned} e_{\lambda\iota_C}(\mathbf{x}) &= \min_{\mathbf{v} \in \mathbb{R}^l} \left\{ \iota_C(\mathbf{v}) + \frac{1}{2\lambda} \|\mathbf{x} - \mathbf{v}\|^2 \right\} \\ &= \min_{\mathbf{v} \in C} \frac{1}{2\lambda} \|\mathbf{x} - \mathbf{v}\|^2 = \frac{1}{2\lambda} d_C(\mathbf{x}), \end{aligned}$$

where d_C stands for the (metric) distance function to C (Example 8.5), defined as $d_C(\mathbf{x}) := \min_{\mathbf{v} \in C} \|\mathbf{x} - \mathbf{v}\|$.

Thus, as said in the beginning of this section, the proximal operator can be considered as the generalization of the projection one.

Example 8.10. In the case where f becomes the ℓ_1 -norm of a vector, that is,

$$\|\mathbf{x}\|_1 = \sum_{i=1}^l |x_i|, \quad \forall \mathbf{x} \in \mathbb{R}^l,$$

then it is easily determined that (8.105) decomposes into a set of l scalar minimization tasks, that is,

$$\text{Prox}_{\lambda\|\cdot\|_1}(\mathbf{x})|_i = \arg \min_{v_i \in \mathbb{R}} \left\{ |v_i| + \frac{1}{2\lambda} (x_i - v_i)^2 \right\}, \quad i = 1, 2, \dots, l, \quad (8.108)$$

where $\text{Prox}_{\lambda\|\cdot\|_1}(\mathbf{x})|_i$ denotes the respective i th element. Minimizing (8.108), is equivalent with requiring the subgradient to be zero, which results in

$$\begin{aligned} \text{Prox}_{\lambda\|\cdot\|_1}(\mathbf{x})|_i &= \begin{cases} x_i - \text{sgn}(x_i)\lambda, & \text{if } |x_i| > \lambda, \\ 0, & \text{if } |x_i| \leq \lambda \end{cases} \\ &= \text{sgn}(x_i) \max\{0, |x_i| - \lambda\}. \end{aligned} \quad (8.109)$$

For the time being, the proof is left as an exercise. The same task is treated in detail in Chapter 9 and the proof is provided in Section 9.3. The operation in (8.109) is also known as *soft thresholding*. In other words, it sets to zero all values with magnitude less than a threshold value (λ) and adds a constant bias (depending on the sign) to the rest. To provoke the unfamiliar reader a bit, this is a way to impose sparsity on a parameter vector.

Having calculated $\text{Prox}_{\lambda \|\cdot\|_1}(\mathbf{x})$, the Moreau envelope of $\|\cdot\|_1$ can be directly obtained by

$$\begin{aligned} e_{\lambda \|\cdot\|_1}(\mathbf{x}) &= \sum_{i=1}^l \left(\frac{1}{2\lambda} (x_i - \text{Prox}_{\lambda \|\cdot\|_1}(\mathbf{x})|_i)^2 + |\text{Prox}_{\lambda \|\cdot\|_1}(\mathbf{x})|_i \right) \\ &= \sum_{i=1}^l \left(\chi_{[0,\lambda]}(|x_i|) \frac{x_i^2}{2\lambda} + \chi_{(\lambda,+\infty)}(|x_i|) \left(|x_i - \text{sgn}(x_i)\lambda| + \frac{\lambda}{2} \right) \right) \\ &= \sum_{i=1}^l \left(\chi_{[0,\lambda]}(|x_i|) \frac{x_i^2}{2\lambda} + \chi_{(\lambda,+\infty)}(|x_i|) \left(|x_i| - \frac{\lambda}{2} \right) \right), \end{aligned}$$

where $\chi_{\mathcal{A}}(\cdot)$ denotes the characteristic function of the set \mathcal{A} , defined in (8.59). For the one-dimensional case, $l = 1$, the previous Moreau envelope boils down to

$$e_{\lambda|\cdot|}(x) = \begin{cases} |x| - \frac{\lambda}{2}, & \text{if } |x| > \lambda, \\ \frac{x^2}{2\lambda}, & \text{if } |x| \leq \lambda. \end{cases}$$

This envelope and the original $|\cdot|$ functions are depicted in Figure 8.31. It is worth-noticing here that $e_{\lambda|\cdot|}$ is a scaled version, more accurately $1/\lambda$ times, of the celebrated Huber's function; a loss function vastly used against outliers in robust statistics, which will be discussed in more detail in Chapter 11. Note that the Moreau envelope is a “blown-up” smoothed version of the ℓ_1 norm function and although the original function is not differentiable, its Moreau envelope is continuously differentiable; moreover, *they both share the same minimum*. This is most interesting and we will come back to that very soon.

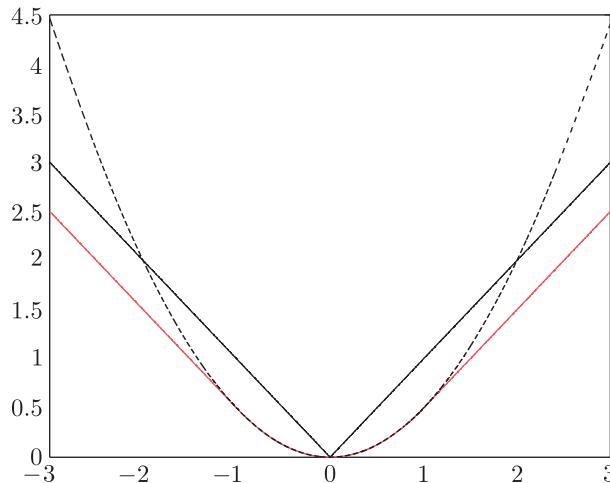


FIGURE 8.31

The $|x|$ function (black solid line), its Moreau envelope $e_{\lambda|\cdot|}(x)$ (red solid line), and $x^2/2$ (black dotted line), for $x \in \mathbb{R}$. Even if $|\cdot|$ is nondifferentiable at 0, $e_{\lambda|\cdot|}(x)$ is everywhere differentiable. Notice also that although $x^2/2$ and $e_{\lambda|\cdot|}(x)$ behave exactly the same for small values of x , $e_{\lambda|\cdot|}(x)$ is more conservative than $x^2/2$ in penalizing large values of x ; this is the reason for the extensive usability of the Huber function, a scaled-down version of $e_{\lambda|\cdot|}(x)$, as a robust tool against outliers in robust statistics.

8.13.1 PROPERTIES OF THE PROXIMAL OPERATOR

We now focus on some basic properties of the proximal operator, which will soon be used to give birth to a new class of algorithms for the minimization of nonsmooth convex loss functions.

Proposition 8.5. *Let a convex function*

$$f : \mathbb{R}^l \mapsto \mathbb{R} \cup \{+\infty\},$$

and $\text{Prox}_{\lambda f}(\cdot)$ its corresponding proximal operator of index λ . Then,

$$\mathbf{p} = \text{Prox}_{\lambda f}(\mathbf{x}),$$

if and only if

$$\langle \mathbf{y} - \mathbf{p}, \mathbf{x} - \mathbf{p} \rangle \leq \lambda (f(\mathbf{y}) - f(\mathbf{p})), \quad \forall \mathbf{y} \in \mathbb{R}^l. \quad (8.110)$$

Another necessary condition is

$$\|\text{Prox}_{\lambda f}(\mathbf{x}) - \text{Prox}_{\lambda f}(\mathbf{y})\|^2 \leq \langle \mathbf{x} - \mathbf{y}, \text{Prox}_{\lambda f}(\mathbf{x}) - \text{Prox}_{\lambda f}(\mathbf{y}) \rangle. \quad (8.111)$$

The proofs of (8.110) and (8.111) are given in [Problems 8.34](#) and [8.35](#), respectively. Note that (8.111) is of the same flavor as (8.16), which is inherited to the proximal operator from its more primitive ancestor. In the sequel, we will make use of these properties to touch upon the algorithmic front, where our main interest lies.

Lemma 8.3. *Consider the convex function*

$$f : \mathbb{R}^l \mapsto \mathbb{R} \cup \{+\infty\},$$

and its proximal operator $\text{Prox}_{\lambda f}(\cdot)$, of index λ . Then, the fixed point set of the proximal operator coincides with the set of minimizers of f , that is,

$$\text{Fix}(\text{Prox}_{\lambda f}) = \left\{ \mathbf{x} : \mathbf{x} = \arg \min_{\mathbf{y}} f(\mathbf{y}) \right\}. \quad (8.112)$$

Proof. The definition of the fixed point set has been given in [Section 8.3.1](#). We first assume that a point \mathbf{x} belongs to the fixed point set, hence the action of the proximal operator leaves it unaffected, that is,

$$\mathbf{x} = \text{Prox}_{\lambda f}(\mathbf{x}),$$

and making use of (8.110), we get

$$\langle \mathbf{y} - \mathbf{x}, \mathbf{x} - \mathbf{x} \rangle \leq \lambda (f(\mathbf{y}) - f(\mathbf{x})), \quad \forall \mathbf{y} \in \mathbb{R}^l, \quad (8.113)$$

which results in

$$f(\mathbf{x}) \leq f(\mathbf{y}), \quad \forall \mathbf{y} \in \mathbb{R}^l. \quad (8.114)$$

That is, \mathbf{x} is a minimizer of f . For the converse, we assume that \mathbf{x} is a minimizer. Then (8.114) is valid, from which (8.113) is deduced, and since this is a necessary and sufficient condition for a point to be equal to the value of the proximal operator, we have proved the claim. \square

This is a very interesting and elegant result. One can obtain the set of minimizers of a nonsmooth convex function by solving an equivalent smooth one. From a practical point of view, the value of the method depends on how easy it is to obtain the proximal operator. For example, we have already seen that if the goal is to minimize the ℓ_1 norm, the proximal operator is a simple soft-thresholding operation. Needless to say that life is not always that generous!

8.13.2 PROXIMAL MINIMIZATION

In this section, we will exploit our experience from [Section 8.4](#) to develop iterative schemes which asymptotically land their estimates in the fixed point set of the respective operator. All that is required is for the operator to own a nonexpansiveness property.

Proposition 8.6. *The proximal operator associated with a convex function is nonexpansive, that is,*

$$\|\text{Prox}_{\lambda f}(\mathbf{x}) - \text{Prox}_{\lambda f}(\mathbf{y})\| \leq \|\mathbf{x} - \mathbf{y}\|. \quad (8.115)$$

Proof. The proof is readily obtained as a combination of the property in (8.111) with the Cauchy-Schwarz inequality. Moreover, it can also be shown that the relaxed version of the proximal operator (also known as the *reflected* version),

$$R_{\lambda f}(\mathbf{x}) := 2 \text{Prox}_{\lambda f}(\mathbf{x}) - I, \quad (8.116)$$

is also nonexpansive with the same fixed point set as that of the proximal operator, [Problem 8.36](#). \square

Proposition 8.7. *Let*

$$f : \mathbb{R}^l \mapsto \mathbb{R} \cup \{+\infty\},$$

be a convex function, with the $\text{Prox}_{\lambda f}$ being the respective proximal operator of index λ . Then, starting from an arbitrary point, $\mathbf{x}_0 \in \mathbb{R}^l$, the following iterative algorithm

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \mu_k (\text{Prox}_{\lambda f}(\mathbf{x}_{k-1}) - \mathbf{x}_{k-1}), \quad (8.117)$$

where $\mu_k \in (0, 2)$ is such as

$$\sum_{k=1}^{\infty} \mu_k (2 - \mu_k) = +\infty,$$

converges to an element of the fixed point set of the proximal operator; that is, it converges to a minimizer off. Proximal minimization algorithms are traced back in the early 1970s [56, 69].

The proof of the proposition is given in [Problem 8.36](#) [81]. Observe that (8.117) is the counterpart of (8.26). A special case occurs if $\mu_k = 1$, which results in

$$\mathbf{x}_k = \text{Prox}_{\lambda f}(\mathbf{x}_{k-1}), \quad (8.118)$$

also known as the *proximal point* algorithm.

Example 8.11. Let us demonstrate the previous findings via the familiar optimization task of the quadratic function

$$f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x}.$$

It does not take long to see that the minimizer occurs at the solution of the linear system of equations

$$Ax_* = b.$$

From the definition in (8.105), taking the gradient of the quadratic function and equating to zero we readily obtain that,

$$\text{Prox}_{\lambda f}(x) = \left(A + \frac{1}{\lambda} I \right)^{-1} \left(b + \frac{1}{\lambda} x \right), \quad (8.119)$$

and setting $\epsilon = \frac{1}{\lambda}$, the recursion in (8.118) becomes

$$x_k = (A + \epsilon I)^{-1} (b + \epsilon x_{k-1}). \quad (8.120)$$

After some simple algebraic manipulations (Problem 8.37), we finally obtain

$$x_k = x_{k-1} + (A + \epsilon I)^{-1} (b - Ax_{k-1}). \quad (8.121)$$

This scheme is known from the numerical linear algebra as *iterative refinement* algorithm [57]. It is used when the matrix A is near singular, so the regularization via ϵ helps the inversion. Note that at each iteration, $b - Ax_{k-1}$ is the error committed by the current estimate. The algorithm belongs to a larger family of algorithms, known as stationary iterative or iterative relaxation schemes; we will meet such schemes in Chapter 10.

The interesting point here is that since the algorithm results as a special case of the proximal minimization algorithm, convergence to the solution is guaranteed even if ϵ is not small!

Resolvent of the subdifferential mapping

We will look at the proximal operator from a slightly different view, which will be useful to us soon, when it will be used for the solution of more general minimization tasks. We will follow a more descriptive and less mathematically formal path.

According to Lemma 8.2 and since the proximal operator is a minimizer of (8.105), it must be chosen so that

$$\mathbf{0} \in \partial f(v) + \frac{1}{\lambda} v - \frac{1}{\lambda} x, \quad (8.122)$$

or

$$\mathbf{0} \in \lambda \partial f(v) + v - x, \quad (8.123)$$

or

$$x \in \lambda \partial f(v) + v. \quad (8.124)$$

Let us now define the mapping

$$(I + \lambda \partial f) : \mathbb{R}^l \mapsto \mathbb{R}^l, \quad (8.125)$$

such that

$$(I + \lambda \partial f)(v) = v + \lambda \partial f(v). \quad (8.126)$$

Note that this mapping is one-to-many, due to the definition of the subdifferential, which is a set.⁹ However, its inverse mapping, denoted as

$$(I + \lambda \partial f)^{-1} : \mathbb{R}^l \longmapsto \mathbb{R}^l, \quad (8.127)$$

is single-valued, and as a matter of fact it coincides with the proximal operator; this is readily deduced from (8.124), which can equivalently be written as,

$$\mathbf{x} \in (I + \lambda \partial f)(\mathbf{v}),$$

which implies that

$$(I + \lambda \partial f)^{-1}(\mathbf{x}) = \mathbf{v} = \text{Prox}_{\lambda f}(\mathbf{x}). \quad (8.128)$$

However, we know that the proximal operator is unique. The operator in (8.127) is known as the *resolvent of the subdifferential mapping* [69].

As an exercise, let us now apply (8.128) to the case of Example 8.11. For this case, the subdifferential set is a singleton comprising the gradient vector,

$$\text{Prox}_{\lambda f}(\mathbf{x}) = (I + \lambda \nabla f)^{-1}(\mathbf{x}) \implies (I + \lambda \nabla f)(\text{Prox}_{\lambda f}(\mathbf{x})) = \mathbf{x},$$

or by definition of the mapping $(I + \lambda \nabla f)(\cdot)$ and taking the gradient of the quadratic function,

$$\text{Prox}_{\lambda f}(\mathbf{x}) + \lambda \nabla f(\text{Prox}_{\lambda f}(\mathbf{x})) = \text{Prox}_{\lambda f}(\mathbf{x}) + \lambda A \text{Prox}_{\lambda f}(\mathbf{x}) - \lambda \mathbf{b} = \mathbf{x},$$

which finally results in

$$\text{Prox}_{\lambda f}(\mathbf{x}) = \left(A + \frac{1}{\lambda} I \right)^{-1} \left(\mathbf{b} + \frac{1}{\lambda} \mathbf{x} \right).$$

8.14 PROXIMAL SPLITTING METHODS FOR OPTIMIZATION

A number of optimization tasks often comes in the form of a summation of individual convex functions, some of them being differentiable and some of them nonsmooth. Sparsity-aware learning tasks are typical examples that have received a lot of attention recently, where the regularizing term is nonsmooth, for example, the ℓ_1 norm.

Our goal in this section is to solve the following minimization task

$$\mathbf{x}_* = \arg \min_{\mathbf{x}} \left\{ f(\mathbf{x}) + g(\mathbf{x}) \right\}, \quad (8.129)$$

where both involved functions are convex,

$$f : \mathbb{R}^l \longmapsto \mathbb{R} \cup \{+\infty\}, \quad g : \mathbb{R}^l \longmapsto \mathbb{R},$$

and g is assumed to be differentiable while f is a nonsmooth one. It turns out that the following iterative scheme

$$\mathbf{x}_k = \underbrace{\text{Prox}_{\lambda_k f}}_{\text{backward step}} \underbrace{(\mathbf{x}_{k-1} - \lambda_k \nabla g(\mathbf{x}_{k-1}))}_{\text{forward step}},$$

(8.130)

⁹ A point-to-set mapping is also called a relation on \mathbb{R}^l .

converges to a minimizer of the sum of the involved functions, that is,

$$\mathbf{x}_k \longrightarrow \arg \min_{\mathbf{x}} \{f(\mathbf{x}) + g(\mathbf{x})\}, \quad (8.131)$$

for a properly chosen sequence, λ_k , and provided that the gradient is continuous Lipschitz, that is,

$$\|\nabla g(\mathbf{x}) - \nabla g(\mathbf{y})\| \leq \gamma \|\mathbf{x} - \mathbf{y}\|, \quad (8.132)$$

for some $\gamma > 0$. It can be shown that if $\lambda_k \in \left(0, \frac{1}{\gamma}\right]$, then the algorithm converges to a minimizer at a sublinear rate, $\mathcal{O}(1/k)$ [7]. This family of algorithms is known as *proximal gradient* or *forward-backward splitting* algorithms. The term *splitting* is inherited from the split of the function into two (or more generally into more) parts. The term *proximal* indicates the presence of the proximal operator of f in the optimization scheme. The iteration involves an (explicit) forward gradient computation step performed on the smooth part and an (implicit) backward step via the use of the proximal operator of the nonsmooth part. The terms *forward-backward* are borrowed from numerical analysis methods involving discretization techniques [92]. Proximal gradient schemes are traced back in, for example, [17, 53], but their spread in machine learning and signal processing matured later on [26, 36].

There are a number of variants of the previous basic scheme. A version that achieves $\mathcal{O}(\frac{1}{k^2})$ rate of convergence is based on the classical Nesterov's modification of the gradient algorithm [62], and it is summarized in [Algorithm 8.5](#), [7]. In the algorithm, the update is split into two parts. In the proximal operator, one uses a smoother version of the obtained estimates, using an averaging that involves previous estimates.

Algorithm 8.5 (Fast proximal gradient splitting algorithm).

- Initialization
 - Select $\mathbf{x}_0, \mathbf{z}_1 = \mathbf{x}_0, t_1 = 1$.
 - Select λ .
- **For** $k = 1, 2, \dots$, **Do**
 - $\mathbf{y}_k = \mathbf{z}_k - \lambda \nabla g(\mathbf{z}_k)$
 - $\mathbf{x}_k = \text{Prox}_{\lambda f}(\mathbf{y}_k)$
 - $t_{k+1} = \frac{1 + \sqrt{4t_k^2 + 1}}{2}$
 - $\mu_k = 1 + \frac{t_k - 1}{t_{k+1}}$
 - $\mathbf{z}_{k+1} = \mathbf{x}_k + \mu_k(\mathbf{x}_k - \mathbf{x}_{k-1})$
- **End For**

Note that the algorithm involves a step-size μ_k . The computation of the variables t_k is done in such a way so that convergence speed is optimized. However, it has to be noted that convergence of the scheme is no more guaranteed, in general.

The proximal forward-backward splitting operator

From a first look, the iterative update given in (8.130) seems to be a bit “magic.” However, this is not the case and we can come to it by following simple arguments starting from the basic property of a minimizer. Indeed, let \mathbf{x}_* be a minimizer of (8.129). Then, we know that it has to satisfy

$$\mathbf{0} \in \partial f(\mathbf{x}_*) + \nabla g(\mathbf{x}_*), \text{ or equivalently}$$

$$\mathbf{0} \in \lambda \partial f(\mathbf{x}_*) + \lambda \nabla g(\mathbf{x}_*), \text{ or equivalently}$$

$$\mathbf{0} \in \lambda \partial f(\mathbf{x}_*) + \mathbf{x}_* - \mathbf{x}_* + \lambda \nabla g(\mathbf{x}_*),$$

or equivalently

$$(I - \lambda \nabla g)(x_*) \in (I + \lambda \partial f)(x_*),$$

or

$$(I + \lambda \partial f)^{-1}(I - \lambda \nabla g)(x_*) = x_*,$$

and finally

$$x_* = \text{Prox}_{\lambda f}(I - \lambda \nabla g(x_*)). \quad (8.133)$$

In other words, a minimizer of the task is a fixed point of the operator

$$(I + \lambda \partial f)^{-1}(I - \lambda \nabla g) : \mathbb{R}^l \longmapsto \mathbb{R}^l. \quad (8.134)$$

The latter is known as the *proximal forward-backward splitting operator* and it can be shown that if $\lambda \in (0, \frac{1}{\gamma}]$, where γ is the Lipschitz constant, then this operator is *nonexpansive* [103]. This short story justifies the reason that the iteration in (8.130) is attracted toward the set of minimizers.

Remarks 8.11.

- The proximal gradient splitting algorithm can be considered as a generalization of some previously considered algorithms. If we set $f(x) = \iota_C(x)$, the proximal operator becomes the projection operator and the projected gradient algorithm of (8.62) results. If $f(x) = 0$, we obtain the gradient algorithm and if $g(x) = 0$ the proximal point algorithm comes up.
- Besides batch proximal splitting algorithms, online schemes have been proposed, see [101, 102], [36, 47], with an emphasis on the ℓ_1 regularization tasks.
- The application and development of novel versions of this family of algorithms in the fields of machine learning and signal processing is still an ongoing field of research and the interested reader can delve deeper into the field, via [18, 28, 64, 103].

Alternating direction method of multipliers (ADMM)

Extensions of the proximal splitting gradient algorithm for the case where both functions, f and g , are nonsmooth have also been developed, such as the *Douglas-Rachford* algorithm [27, 53]. Here, we are going to focus on one of the most popular schemes known as the *alternating direction method of multipliers* (ADMM) algorithm [39].

The ADMM algorithm is based on the notion of the *augmented Lagrangian* and at its very heart lies the Lagrangian duality concept (Appendix C).

The goal is to minimize the sum $f(x) + g(x)$, where both f and g can be nonsmooth. This equivalently can be written as

$$\text{minimize with respect to } x, y \quad f(x) + g(y), \quad (8.135)$$

$$\text{subject to} \quad x - y = \mathbf{0}. \quad (8.136)$$

The augmented Lagrangian is defined as

$$L_\lambda(x, y, z) := f(x) + g(y) + \frac{1}{\lambda} z^T(x - y) + \frac{1}{2\lambda} \|x - y\|^2, \quad (8.137)$$

where we have denoted the corresponding Lagrange multipliers¹⁰ by z . The previous equation can be rewritten as

$$L_\lambda(\mathbf{x}, \mathbf{y}, \mathbf{z}) := f(\mathbf{x}) + g(\mathbf{y}) + \frac{1}{2\lambda} \|\mathbf{x} - \mathbf{y} + \mathbf{z}\|^2 - \frac{1}{2\lambda} \|\mathbf{z}\|^2. \quad (8.138)$$

The ADMM is given in [Algorithm 8.6](#).

Algorithm 8.6 (The ADMM algorithm).

- Initialization
 - Fix $\lambda > 0$.
 - Select $\mathbf{y}_0, \mathbf{z}_0$.
- **For** $k = 1, 2, \dots$, **Do**
 - $\mathbf{x}_k = \text{prox}_{\lambda f}(\mathbf{y}_{k-1} - \mathbf{z}_{k-1})$
 - $\mathbf{y}_k = \text{prox}_{\lambda g}(\mathbf{x}_k + \mathbf{z}_{k-1})$
 - $\mathbf{z}_k = \mathbf{z}_{k-1} + (\mathbf{x}_k - \mathbf{y}_k)$
- **End For**

Looking carefully at the algorithm and (8.138), observe that the first recursion corresponds to the minimization of the augmented Lagrangian with respect to \mathbf{x} , keeping the \mathbf{y} and \mathbf{z} fixed from the previous iteration. The second recursion corresponds to the minimization with respect to \mathbf{y} , by keeping \mathbf{x} and \mathbf{z} frozen to their currently available estimates. The last iteration is an update of the *dual* variables (Lagrange multipliers) in the *ascent* direction; note that the difference in the parentheses is the gradient of the augmented Lagrangian with respect to \mathbf{z} . Recall from Appendix C, that the saddle point is found as a max-min problem of the primal (\mathbf{x}, \mathbf{y}) and the dual variables. The convergence of the algorithm has been analyzed in [39]. For related tutorial papers the reader can look in Refs. [15, 45].

Mirror descent algorithms

A closely related algorithmic family to the forward-backward optimization algorithms is traced back to the work in [61] and it is known as *mirror descent algorithms* (MDA). The method has undergone a number of evolutionary steps, for example, [8, 63]. Our focus will be on adopting online schemes to minimize the regularized expected loss function

$$J(\boldsymbol{\theta}) = \mathbb{E} [\mathcal{L}(\boldsymbol{\theta}, \mathbf{y}, \mathbf{x})] + \phi(\boldsymbol{\theta}),$$

where the regularizing function, ϕ , is assumed to be convex, but not necessarily a smooth one. In a recent representative of this algorithmic class, known also as *regularized dual averaging* (ARD) algorithm [95], the main iterative equation is expressed as

$$\boldsymbol{\theta}_n = \min_{\boldsymbol{\theta}} \left\{ \langle \bar{\mathcal{L}}', \boldsymbol{\theta} \rangle + \phi(\boldsymbol{\theta}) + \mu_n \psi(\boldsymbol{\theta}) \right\}, \quad (8.139)$$

where ψ is a *strongly convex* auxiliary function. For example, one possibility is to choose $\phi(\boldsymbol{\theta}) = \lambda \|\boldsymbol{\theta}\|_1$ and $\psi(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$, [95]. $\bar{\mathcal{L}}'$ denotes the average subgradient of \mathcal{L} up to and including time instant $n-1$, that is,

$$\bar{\mathcal{L}}' = \frac{1}{n-1} \sum_{j=1}^{n-1} \mathcal{L}'_j(\boldsymbol{\theta}_j),$$

where $\mathcal{L}_j(\boldsymbol{\theta}) := \mathcal{L}(\boldsymbol{\theta}, \mathbf{y}_j, \mathbf{x}_j)$.

¹⁰ In the book, we have used λ for the Lagrange multipliers. However, here, we have already reserved λ for the proximal operator.

It can be shown that if the subgradients are bounded, and $\mu_n = \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$, then following regret analysis arguments an $\mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$ convergence rate is achieved. If, on the other hand, the regularizing term is strongly convex and $\mu_n = \mathcal{O}\left(\frac{\ln n}{n}\right)$, then an $\mathcal{O}\left(\frac{\ln n}{n}\right)$ rate is obtained. In [95], different variants are proposed. One is based on Nesterov's arguments, as used in [Algorithm 8.5](#), which achieves an $\mathcal{O}\left(\frac{1}{n^2}\right)$ convergence rate.

A closer observation of (8.139) reveals that it can be considered as a generalization of the recursion given in (8.130). Indeed, let us set in (8.139)

$$\psi(\boldsymbol{\theta}) = \frac{1}{2} \|\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}\|^2,$$

and in place of the average gradient consider the most recent value, \mathcal{L}'_{n-1} . Then, (8.139) becomes equivalent to

$$\boldsymbol{\theta} \in \mathcal{L}'_{n-1} + \partial\phi(\boldsymbol{\theta}) + \mu_n(\boldsymbol{\theta} - \boldsymbol{\theta}_{n-1}). \quad (8.140)$$

This is the same relation that would result from (8.130), if we set

$$f \rightarrow \phi, \mathbf{x}_k \rightarrow \boldsymbol{\theta}_n, \mathbf{x}_{k-1} \rightarrow \boldsymbol{\theta}_{n-1}, g \rightarrow \mathcal{L}, \lambda_k \rightarrow \frac{1}{\mu_n}. \quad (8.141)$$

As a matter of fact, using these substitutions and setting $\phi(\cdot) = \|\cdot\|_1$, the FOBOS algorithm, cited before as an example of an online forward-backward scheme [36], results. However, in the case of (8.139) one has the luxury of using other functions in place of the squared Euclidean distance from $\boldsymbol{\theta}_{n-1}$.

A popular auxiliary function that has been exploited is the *Bregman divergence*. The Bregman divergence with respect to a function, say ψ , between two points \mathbf{x}, \mathbf{y} , is defined as

$$B_\psi(\mathbf{x}, \mathbf{y}) = \psi(\mathbf{x}) - \psi(\mathbf{y}) - \langle \nabla\psi(\mathbf{y}), \mathbf{x} - \mathbf{y} \rangle : \text{ Bregman Divergence.} \quad (8.142)$$

It is left as a simple exercise to verify that the Euclidean distance results as the Bregman divergence if $\psi(\mathbf{x}) = \|\mathbf{x}\|^2$.

Another algorithmic variant is the so-called *composite mirror descent*, which employs the currently available estimate of the subgradient, instead of the average, combined with the Bregman divergence; that is, $\bar{\mathcal{L}}$ is replaced by \mathcal{L}'_{n-1} and $\psi(\boldsymbol{\theta})$ by $B_\psi(\boldsymbol{\theta}, \boldsymbol{\theta}_{n-1})$ for some function ψ , [37]. In [38], a time-varying ψ_n is involved by using a weighted average of the Euclidean norm, as pointed out already in [Section 8.10.3](#). Note that in these modifications, although they may look simple, the analysis of the respective algorithms can be quite hard and substantial differences can be obtained in the performance.

At the time this book was being compiled, this area was still a hot topic of research, and it was still too early to draw definite conclusions. It may turn out, as it is often the case, that different algorithms are better suited for different applications and data sets.

PROBLEMS

8.1 Prove the Cauchy-Schwarz inequality in a general Hilbert space.

8.2 Show (a) that the set of points in a Hilbert space \mathbb{H} ,

$$C = \{\mathbf{x} : \|\mathbf{x}\| \leq 1\}$$

is a convex set, and (b) the set of points

$$C = \{x : \|x\| = 1\}$$

is a nonconvex one.

8.3 Show the first order convexity condition.

8.4 Show that a function f is convex, if the one-dimensional function,

$$g(t) := f(x + ty),$$

is convex, $\forall x, y$ in the domain of definition of f .

8.5 Show the second order convexity condition.

Hint. Show the claim first for the one-dimensional case, and then use the result of the previous problem for the generalization.

8.6 Show that a function

$$f : \mathbb{R}^l \mapsto \mathbb{R}$$

is convex iff its epigraph is convex.

8.7 Show that if a function is convex, then its lower level set is convex for any ξ .

8.8 Show that in a Hilbert space, \mathbb{H} , the parallelogram rule,

$$\|x + y\|^2 + \|x - y\|^2 = 2(\|x\|^2 + \|y\|^2), \quad \forall x, y \in \mathbb{H}.$$

holds true.

8.9 Show that if $x, y \in \mathbb{H}$, where \mathbb{H} is a Hilbert space, then the induced by the inner product norm satisfies the triangle inequality, as required by any norm, that is,

$$\|x + y\| \leq \|x\| + \|y\|$$

8.10 Show that if a point x_* is a local minimizer of a convex function, it is necessarily a global one.

Moreover, it is the unique minimizer if the function is strictly convex.

8.11 Let C be a closed convex set in a Hilbert space, \mathbb{H} . Then show that $\forall x \in \mathbb{H}$, there exists a point, denoted as $P_C(x) \in C$, such that

$$\|x - P_C(x)\| = \min_{y \in C} \|x - y\|.$$

8.12 Show that the projection of a point $x \in \mathbb{H}$ onto a nonempty closed convex set, $C \subset \mathbb{H}$, lies on the boundary of C .

8.13 Derive the formula for the projection onto a hyperplane in a (real) Hilbert space, \mathbb{H} .

8.14 Derive the formula for the projection onto a closed ball, $B[\mathbf{0}, \delta]$.

8.15 Find an example of a point whose projection on the ℓ_1 ball is not unique.

8.16 Show that if $C \subset \mathbb{H}$, is a closed convex set in a Hilbert space, then $\forall x \in \mathbb{H}$ and $\forall y \in C$, the projection $P_C(x)$ satisfies the following properties:

- $\text{Real}\{\langle x - P_C(x), y - P_C(x) \rangle\} \leq 0$.
- $\|P_C(x) - P_C(y)\|^2 \leq \text{Real}\{\langle x - y, P_C(x) - P_C(y) \rangle\}$.

8.17 Prove that if S is a closed subspace $S \subset \mathbb{H}$ in a Hilbert space \mathbb{H} , then $\forall x, y \in \mathbb{H}$,

$$\langle x, P_S(y) \rangle = \langle P_S(x), y \rangle = \langle P_S(x), P_S(y) \rangle.$$

and

$$P_S(ax + by) = aP_S(x) + bP_S(y).$$

Hint. Use the result of Problem 8.18.

- 8.18** Let S be a closed convex subspace in a Hilbert space \mathbb{H} , $S \subset \mathbb{H}$. Let S^\perp be the set of all elements $x \in \mathbb{H}$ which are orthogonal to S . Then show that, (a) S^\perp is also a closed subspace, (b) $S \cap S^\perp = \{\mathbf{0}\}$, (c) $\mathbb{H} = S \oplus S^\perp$; that is, $\forall x \in \mathbb{H}$, $\exists x_1 \in S$ and $x_2 \in S^\perp : x = x_1 + x_2$, where x_1, x_2 are unique.
- 8.19** Show that the relaxed projection operator is a nonexpansive mapping.
- 8.20** Show that the relaxed projection operator is a strongly attractive mapping.
- 8.21** Give an example of a sequence in a Hilbert space \mathbb{H} , which converges weakly but not strongly.
- 8.22** Prove that if $C_1 \dots C_K$ are closed convex sets in a Hilbert space \mathbb{H} , then the operator

$$T = T_{C_K} \cdots T_{C_1},$$

is a *regular* one; that is,

$$\|T^{n-1}(x) - T^n(x)\| \rightarrow 0, n \rightarrow \infty,$$

where $T^n := TT \dots T$ is the application of T n successive times.

- 8.23** Show the fundamental POCS theorem for the case of closed subspaces in a Hilbert space, \mathbb{H} .
- 8.24** Derive the subdifferential of the metric distance function $d_C(x)$, where C is a closed convex set $C \subseteq \mathbb{R}^l$ and $x \in \mathbb{R}^l$.
- 8.25** Derive the bound in (8.55).
- 8.26** Show that if a function is γ -Lipschitz, then any of its subgradients is bounded.
- 8.27** Show the convergence of the generic projected subgradient algorithm in (8.61).
- 8.28** Derive Eq. (8.99).
- 8.29** Consider the online version of PDMb in (8.64), that is,

$$\theta_n = \begin{cases} P_C\left(\theta_{n-1} - \mu_n \frac{J(\theta_{n-1})}{\|J'(\theta_{n-1})\|^2} J'(\theta_{n-1})\right), & \text{if } J'(\theta_{n-1}) \neq \mathbf{0}, \\ P_C(\theta_{n-1}), & \text{if } J'(\theta_{n-1}) = \mathbf{0}, \end{cases} \quad (8.143)$$

where we have assumed that $J_* = 0$. If this is not the case, a shift can accommodate for the difference. Thus, we assume that we know the minimum. For example, this is the case for a number tasks, such as the hinge loss function, assuming linearly separable classes, or the linear ϵ -insensitive loss function, for bounded noise. Assume that

$$\mathcal{L}_n(\theta) = \sum_{k=n-q+1}^n \frac{\omega_k d_{C_k}(\theta_{n-1})}{\sum_{k=n-q+1}^n \omega_k d_{C_k}(\theta_{n-1})} d_{C_k}(\theta).$$

Then derive that APSM algorithm of (8.39).

- 8.30** Derive the regret bound for the subgradient algorithm in (8.82).
- 8.31** Show that a function $f(x)$ is σ -strongly convex if and only if the function $f(x) - \frac{\sigma}{2} \|x\|^2$ is convex.
- 8.32** Show that if the loss function is σ -strongly convex, then if $\mu_n = \frac{1}{\sigma n}$, the regret bound for the subgradient algorithm becomes

$$\frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(\theta_{n-1}) \leq \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(\theta_*) + \frac{G^2(1 + \ln N)}{2\sigma N}. \quad (8.144)$$

- 8.33** Consider a batch algorithm that computes the minimum of the empirical loss function, $\theta_*(N)$, having a quadratic convergence rate, that is,

$$\ln \ln \frac{1}{\|\theta^{(i)} - \theta_*(N)\|^2} \sim i.$$

Show that an online algorithm, running for n time instants so that to spend the same computational processing resources as the batch one, achieves for large values of N better performance than the batch algorithm, shown as [11]

$$\|\theta_n - \theta_*\|^2 \sim \frac{1}{N \ln \ln N} \ll \frac{1}{N} \sim \|\theta_*(N) - \theta_*\|^2.$$

Hint. Use the fact that

$$\|\theta_n - \theta_*\|^2 \sim \frac{1}{n}, \quad \text{and} \quad \|\theta_*(N) - \theta_*\|^2 \sim \frac{1}{N}.$$

- 8.34** Show property (8.110) for the proximal operator.

- 8.35** Show property (8.111) for the proximal operator.

- 8.36** Prove that the recursion in (8.117) converges to a minimizer of f .

- 8.37** Derive (8.121) from (8.120).

MATLAB Exercises

- 8.38** Consider the regression model,

$$y_n = \theta_o^T \mathbf{x}_n + \eta_n,$$

where $\theta_o \in \mathbb{R}^{200}$ ($l = 200$) and the coefficients of the unknown vector are obtained randomly via the Gaussian distribution $\mathcal{N}(0, 1)$. The noise samples are also i.i.d. having zero mean and variance $\sigma_\eta^2 = 0.01$. The input sequence is a white noise one, i.i.d. generated via the Gaussian, $\mathcal{N}(0, 1)$.

Using as training data the samples, $(y_n, \mathbf{x}_n) \in \mathbb{R} \times \mathbb{R}^{200}$, $n = 1, 2, \dots$, run the APA (Algorithm 5.2), the NLMS (Algorithm 5.3), the RLS (Algorithm 6.1) and the APSM (Algorithm 8.2) algorithms to estimate the unknown θ_o .

For the APA algorithm, choose $\mu = 0.2$, $\delta = 0.001$, and $q = 30$. For the APSM $\mu = 0.5 \times M_n$, $\epsilon = \sqrt{2}\sigma$ and $q = 30$. Furthermore, in the NLMS set $\mu = 1.2$ and $\delta = 0.001$. Finally, for the RLS set the forgetting factor β is equal to 1. Run 100 independent experiments and plot the average error per iteration in dBs, that is, $10 \log_{10}(e_n^2)$, where $e_n^2 = (y_n - \mathbf{x}_n^T \theta_{n-1})^2$. Compare the performance of the algorithms.

Keep the same parameters, but alter the noise variance so that it becomes 0.3. Plot the average error per iteration as in the previous experiment. What do you observe regarding the performance of the APA compared to the previous low-noise scenario?

Keep playing with different parameters and study the effect on the convergence speed and the error floor in which the algorithms converge.

- 8.39** Create an ad hoc network, having 10 nodes and a total number of 32 connections. Generate at each node the data, which adhere to the following model:

$$y_k(n) = \theta_o^T \mathbf{x}_k(n) + \eta_k(n), \quad k = 1, \dots, 10.$$

The unknown vector $\boldsymbol{\theta}_o \in \mathbb{R}^{60}$ and its coefficients are generated randomly via the Gaussian $\mathcal{N}(0, 1)$. The input vectors are i.i.d. and follow a $\mathcal{N}(0, 1)$. Moreover, the noise samples are i.i.d. generated from zero mean Gaussians with variances corresponding to different signal-to-noise levels, varying randomly from 20 to 25 dBs from node to node.

For the unknown vector estimation employ the combine-then-adapt diffusion APSM ([Algorithm 8.3](#)), the adapt-then-combine LMS ([Algorithm 5.7](#)), the combine-then-adapt LMS ([Algorithm 5.8](#)), and the noncooperative LMS ([Algorithm 5.1](#)). For the combine-then-adapt APSM set $\mu_n = 0.5 \times M_n$, $\epsilon_k = \sqrt{2}\sigma_k$, and $q = 20$. For the adapt-then-combine, combine-then-adapt and noncooperative LMS set the step-size equal to 0.03. Finally, choose the combination weights a_{mk} with respect to the Metropolis rule ([Remark 5.4](#)).

Run 100 independent experiments and plot the average MSD per iteration in dBs, that is,

$$\text{MSD}(n) = 10 \log_{10} \left(\frac{1}{K} \sum_{k=1}^K \|\boldsymbol{\theta}_k(n) - \boldsymbol{\theta}_o\|^2 \right).$$

Compare the performance of the combine-then-adapt APSM with the performance of the LMS-based algorithms.

Keep playing with different parameters for the involved algorithms and observe their influence on the obtained performance.

- 8.40** Download the *banknote authentication* dataset.¹¹ Develop a Matlab program that implements the PEGASOS algorithm for classification ([Algorithm 8.4](#)). Keep 90% of the data as a training set and the remaining 10% as a test set. Set $\lambda = 0.1$ and $m = 1, 10, 30$. Once the training phase has been completed, freeze the parameters for the obtained classifier. Compute the classification error on the test set using the obtained classifier. Compare the classification error for the three choices of m .

8.15 APPENDIX TO CHAPTER 8

In this appendix, we summarize some basic definitions and theorems concerning Hilbert spaces and convex analysis. No proofs are provided and the interested reader may consult more specialized books, given in the references.

Definition 8.11 (Linear Spaces). A nonempty set of elements, V , is called *linear space* if there are defined two operations, *addition* and *scalar multiplication*, so that the following properties hold true:

- $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$, $\forall \mathbf{x}, \mathbf{y} \in V$.
- $(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$, $\forall \mathbf{x}, \mathbf{y}, \mathbf{z} \in V$.
- There exists an element $\mathbf{0} \in V$, known as the *zero vector*, such that, $\forall \mathbf{x} \in V$, $\mathbf{x} + \mathbf{0} = \mathbf{x}$.
- $\forall \mathbf{x} \in V$, there is an element $\mathbf{y} \in V$ such that

$$\mathbf{x} + \mathbf{y} = \mathbf{0}.$$

- For each pair of scalars, $\alpha, \beta \in \mathbb{C}$ and $\forall \mathbf{x}, \mathbf{y} \in V$,

$$(\alpha\beta)\mathbf{x} = \alpha(\beta\mathbf{x}) \text{ and } \alpha(\mathbf{x} + \mathbf{y}) = \alpha\mathbf{x} + \alpha\mathbf{y}.$$

¹¹ <https://archive.ics.uci.edu/ml/datasets/banknote+authentication>.

- For each pair of scalars, $\alpha, \beta \in \mathbb{C}$, $\forall \mathbf{x} \in V$

$$(\alpha + \beta)\mathbf{x} = \alpha\mathbf{x} + \beta\mathbf{x}.$$

- $\forall \mathbf{x} \in V$, and the scalar $1 \in \mathbb{C}$,

$$1\mathbf{x} = \mathbf{x}.$$

Linear spaces are sometimes called *vector spaces* and the elements in V vectors. If the scalars α, β are restricted in \mathbb{R} , then the linear space is known as *real linear space*; otherwise if $\alpha, \beta \in \mathbb{C}$, the linear space is known as *complex linear space*.

Example 8.12 (The Vector Space \mathbb{R}^l). The set of l -tuples $\mathbf{x} := (x_1, \dots, x_l)$, $x_i \in \mathbb{R}$, $i = 1, 2, \dots, l$, is a real vector space, where the addition and scalar multiplication are defined as

$$\begin{aligned}\mathbf{x} + \mathbf{y} &= (x_1 + y_1, \dots, x_l + y_l) \\ a\mathbf{x} &= (ax_1, \dots, ax_l), \quad a \in \mathbb{R}.\end{aligned}$$

Example 8.13. Let the set of all real functions

$$\mathcal{F}(\mathbb{R}) = \{f := f(x) | f : \mathbb{R} \mapsto \mathbb{R}\},$$

where f denotes the “entire” function rather than the value at a specific point $x \in \mathbb{R}$. Then $\mathcal{F}(\mathbb{R})$ is a real linear space with respect to the following operations,

$$(f + h)(x) = f(x) + h(x), \quad \forall f, h \in \mathcal{F}(\mathbb{R}),$$

and

$$(af)(x) = af(x), \quad \forall f \in \mathcal{F}(\mathbb{R}), \quad a \in \mathbb{R}.$$

Definition 8.12. Let V be a linear space and S a nonempty set, $S \subseteq V$. Then S is called a *subspace* of V , if

- $\forall \mathbf{x}, \mathbf{y} \in S$, $\mathbf{x} + \mathbf{y} \in S$.
- $\forall a \in \mathbb{C}$, and $\mathbf{x} \in S$, $a\mathbf{x} \in S$.

Definition 8.13 (Linear Independency). Let V be a linear space and $S \subseteq V$. S is said to be *linearly dependent*, if there exist a *finite* number of *distinct* elements, $\mathbf{x}_k \in S$, $k = 1, 2, \dots, K$, such that

$$\sum_{k=1}^K a_k \mathbf{x}_k = \mathbf{0},$$

for some combination of scalars $a_k \in \mathbb{C}$, $k = 1, 2, \dots, K$, which are *not all zero*. If this is not the case, the set S is known as *linearly independent*.

Let S be a nonempty set, $S \subseteq V$. The set of all possible linear combinations, denoted as $\text{span}\{S\}$,

$$\text{span}\{S\} = \left\{ \mathbf{x} : \mathbf{x} = \sum_{k=1}^K a_k \mathbf{x}_k \mid \mathbf{x}_k \in S, K \in \mathbb{N} \right\}$$

and it is known as the *span* of S . Note that $\text{span}\{S\}$ is always a subspace of V . Moreover, if $\text{span}\{S\} = V$, we say that S spans the linear space V .

Definition 8.14 (Bases). Let V be a linear space and $S \subseteq V$. The set S is known as *basis* of V , if and only if:

- S is linearly independent
- S spans V .

If the number of elements comprising S is finite, we say that V is *finite dimensional* and the number of distinct elements of S defines the *dimension* of V . If the number of elements in S is not finite, we say that V is *infinite dimensional*. Note that there is not a unique basis in V . However, any basis of V has the same number of elements. Moreover, it has been shown that every linear space has a basis. This is known as Zorn's lemma. However, finding a basis is not necessarily a trivial task. The dimension of \mathbb{R}^l is l , and the linear space $\mathcal{F}(\mathbb{R})$ is infinite dimensional.

Definition 8.15 (Inner Product). Let V be a linear space. The *inner product* is a function

$$f : V \times V \mapsto \mathbb{C},$$

which assigns a value in \mathbb{C} , denoted as $\langle x, y \rangle$, to every point of elements $x, y \in V$, with the following properties:

- $\langle x, x \rangle \geq 0$, and $\langle x, x \rangle = 0$ iff $x = 0$.
- $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$.
- $\langle ax, y \rangle = a\langle x, y \rangle$.
- $\langle x, y \rangle = \langle y, x \rangle^*$.

where $*$ denotes complex conjugation. A space where an inner product operation has been defined is known as an *inner product space*.

A direct consequence of the previous inner product properties are the following,

$$\langle x, y + z \rangle = \langle x, y \rangle + \langle x, z \rangle,$$

and

$$\langle x, ay \rangle = a^* \langle x, y \rangle.$$

Example 8.14. Let us consider the vector space \mathbb{C}^l . Then the operation

$$\langle x, y \rangle := \sum_{i=1}^l x_i y_i^* = y^H x, \quad (8.145)$$

is an inner product with $x, y \in \mathbb{C}^l$.

Definition 8.16 (Norm and Normed Spaces). Let V be a linear space. A *norm* is a function

$$f : V \mapsto [0, \infty),$$

that assigns a nonnegative real number to any $x \in V$, it is denoted as $\|x\|$, and it has the following properties:

- $\|x\| \geq 0$, and $\|x\| = 0$ iff $x = \mathbf{0}$.
- $\|ax\| = |a|\|x\|$, $\forall a \in \mathbb{C}$, and $x \in V$.
- $\|x + y\| \leq \|x\| + \|y\|$, $\forall x, y \in V$.

The vector space, V , together with its norm $\|\cdot\|$ is known as a *normed linear space*. The third property is known as the *triangle inequality*.

Given a linear space, one can define different norms. Take, for example, the vector space \mathbb{C}^l . Then define the ℓ_p norm as

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^l |x_i|^p \right)^{\frac{1}{p}}, \quad p \geq 1.$$

It can be shown that the above definition complies with all the required properties for a function to be a norm. For $p = 1$, we refer to the ℓ_1 norm and for $p = 2$ it is known as the *Euclidean norm* or ℓ_2 norm. Note that the latter one results from the inner product operation as defined in (8.145), that is,

$$\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^H \mathbf{x}}. \quad (8.146)$$

This is valid for any inner product linear space. That is, given an inner product linear space, V , with $\langle \cdot, \cdot \rangle$, then the inner product operation induces a norm, that is,

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}, \quad \forall \mathbf{x} \in V.$$

Theorem 8.7 (Cauchy-Schwarz Inequality). *Let V be an inner product space and $\|\cdot\|$ the induced by the inner product norm. Then*

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\| \|\mathbf{y}\|, \quad \forall \mathbf{x}, \mathbf{y} \in V : \quad \text{Cauchy-Schwarz Inequality.} \quad (8.147)$$

This is one of among the most fundamental and important properties in the theory of linear spaces.

A direct consequence of the Cauchy-Schwarz inequality are the following properties:

Given a inner vector space and its induced norm $\|\cdot\|$, then

- $$|\|\mathbf{x}\| - \|\mathbf{y}\|| \leq \|\mathbf{x} - \mathbf{y}\|. \quad (8.148)$$
-

$$\|\mathbf{x} + \mathbf{y}\|^2 + \|\mathbf{x} - \mathbf{y}\|^2 = 2(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2), \quad (8.149)$$

The latter is known as the *parallelogram law*. Note that, all these properties, which may be known from the basic geometry, are valid for *any* linear space, even for infinite dimensional ones.

Example 8.15 (The ℓ^2 space). This is the linear space of all sequences

$$\mathbf{x} = (x_1, x_2, \dots, x_n, \dots),$$

with inner product operation

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{n=1}^{\infty} x_n y_n^*,$$

whose inner product induced norm satisfies the property

$$\|\mathbf{x}\| := \sqrt{\sum_{n=1}^{\infty} |x_n|^2} < \infty.$$

Example 8.16 (The L^2 Space). This is the linear space of all integrable functions

$$f : \mathbb{R} \mapsto \mathbb{R},$$

with inner product operation

$$\langle f, h \rangle := \int_{-\infty}^{\infty} f(x)h(x)dx,$$

whose inner product induced norm satisfies the property

$$\|f\| := \sqrt{\int_{-\infty}^{\infty} |f(x)|^2 dx} < +\infty.$$

Definition 8.17 (Convergence, Cauchy Sequences, and Complete Spaces). Let V be a normed linear space, and let a sequence of elements in V , $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n, \dots$. The sequence is said to *converge* to \mathbf{x} if

$$\lim_{n \rightarrow \infty} \|\mathbf{x}_n - \mathbf{x}\| = 0.$$

Note that if \mathbf{x} exists, it is unique and is known as the limit of \mathbf{x}_n .

A sequence of elements in a normed linear space V is called a *Cauchy sequence* if it satisfies

$$\lim_{n,m \rightarrow \infty} \|\mathbf{x}_n - \mathbf{x}_m\| = 0.$$

In other words, the norm of the difference of *any* two elements in the sequence eventually becomes zero. It turns out that any convergent sequence is Cauchy, but the opposite is not always true.

A normed linear space, V , in which every Cauchy sequence converges in V is said to be *complete*.

Note that any finite dimensional linear space is complete. However, this is not always true for infinite dimensional spaces. Intuitively, a space is complete if there are no points missing from it (including points at the boundary or the interior). Note that a complete space is something stronger than a closed space. In a closed subspace, every convergent sequence has its limit point in the subspace.

Definition 8.18 (Hilbert Spaces). An inner product space, which is complete with respect to the norm induced by the inner product, is called a *Hilbert space*.

Examples of Hilbert spaces are the ℓ^2 and L^2 spaces. Also, the vector spaces \mathbb{C}^l and \mathbb{R}^l , equipped with the inner product operation in (8.145) and the Euclidean norm in (8.146), known as Euclidean spaces are special finite dimensional cases of Hilbert spaces. The spaces ℓ^2 and L^2 are of infinite dimensionality. Note that \mathbb{C}^l , equipped with the ℓ_p norm, $p \neq 2$, is not a Hilbert space, since such a norm is not induced by an inner product.

Definition 8.19 (Weak Convergence). Let \mathbf{x}_n be a sequence of elements/points in a Hilbert space \mathbb{H} . We say that \mathbf{x}_n converges *weakly* to a point $\mathbf{x} \in \mathbb{H}$, if

$$\langle \mathbf{x}_n, \mathbf{y} \rangle \xrightarrow[n \rightarrow \infty]{} \langle \mathbf{x}, \mathbf{y} \rangle, \quad \forall \mathbf{y} \in \mathbb{H},$$

and we write

$$\mathbf{x}_n \xrightarrow[n \rightarrow \infty]{w} \mathbf{x}.$$

Theorem 8.8. Let \mathbb{H} be a Hilbert space. Then the following hold true:

- If $\mathbf{x}_n \xrightarrow[n \rightarrow \infty]{} \mathbf{x}$, then $\mathbf{x}_n \xrightarrow[n \rightarrow \infty]{w} \mathbf{x}$.
- If \mathbb{H} is of finite dimensionality, then $\mathbf{x}_n \xrightarrow[n \rightarrow \infty]{w} \mathbf{x}$, implies $\mathbf{x}_n \xrightarrow[n \rightarrow \infty]{} \mathbf{x}$.

Thus, in a Euclidean space \mathbb{C}^l , weak convergence and convergence coincide.

REFERENCES

- [1] A. Agarwal, P. Bartlett, P. Ravikumar, M.J. Wainwright, Information-theoretic lower bounds on the oracle complexity of convex optimization, *IEEE Trans. Inform. Theory* 58(5) (2012) 3235-3249.
- [2] A.E. Albert, L.A. Gardner, *Stochastic Approximation and Nonlinear Regression*, MIT Press, 1967.
- [3] S. Amari, Natural gradient works efficiently in learning, *Neural Comput.* 10(2) (1998) 251-276.
- [4] F. Bach, E. Moulines, Non-strongly-convex smooth stochastic approximation with convergence rate $O(1/n)$, arXiv:1306.2119v1[cs.LG], 2013.
- [5] F. Bach, Adaptivity of averaged stochastic gradient descent to local strong convexity for logistic regression, *J. Machine Learn. Res.* 15 (2014) 595-627.
- [6] H.H. Bauschke, J.M. Borwein, On projection algorithms for solving convex feasibility problems, *SIAM Rev.* 38(3) (1996) 367-426.
- [7] A. Beck, M. Teboulle, Gradient-based algorithms with applications to signal recovery problems, in: D. Palomar, Y. Eldar (Eds.), *Convex Optimization in Signal Processing and Communications*, Cambridge University Press, 2010, pp. 42-88.
- [8] A. Beck, M. Teboulle, Mirror descent and nonlinear projected subgradient methods for convex optimization, *Operat. Res. Lett.* 31 (2003) 167-175.
- [9] D.P. Bertsekas, *Nonlinear Programming*, second ed., Athena Scientific, 1999.
- [10] D.P. Bertsekas, A. Nedic, A.E. Ozdaglar, *Convex Analysis and Optimization*, Athena Scientific, 2003.
- [11] L. Bottou, Y. Le Cun, Large scale online learning, *Advances in Neural Information Processing Systems*, NIPS, MIT Press, 2003, pp. 2004-2011.
- [12] L. Bottou, O. Bousquet, The tradeoffs of large scale learning, *Adv. Neural Inform. Process. Syst.* 20 (2007) 161-168.
- [13] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Y. Lechevallier, G. Saporta (Eds.), *Proceedings 19th Intl. Conference on Computational Statistics, COMPSTAT 2010*, Paris, France, Springer, 2010.
- [14] S. Boyd, L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
- [15] S. Boyd, N. Parikh, E. Chu, P. Peleato, J. Eckstein, Distributed optimization and statistical learning via the alternating direction method of multipliers, *Found. Trends Machine Learn.* 3(1) (2011) 1122, NOW.
- [16] L.M. Bregman, The method of successive projections for finding a common point of convex sets, *Soviet Math. Dokl.* 6 (1965) 688-692.
- [17] R. Bruck, An iterative solution of a variational inequality for certain monotone operator in a Hilbert space, *Bull. Amer. Math. Soc.* 81(5) (1975) 890-892.
- [18] H.H. Bauschke, P.L. Combettes, *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*, Springer, 2011.
- [19] N. Cesa-Bianchi, A. Conconi, C. Gentile, On the generalization ability of on-line learning algorithms, *IEEE Trans. Inform. Theory*, 50(9) (2004) 2050-2057.
- [20] R.L.G. Cavalcante, I. Yamada, B. Mulgrew, An adaptive projected subgradient approach to learning in diffusion networks, *IEEE Trans. Signal Process.* 57(7) (2009) 2762-2774.
- [21] N. Cesa-Bianchi, G. Lugosi, *Prediction, Learning, and Games*, Cambridge University Press, 2006.
- [22] S. Chouvardas, K. Slavakis, S. Theodoridis, Adaptive robust distributed learning in diffusion sensor networks, *IEEE Trans. Signal Process.* 59(10) (2011) 4692-4707.
- [23] S. Chouvardas, K. Slavakis, S. Theodoridis, I. Yamada, Stochastic analysis of hyperslab-based adaptive projected subgradient method under boundary noise, *IEEE Signal Process. Lett.* 20(7) (2013) 729-732.
- [24] P.L. Combettes, The foundations of set theoretic estimation, *Proc. IEEE* 81(2) (1993) 182-208.
- [25] P.L. Combettes, Inconsistent signal feasibility problems: least-squares solutions in a product space, *IEEE Trans. Signal Process.* 42(11) (1994) 2955-2966.

- [26] P.L. Combettes, V.R. Wajs, Signal recovery by proximal forward-backward splitting, *Multiscale Model. Simulat.* 4 (2005) 1168-1200.
- [27] P.L. Combettes, J.-C. Pesquet, A Douglas-Rachford splitting approach to nonsmooth convex variational signal recovery, *IEEE J. Select. Topics Signal Process.* 1 (2007) 564-574.
- [28] P.L. Combettes, J.-C. Pesquet, Proximal splitting methods in signal processing, in: H.H. Bauschke, R.S. Burachik, P.L. Combettes, V. Elser, D.R. Luke, H. Wolkowicz (Eds.), *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, Springer-Verlag, 2011, pp. 185-212.
- [29] J.R. Deller, Set-membersip identification in digital signal processing, *IEEE Signal Process. Mag.* 6 (1989) 4-20.
- [30] F. Deutsch, *Best Approximation in Inner Product Spaces*, CMS, Springer, 2000.
- [31] L. Devroye, L. Györfi, G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, Springer, 1991.
- [32] P.S.R. Diniz, S. Werner, Set-membership binormalized data-reusing LMS algorithms, *IEEE Trans. Signal Process.* 52(1) (2003) 124-134.
- [33] P.S.R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*, fourth ed., Springer Verlag, 2014.
- [34] P.S.R. Diniz, Convergence performance of the simplified set-membership affine projection algorithm, *J. Circuits Syst. Signal Process.* 30(2) (2011) 439-462.
- [35] J. Duchi, S.S. Shwartz, Y. Singer, T. Chandra, Efficient projections onto the ℓ_1 -ball for learning in high dimensions, in: *Proceedings of the International Conference on Machine Learning (ICML)*, 2008, pp. 272-279.
- [36] J. Duchi, Y. Singer, Efficient online and batch learning using forward backward splitting, *J. Machine Learn. Res.* 10 (2009) 2899-2934.
- [37] J. Duchi, S. Shalev-Shwartz, Y. Singer, A. Tewari, Composite objective mirror descent, in: *Proceedings of the 23rd Annual Conference on Computational Learning Theory*, 2010.
- [38] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, *J. Machine Learn. Res.* 12 (2011) 2121-2159.
- [39] M. Fortin, R. Glowinski, *Augmented Lagrangian Methods: Applications to the Numerical Solution of Boundary-Value Problems*, Elsevier Science, Amsterdam; North-Holland, 1983.
- [40] A.A. Goldstein, Convex Programming in Hilbert spaces, *Bull. Amer. Math. Soc.* 70(5) (1964) 709-710.
- [41] L.G. Gubin, B.T. Polyak, E.V. Raik, The method of projections for finding the common point of convex sets, *USSR Comput. Math. Phys.* 7(6) (1967) 1-24.
- [42] E. Hazan, A. Agarwal, S. Kale, Logarithmic regret algorithms for online convex optimization, *Machine Learn.* 69(2-3) (2007) 169-192.
- [43] J.B. Hiriart-Urruty, C. Lemarechal, *Convex Analysis and Minimization Algorithms*, Springer-Verlag, Berlin, 1993.
- [44] J. Kivinen, A.J. Smola, R.C. Williamson, Online learning with kernels, *IEEE Trans. Signal Process.* 52(8) (2004) 2165-2176.
- [45] N. Komodakis, J.-C. Pesquet, Playing with duality: An overview of recent primal-dual approaches for solving large scale optimization problems, *arXiv:1406.5429v1 [cs.NA]* 20 June 2014, 2014.
- [46] Y. Kopsinis, K. Slavakis, S. Theodoridis, Online sparse system identification and signal reconstruction using projections onto weighted ℓ_1 -balls, *IEEE Trans. Signal Process.* 59(3) (2011) 936-952.
- [47] J. Langford, L. Li, T. Zhang, Sparse online learning via truncated gradient, *J. Machine Learn. Res.* 10 (2009) 747-776.
- [48] Y. LeCun, L. Bottou, G.B. Orr, K.R. Müller, Efficient BackProp, in: G.B. Orr, K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade*, Springer, 1998, pp. 9-50.
- [49] N. Le Roux, M. Schmidt, F. Bach, A stochastic gradient method with an exponential convergence rate for finite training sets, *arXiv:1202.6258v4 [math.OC]*, 2013.

- [50] W.S. Lee, P.L. Bartlett, R.C. Williamson, The importance of convexity in learning with squared loss, *IEEE Trans. Informat. Theory* 44(5) (1998) 1974-1980.
- [51] B.S. Levitin, B.T. Polyak, Constrained minimization methods, *Zhurnal Vychishitel noi Matematik Matematicheskoi Fiziki* 6(5) (1966) 787-823.
- [52] D.D. Lewis, Y. Yang, T.G. Rose, F. Li, RCV1: a new benchmark collection for text categorization research, *J. Machine Learn. Res.* 5 (2004) 361-397.
- [53] P. Lions, B. Mercier, Splitting algorithms for the sum of two nonlinear operators, *SIAM J. Numer. Anal.* 16 (1979) 964-979.
- [54] P.E. Maingé, Strong convergence of projected subgradient methods for nonsmooth and nonstrictly convex minimization, *Set-Valued Anal.* 16 (2008) 899-912.
- [55] N. Murata, S. Amari, Statistical analysis of learning dynamics, *Signal Process.* 74(1) (1999) 3-28.
- [56] B. Martinet, Regularisation d' inéquations variationnelles par approximations successives, *Revue Francaise de Informatique et Recherche Operationelle* 4 (1970) 154-158.
- [57] C. Moler, Iterative refinement in floating point, *J. ACM* 14(2) (1967) 316-321.
- [58] J.J. Moreau, Fonctions convexes duales et points proximaux dans un espace Hilbertien, *Rep. Paris Acad. Sci. A* 255 (1962) 2897-2899.
- [59] J.J. Moreau, Proximité et dualité dans un espace hilbertien, *Bull. Soc. Math. France* 93 (1965) 273-299.
- [60] S. Nagaraj, S. Gollamudi, S. Kapoor, Y.F. Huang, BEACON: An adaptive set-membership filtering technique with sparse updates, *IEEE Trans. Signal Process.* 47(11) (1999) 2928-2941.
- [61] A. Nemirovsky, D. Yudin, *Problem Complexity and Method Efficiency in Optimization*, J. Wiley & Sons, New York, 1983.
- [62] Y.E. Nesterov, A method of solving a convex programming problem with convergence rate $O(1/k^2)$, *Soviet Math. Dokl.* 27 (2) (1983) 372-376.
- [63] Y. Nesterov, Primal-dual subgradient methods for convex problems, *Math. Program.* 120(1) (2009) 221-259.
- [64] N. Parish, S. Boyd, Proximal Algorithms, *Found. Trends Optim.* 1(2) (2013) 123-231.
- [65] G. Pierra, Decomposition through formalization in a product space, *Math. Program.* 8 (1984) 96-115.
- [66] T. Poggio, S. Voinea, L. Rosasco, Online learning, stability and stochastic gradient descent, arXiv: 1105.4701 [cs.LG], Sep 2011.
- [67] B.T. Polyak, Minimization of unsmooth functionals, *Zhurnal Vychishitel noi Matematik Matematicheskoi Fiziki* 9(3) (1969) 509-521.
- [68] R.T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, NJ, 1970.
- [69] R.T. Rockafellar, Monotone operators and the proximal point algorithm, *SIAM J. Control Optim.* 14 (1976) 877-898.
- [70] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, PEGASOS: primal estimated sub-gradient solver for SVM, *Math. Programm. B* 127 (2011) 3-30.
- [71] N.Z. Shor, *On the Structure of Algorithms for the Numerical Solution of Optimal Planning and Design Problems*, PhD Thesis, Cybernetics Institute, Academy of Sciences, Kiev, 1964.
- [72] N.Z. Shor, *Minimization Methods for Non-differentiable Functions*, Springer Series in Computational Mathematics, Springer, 1985.
- [73] K. Slavakis, I. Yamada, N. Ogura, The adaptive projected subgradient method over the fixed point set of strongly attracting nonexpansive mappings, *Numer. Funct. Anal. Optim.* 27(7-8) (2006) 905-930.
- [74] K. Slavakis, I. Yamada, Robust wideband beamforming by the hybrid steepest descent method, *IEEE Trans. Signal Process.* 55(9) (2007) 4511-4522.
- [75] K. Slavakis, S. Theodoridis, I. Yamada, Online classification using kernels and projection-based adaptive algorithms, *IEEE Trans. Signal Process.* 56(7) (2008) 2781-2797.
- [76] K. Slavakis, S. Theodoridis, Sliding window generalized kernel affine projection algorithm using projection mappings, *EURASIP J. Adv. Signal Process.* 2008, Article ID 830381, 2008. doi: 10.1155/2008/830381.

- [77] K. Slavakis, S. Theodoridis, I. Yamada, Adaptive constrained learning in reproducing kernel Hilbert spaces, *IEEE Trans. Signal Process.* 5(12) (2009) 4744-4764.
- [78] K. Slavakis, I. Yamada, The adaptive projected subgradient method constrained by families of quasi-non-expansive mappings and its application to online learning, *SIAM J. Optim.* 23(1) (2013) 126-152.
- [79] K. Slavakis, A. Bouboulis, S. Theodoridis, Online learning in reproducing kernel Hilbert spaces, in: S. Theodoridis, R. Chellappa (Eds.), *E-reference for Signal Processing*, Academic Press, 2013.
- [80] K. Slavakis, Y. Kopsinis, S. Sheodoridis, S. McLaughlin, Generalized thresholding and online sparsity-aware learning in a union of subspaces, *IEEE Trans. Signal Process.* 61(15) (2013) 3760-3773.
- [81] K. Slavakis, Personal Communication, March 2014.
- [82] H. Stark, Y. Yang, *Vector Space Projections*, John Wiley, 1998.
- [83] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, fourth ed., Academic Press, 2009.
- [84] S. Theodoridis, K. Slavakis, I. Yamada, Adaptive learning in a world of projections, *IEEE Signal Processing Magazine*, Vol 28(1) (97-123, 2011).
- [85] M. Todd, Semidefinite Optimization, *Acta Numerica* 10 (2001) 515-560.
- [86] O.P. Tseng, An incremental gradient (projection) method with momentum term and adaptive step size rule, *SIAM J. Optim.* 8(2) (1998) 506-531.
- [87] A.B. Tsybakov, Optimal aggregation of classifiers in statistical learning, *Ann. Stat.* 32(1) (2004) 135-166.
- [88] Y. Tsyplkin, *Foundation of the Theory of Learning Systems*, Academic Press, 1973.
- [89] V.N. Vapnik, *Estimation of Dependences Based on Empirical Data*, Springer Series in Statistics, Springer-Verlag, Berlin 1982.
- [90] V.N. Vapnik, *Statistical Learning Theory*, John Wiley & Sons, 1998.
- [91] V.N. Vapnik, *The Nature of Statistical Learning Theory*, Springer, 2000.
- [92] R.S. Varga, *Matrix Iterative Analysis*, second ed., Springer-Verlag, New York, 2000.
- [93] J. von Neumann, Functional operators, Vol II. The geometry of orthogonal spaces, *Ann. Math. Stud.* 22, Princeton Univ. Press, NJ, 1950 (Reprint of lecture notes first distributed in 1933).
- [94] S. Werner, P.S.R. Diniz, Set-membership affine projection algorithm, *IEEE Signal Process. Lett.* 8(8) (2001) 231-235.
- [95] L. Xiao, Dual averaging methods for regularized stochastic learning and online optimization, *J. Machine Learn. Res.* 11 (2010) 2543-2596.
- [96] I. Yamada, The hybrid steepest descent method for the variational inequality problem over the intersection of fixed point sets of nonexpansive mappings, *Stud. Comput. Math.* 8 (2001) 473-504.
- [97] I. Yamada, K. Slavakis, K. Yamada, An efficient robust adaptive filtering algorithm based on parallel subgradient projection techniques, *IEEE Trans. Signal Process.* 50(5) (2002) 1091-1101.
- [98] I. Yamada, Adaptive projected subgradient method: a unified view of projection based adaptive algorithms, *J. IEICE* 86(6) (2003) 654-658 (in Japanese).
- [99] I. Yamada, N. Ogura, Adaptive projected subgradient method for asymptotic minimization of nonnegative convex functions, *Numer. Funct. Anal. and Optim.* 25(7-8) (2004) 593-617.
- [100] I. Yamada, N. Ogura, Hybrid steepest descent method for variational inequality problem over the fixed point set of certain quasi-nonexpansive mappings, *Numer. Funct. Anal. Optim.* 25 (2004) 619-655.
- [101] I. Yamada, S. Gandy, M. Yamagishi, Sparsity-aware adaptive filtering based on Douglas-Rachford splitting, in: Proceedings, 19th European Signal Processing Conference (EUSIPCO), Barcelona, Spain, 2011.
- [102] M. Yamagishi, M. Yukawa, I. Yamada, Acceleration of adaptive proximal forward-backward slitting method and its application in systems identification, in: Proceedings IEEE International Conference on Acoustics Speech and Signal processing, ICASSP, Prague, 2011.
- [103] I. Yamada, M. Yukawa, M. Yamagishi, Minimizing the Moreau envelope of non-smooth convex functions over the fixed point set of certain quasi-nonexpansive mappings, in: H.H. Bauschke, R.S. Burachik, P.L. Combettes, V. Elser, D.R. Luke, H. Wolkowicz (Eds.), *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, Springer, New York, 2011, pp. 345-390.

- [104] K. Yosida, *Functional Analysis*, Springer, 1968.
- [105] T. Zhang, Solving large scale linear prediction problems using stochastic gradient descent algorithms, in: Proceedings of the 21st International Conference on Machine Learning, ICML, Banff, Alberta, Canada, 2004, pp. 919-926.
- [106] M. Zinkevich, Online convex programming and generalized infinitesimal gradient ascent, in: Proceedings of the 20th International Conference on Machine Learning (ICML), Washington, DC, 2003.

SPARSITY-AWARE LEARNING: CONCEPTS AND THEORETICAL FOUNDATIONS

CHAPTER OUTLINE

9.1	Introduction	409
9.2	Searching for a Norm	410
9.3	The Least Absolute Shrinkage and Selection Operator (LASSO).....	413
9.4	Sparse Signal Representation	417
9.5	In Search of the Sparsest Solution	421
	<i>The ℓ_2 Norm Minimizer</i>	423
	<i>The ℓ_0 Norm Minimizer</i>	423
	<i>The ℓ_1 Norm Minimizer</i>	424
	<i>Characterization of the ℓ_1 Norm Minimizer</i>	425
	<i>Geometric Interpretation.....</i>	425
9.6	Uniqueness of the ℓ_0 Minimizer	428
9.6.1	Mutual Coherence	430
9.7	Equivalence of ℓ_0 and ℓ_1 Minimizers: Sufficiency Conditions.....	432
9.7.1	Condition Implied by the Mutual Coherence Number	432
9.7.2	The Restricted Isometry Property (RIP)	433
	<i>Constructing Matrices that Obey the RIP of Order k.....</i>	434
9.8	Robust Sparse Signal Recovery from Noisy Measurements	435
9.9	Compressed Sensing: The Glory of Randomness	436
	<i>Compressed Sensing</i>	437
9.9.1	Dimensionality Reduction and Stable Embeddings.....	439
9.9.2	Sub-Nyquist Sampling: Analog-to-Information Conversion	440
9.10	A Case Study: Image De-Noising	444
Problems		446
	<i>MATLAB Exercises</i>	448
References		450

9.1 INTRODUCTION

In Chapter 3, the notion of regularization was introduced as a tool to address a number of problems that are usually encountered in machine learning. Improving the performance of an estimator by shrinking the norm of the minimum variance unbiased (MVU) estimator, guarding against overfitting,

coping with ill-conditioning, and providing a solution to an underdetermined set of equations are some notable examples where regularization has provided successful answers. Some of the advantages were demonstrated via the ridge regression concept, where the least-squares (LS) cost function was combined, in a tradeoff rationale, with the squared Euclidean norm of the desired solution.

In this and the next chapter, our interest will be on alternatives to the Euclidean norm, and in particular the focus will revolve around the ℓ_1 norm; this is the sum of the absolute values of the components comprising a vector. Although seeking a solution to a problem via the ℓ_1 norm regularization of a cost function has been known and used since the 1970s, it is only recently that it has become the focus of attention of a massive volume of research in the context of compressed sensing. At the heart of this problem lies an underdetermined set of linear equations, which, in general, accepts an infinite number of solutions. However, in a number of cases, an extra piece of information is available: the true model, whose estimate we want to obtain, is sparse; that is, only a few of its coordinates are nonzero. It turns out that a large number of commonly used applications can be cast under such a scenario and can benefit by sparse modeling.

Besides its practical significance, sparsity-aware learning has offered to the scientific community novel theoretical tools and solutions to problems that only a few years ago seemed intractable. This is also a reason that this is an interdisciplinary field of research encompassing scientists from, for example, mathematics, statistics, machine learning, and signal processing. Moreover, it has already been applied in many areas, ranging from biomedicine to communications and astronomy. At the time this book was compiled, there was a “research happening” in the field, which posed some difficulties in assembling related material. We made an effort to present in a unifying way the basic notions and ideas that run across this field. Our goal is to provide the reader with an overview of the major contributions that have taken place in the theoretical and algorithmic fronts and have been consolidated as a distinct scientific area.

In the current chapter, the focus is on presenting the main concepts and theoretical foundations related to sparsity-aware learning techniques. We start by reviewing various norms, then we move on to establish conditions on the recovery of sparse vectors, or vectors that are sparse in a transform domain, using less observations than the dimension of the corresponding space. Geometry plays an important part in our approach. Finally, some theoretical advances that tie sparsity and sampling theory are presented. At the end of the chapter, a case study concerning image de-noising is discussed.

9.2 SEARCHING FOR A NORM

Mathematicians have been very imaginative in proposing various norms in order to equip linear spaces. Among the most popular norms used in functional analysis are the ℓ_p norms. To tailor things to our needs, given a vector $\theta \in \mathbb{R}^l$, its ℓ_p norm is defined as

$$\|\theta\|_p := \left(\sum_{i=1}^l |\theta_i|^p \right)^{1/p}. \quad (9.1)$$

For $p = 2$, the Euclidean or ℓ_2 norm is obtained, and for $p = 1$, (9.1) results in the ℓ_1 norm, that is,

$$\|\theta\|_1 = \sum_{i=1}^l |\theta_i|. \quad (9.2)$$

If we let $p \rightarrow \infty$, then we get the ℓ_∞ norm; let $|\theta_{i_{\max}}| := \max\{|\theta_1|, |\theta_2|, \dots, |\theta_l|\}$, and notice that

$$\|\boldsymbol{\theta}\|_\infty := \lim_{p \rightarrow \infty} \left(|\theta_{i_{\max}}|^p \sum_{i=1}^l \left(\frac{|\theta_i|}{|\theta_{i_{\max}}|} \right)^p \right)^{1/p} = |\theta_{i_{\max}}|, \quad (9.3)$$

that is, $\|\boldsymbol{\theta}\|_\infty$ is equal to the maximum of the absolute values of the coordinates of $\boldsymbol{\theta}$. One can show that all the ℓ_p norms are true norms for $p \geq 1$; that is, satisfy all four requirements that a function $\mathbb{R}^l \mapsto [0, \infty)$ must respect in order to be called a norm, that is,

1. $\|\boldsymbol{\theta}\|_p \geq 0$,
2. $\|\boldsymbol{\theta}\|_p = 0 \Leftrightarrow \boldsymbol{\theta} = \mathbf{0}$,
3. $\|\alpha\boldsymbol{\theta}\|_p = |\alpha| \|\boldsymbol{\theta}\|_p, \forall \alpha \in \mathbb{R}$,
4. $\|\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2\|_p \leq \|\boldsymbol{\theta}_1\|_p + \|\boldsymbol{\theta}_2\|_p$.

The third condition enforces the norm function to be (*positively*) *homogeneous* and the fourth one is the *triangle inequality*. These properties also guarantee that any function that is a norm is also a convex one (Problem 9.3). Though strictly speaking, if we allow $p > 0$ to take values less than one in (9.1), the resulting function is not a true norm (Problem 9.8), we can still call them norms, albeit knowing that this is an abuse of the definition of a norm. An interesting case, which will be used extensively in this chapter, is the ℓ_0 norm, which can be obtained as the limit, for $p \rightarrow 0$, of

$$\|\boldsymbol{\theta}\|_0 := \lim_{p \rightarrow 0} \|\boldsymbol{\theta}\|_p^p = \lim_{p \rightarrow 0} \sum_{i=1}^l |\theta_i|^p = \sum_{i=1}^l \chi_{(0,\infty)}(|\theta_i|), \quad (9.4)$$

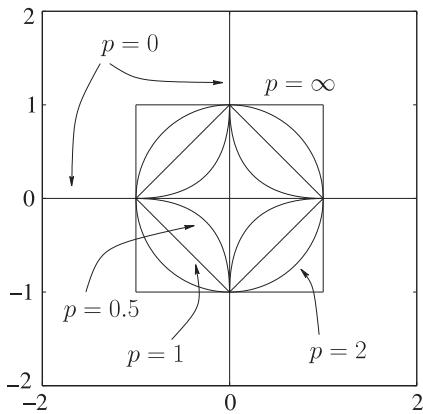
where $\chi_{\mathcal{A}}(\cdot)$ is the characteristic function with respect to a set \mathcal{A} , defined as

$$\chi_{\mathcal{A}}(\tau) := \begin{cases} 1, & \text{if } \tau \in \mathcal{A}, \\ 0, & \text{if } \tau \notin \mathcal{A}. \end{cases}$$

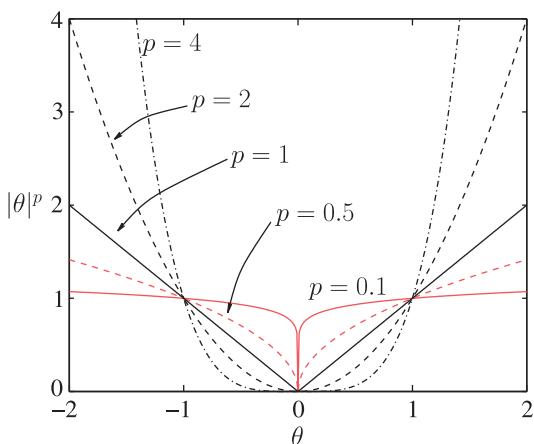
That is, the ℓ_0 norm is equal to the number of nonzero components of the respective vector. It is very easy to check that this function is not a true norm. Indeed, this is not homogeneous, that is, $\|\alpha\boldsymbol{\theta}\|_0 \neq |\alpha| \|\boldsymbol{\theta}\|_0, \forall \alpha \neq 1$. Figure 9.1 shows the isovalue curves, in the two-dimensional space, that correspond to $\|\boldsymbol{\theta}\|_p = \rho \equiv 1$, for $p = 0, 0.5, 1, 2$, and ∞ . Observe that for the Euclidean norm the isovalue curve has the shape of a “ball” and for the ℓ_1 norm the shape of a rhombus. We refer to them as the ℓ_2 and the ℓ_1 balls, respectively, by slightly “abusing” the meaning of a ball.¹ Observe that in the case of the ℓ_0 norm, the isovalue curve comprises both the horizontal and the vertical axes, excluding the $(0, 0)$ element. If we restrict the size of the ℓ_0 norm to be less than one, then the corresponding set of points becomes a singleton, that is, $(0, 0)$. Also, the set of all the two-dimensional points that have ℓ_0 norm less than or equal to two is the \mathbb{R}^2 space. This, slightly “strange” behavior, is a consequence of the discrete nature of this “norm.”

Figure 9.2 shows the graph of $|\cdot|^p$, which is the individual contribution of each component of a vector to the ℓ_p norm, for different values of p . Observe that (a) for $p < 1$, the region that is formed above the graph (epigraph, see Chapter 8) is not a convex one, which verifies what we have already said, that is, the respective function is not a true norm; and (b) for values of the argument $|\theta| > 1$, the

¹ Strictly speaking, a ball must also contain all the points in the interior, that is, all concentric spheres of smaller radius, Chapter 8.

**FIGURE 9.1**

The isovalue curves for $\|\theta\|_p = 1$ and for various values of p , in the two-dimensional space. Observe that for the ℓ_0 norm, the respective values cover the two axes with the exception of the point $(0, 0)$. For the ℓ_1 norm, the isovalue curve is a rhombus, and for the ℓ_2 (Euclidean) norm, it is a circle.

**FIGURE 9.2**

Observe that the epigraph, that is, the region above the graph, is nonconvex for values $p < 1$, indicating the nonconvexity of the respective $|\cdot|^p$ function. The value $p = 1$ is the smallest one for which convexity is retained. Also note that, for large values of $p > 1$, the contribution of small values of $|\theta| < 1$ to the respective norm becomes insignificant.

larger the value of $p \geq 1$ and the larger the value of $|\theta|$, the higher the contribution of the respective component to the norm. Hence, if ℓ_p norms, $p \geq 1$, are used in the regularization method, components with large values become the dominant ones and the optimization algorithm will concentrate on these by penalizing them to get smaller so that the overall cost can be reduced. The opposite is true for values $|\theta| < 1$; ℓ_p , $p > 1$ norms tend to push the contribution of such components to zero. The ℓ_1 norm is the only one (among $p \geq 1$) that retains relatively large values even for small values of $|\theta| < 1$ and, hence, components with small values can still have a say in the optimization process and can be penalized by being pushed to smaller values. Hence, if the ℓ_1 norm is used to replace the ℓ_2 one in (3.39), *only those components of the vector that are really significant in reducing the model misfit measuring term in the regularized cost function will be kept, and the rest will be forced to zero.* The same tendency, yet more aggressive, is true for $0 \leq p < 1$. The extreme case is when one considers the ℓ_0 norm. Even a small increase of a component from zero makes its contribution to the norm large, so the optimizing algorithm has to be very “cautious” in making an element nonzero.

In a nutshell, from all the true norms ($p \geq 1$), the ℓ_1 is the only one that shows respect to small values. The rest of the ℓ_p norms, $p > 1$, just squeeze them to make their values even smaller, and care mainly for the large values. We will return to this point very soon.

9.3 THE LEAST ABSOLUTE SHRINKAGE AND SELECTION OPERATOR (LASSO)

In Chapter 3, we discussed some of the benefits in adopting the regularization method for enhancing the performance of an estimator. In this chapter, we will see and study more reasons that justify the use of regularization. The first one refers to what is known as the *interpretation* power of an estimator. For example, in the regression task, we want to select those components, θ_i , of $\boldsymbol{\theta}$ that have the most important say in the formation of the output variable. This is very important if the number of parameters, l , is large and we want to concentrate on the most important of them. In a classification task, not all features are informative, hence one would like to keep the most informative of them and make the less informative ones equal to zero. Another related problem refers to those cases where we know, *a priori*, that a number of the components of a parameter vector are zero, but we do not know which ones. Now, the discussion at the end of the previous section becomes more meaningful. Can we use, while regularizing, an appropriate norm that can assist the optimization process (a) in unveiling such zeros or (b) to put more emphasis on the most significant of its components, those that play a decisive role in reducing the misfit measuring term in the regularized cost function, and set the rest of them equal to zero? Although the ℓ_p norms, with $p < 1$, seem to be the natural choice for such a regularization, the fact that they are not convex makes the optimization process hard. The ℓ_1 norm is the one that is “closest” to them, yet it retains the computationally attractive property of convexity.

The ℓ_1 norm has been used for such problems for a long time. In the 1970s, it was used in seismology [27, 85], where the reflected signal that indicates changes in the various earth substrates is a sparse one, that is, very few values are relatively large and the rest are small and insignificant. Since then, it has been used to tackle similar problems in different applications (e.g., [40, 80]). However, one can trace two papers that were catalytic in providing the spark for the current strong interest around the ℓ_1 norm. One came from statistics, [88], which addressed the LASSO task (first formulated, to our knowledge, in [80]), to be discussed next, and the other from the signal analysis community, [26], which formulated the *Basis Pursuit*, to be discussed in a later section.

We first address our familiar regression task

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta}, \quad \mathbf{y}, \boldsymbol{\eta} \in \mathbb{R}^N, \quad \boldsymbol{\theta} \in \mathbb{R}^l, \quad N \geq l,$$

and obtain the estimate of the unknown parameter $\boldsymbol{\theta}$ via the LS loss, regularized by the ℓ_1 norm, that is, for $\lambda \geq 0$,

$$\hat{\boldsymbol{\theta}} := \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} L(\boldsymbol{\theta}, \lambda) \quad (9.5)$$

$$\begin{aligned} &:= \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \left(\sum_{n=1}^N (y_n - \mathbf{x}_n^\top \boldsymbol{\theta})^2 + \lambda \|\boldsymbol{\theta}\|_1 \right) \\ &= \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} ((\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1). \end{aligned} \quad (9.6)$$

Following the discussion with respect to the bias term given in [Section 3.8](#) and in order to simplify the analysis, we will assume hereafter, without harming generality, that the data are of zero mean values. If this is not the case, the data can be centered by subtracting their respective sample means.

It turns out that the task in (9.6) can be equivalently written in the following two formulations:

$$\begin{aligned} \hat{\boldsymbol{\theta}} : \quad &\min_{\boldsymbol{\theta} \in \mathbb{R}^l} (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}), \\ \text{s.t.} \quad &\|\boldsymbol{\theta}\|_1 \leq \rho, \end{aligned} \quad (9.7)$$

or

$$\begin{aligned} \hat{\boldsymbol{\theta}} : \quad &\min_{\boldsymbol{\theta} \in \mathbb{R}^l} \|\boldsymbol{\theta}\|_1, \\ \text{s.t.} \quad &(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \leq \epsilon, \end{aligned} \quad (9.8)$$

given the user-defined parameters $\rho, \epsilon \geq 0$. The formulation in (9.7) is known as the LASSO and the one in (9.8) as the *basis pursuit de-noising* (BPDN) (e.g., [\[15\]](#)). All three formulations are equivalent for specific choices of λ, ϵ , and ρ (see, e.g., [\[14\]](#)). Observe that the minimized cost function in (9.6) corresponds to the Lagrangian of the formulation in (9.7). However, this functional dependence among λ, ϵ , and ρ is hard to compute, unless the columns of \mathbf{X} are mutually orthogonal. Moreover, this equivalence does not necessarily imply that all three formulations are equally easy or difficult to solve. As we will see later in this chapter, algorithms have been developed along each one of the previous formulations. From now on, we will refer to all three formulations as the LASSO task, in a slight abuse of the standard terminology, and the specific formulation will be apparent from the context, if not stated explicitly.

We know that ridge regression admits a closed form solution, that is,

$$\hat{\boldsymbol{\theta}}_R = (\mathbf{X}^\top \mathbf{X} + \lambda I)^{-1} \mathbf{X}^\top \mathbf{y}.$$

In contrast, this is not the case for LASSO, and its solution requires iterative techniques. It is straightforward to see that LASSO can be formulated as a standard convex quadratic problem with linear inequalities. Indeed, we can rewrite (9.6) as

$$\begin{aligned} \min_{\{\theta_i, u_i\}_{i=1}^l} \quad &(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) + \lambda \sum_{i=1}^l u_i \\ \text{s.t.} \quad &\begin{cases} -u_i \leq \theta_i \leq u_i, & i = 1, 2, \dots, l, \\ u_i \geq 0, \end{cases} \end{aligned}$$

which can be solved by any standard convex optimization method (e.g., [14, 100]). The reason that developing algorithms for the LASSO has been a hot research topic is due to the emphasis on obtaining *efficient* algorithms by exploiting the specific nature of this task, especially for cases where l is very large, as is often the case in practice.

In order to get better insight into the nature of the solution that is obtained by LASSO, let us assume that the regressors are mutually orthogonal and of unit norm, hence $X^T X = I$. Orthogonality of the input matrix helps to decouple the coordinates and results to l one-dimensional problems that can be solved analytically. For this case, the LS estimate becomes

$$\hat{\theta}_{LS} = (X^T X)^{-1} X^T y = X^T y,$$

and the ridge regression gives

$$\hat{\theta}_R = \frac{1}{1 + \lambda} \hat{\theta}_{LS}, \quad (9.9)$$

that is, every component of the LS estimate is simply shrunk by the *same* factor, $\frac{1}{1+\lambda}$.

In the case of the ℓ_1 regularization, the minimized Lagrangian function is no more differentiable, due to the presence of the absolute values in the ℓ_1 norm. So, in this case, we have to consider the notion of the subdifferential. It is known (Chapter 8) that if the zero vector belongs to the subdifferential set of a convex function at a point, this means that this point corresponds to a minimum of the function. Taking the subdifferential of the Lagrangian defined in (9.6) and recalling that the subdifferential set of a differentiable function includes as its *single* element the respective gradient, the resulting from the ℓ_1 regularized task estimate, $\hat{\theta}_1$, must satisfy

$$\mathbf{0} \in -2X^T y + 2X^T X\theta + \lambda \partial \|\theta\|_1,$$

where ∂ stands for the subdifferential set (Chapter 8). If X has orthonormal columns, the previous equation can be written component-wise as follows:

$$0 \in -\hat{\theta}_{LS,i} + \hat{\theta}_{1,i} + \frac{\lambda}{2} \partial |\hat{\theta}_{1,i}|, \quad \forall i, \quad (9.10)$$

where the subdifferential of the function $|\cdot|$, derived in Example 8.4 (Chapter 8), is given as

$$\partial |\theta| = \begin{cases} \{1\}, & \text{if } \theta > 0, \\ \{-1\}, & \text{if } \theta < 0, \\ [-1, 1], & \text{if } \theta = 0. \end{cases}$$

Thus, we can now write for each component of the LASSO optimal estimate

$$\hat{\theta}_{1,i} = \begin{cases} \hat{\theta}_{LS,i} - \frac{\lambda}{2}, & \text{if } \hat{\theta}_{1,i} > 0, \\ \hat{\theta}_{LS,i} + \frac{\lambda}{2}, & \text{if } \hat{\theta}_{1,i} < 0. \end{cases} \quad (9.11)$$

Notice that (9.11) can only be true if $\hat{\theta}_{LS,i} > \frac{\lambda}{2}$, and (9.12) only if $\hat{\theta}_{LS,i} < -\frac{\lambda}{2}$. Moreover, in the case where $\hat{\theta}_{1,i} = 0$, then (9.10) and the subdifferential of $|\cdot|$ suggest that necessarily $|\hat{\theta}_{LS,i}| \leq \frac{\lambda}{2}$. Concluding, we can write in a more compact way that

$$\hat{\theta}_{1,i} = \text{sgn}(\hat{\theta}_{\text{LS},i}) \left(\left| \hat{\theta}_{\text{LS},i} \right| - \frac{\lambda}{2} \right)_+ : \quad \text{Soft Thresholding Operation,} \quad (9.13)$$

where $(\cdot)_+$ denotes the “positive part” of the respective argument; it is equal to the argument if this is nonnegative, and zero otherwise. This is very interesting indeed. In contrast to the ridge regression that shrinks all coordinates of the unregularized LS solution by the same factor, LASSO forces all coordinates, whose absolute value is less than or equal to $\lambda/2$, to zero, and the rest of the coordinates are reduced, in absolute value, by the same amount $\lambda/2$. This is known as *soft thresholding*, to distinguish it from the *hard thresholding* operation; the latter is defined as $\theta \cdot \chi_{(0,\infty)}(|\theta| - \frac{\lambda}{2})$, $\theta \in \mathbb{R}$, where $\chi_{(0,\infty)}(\cdot)$ stands for the characteristic function with respect to the set $(0, \infty)$. Figure 9.3 shows the graphs illustrating the effect that the ridge regression, LASSO, and hard thresholding have on the unregularized LS solution, as a function of its value (horizontal axis). Note that our discussion here, simplified via the orthonormal input matrix case, has quantified what we said before about the tendency of the ℓ_1 norm to push small values to become *exactly zero*. This will be further strengthened, via a more rigorous mathematical formulation, in Section 9.5.

Example 9.1. Assume that the unregularized LS solution, for a given regression task, $\mathbf{y} = X\boldsymbol{\theta} + \boldsymbol{\eta}$, is given by

$$\hat{\boldsymbol{\theta}}_{\text{LS}} = [0.2, -0.7, 0.8, -0.1, 1.0]^T.$$

Derive the solutions for the corresponding ridge regression and ℓ_1 norm regularization tasks. Assume that the input matrix X has orthonormal columns and that the regularization parameter is $\lambda = 1$. Also, what is the result of hard thresholding the vector $\hat{\boldsymbol{\theta}}_{\text{LS}}$ with threshold equal to 0.5?

We know that the corresponding solution for the ridge regression is

$$\hat{\boldsymbol{\theta}}_R = \frac{1}{1+\lambda} \hat{\boldsymbol{\theta}}_{\text{LS}} = [0.1, -0.35, 0.4, -0.05, 0.5]^T.$$

The solution for the ℓ_1 norm regularization is given by soft thresholding, with threshold equal to $\lambda/2 = 0.5$, hence the corresponding vector is

$$\hat{\boldsymbol{\theta}}_1 = [0, -0.2, 0.3, 0, 0.5]^T.$$

The result of the hard thresholding operation is the vector $[0, -0.7, 0.8, 0, 1.0]^T$.

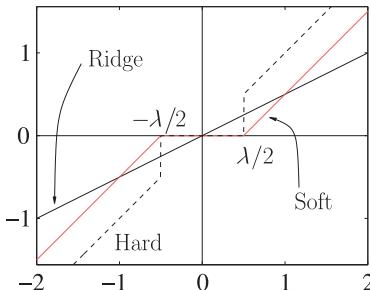


FIGURE 9.3

Output-input curves for the hard thresholding, soft thresholding operators together with the linear operator associated with the ridge regression, for the same value of $\lambda = 1$.

Remarks 9.1.

- The hard and soft thresholding rules are only two possibilities out of a larger number of alternatives. Note that the hard thresholding operation is defined via a discontinuous function, and this makes this rule unstable in the sense of being very sensitive to small changes of the input. Moreover, this shrinking rule tends to exhibit large variance in the resulting estimates. The soft thresholding rule is a continuous function, but, as readily seen from the graph in Figure 9.3, it introduces bias even for the large values of the input argument. In order to ameliorate such shortcomings, a number of alternative thresholding operators have been introduced and studied both theoretically and experimentally. Although these are not within the mainstream of our interest, we provide two popular examples for the sake of completeness—the *smoothly clipped absolute deviation* (SCAD) thresholding rule:

$$\hat{\theta}_{\text{SCAD}} = \begin{cases} \text{sgn}(\theta) (|\theta| - \lambda_{\text{SCAD}})_+, & |\theta| \leq 2\lambda_{\text{SCAD}}, \\ \frac{(\alpha - 1)\theta - \alpha\lambda_{\text{SCAD}} \text{sgn}(\theta)}{\alpha - 2}, & 2\lambda_{\text{SCAD}} < |\theta| \leq \alpha\lambda_{\text{SCAD}}, \\ \theta, & |\theta| > \alpha\lambda_{\text{SCAD}}, \end{cases}$$

and the *nonnegative garrote* thresholding rule:

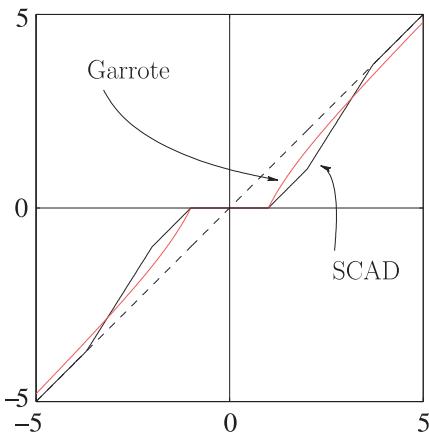
$$\hat{\theta}_{\text{garr}} = \begin{cases} 0, & |\theta| \leq \lambda_{\text{garr}}, \\ \theta - \frac{\lambda_{\text{garr}}^2}{\theta}, & |\theta| > \lambda_{\text{garr}}. \end{cases}$$

Figure 9.4 shows the respective graphs. Observe that, in both cases, an effort has been made to remove the discontinuity (associated with the hard thresholding) and to remove/reduce the bias for large values of the input argument. The parameter $\alpha > 2$ is a user-defined one. For a more detailed discussion on this topic, the interested reader can refer, for example, to [2].

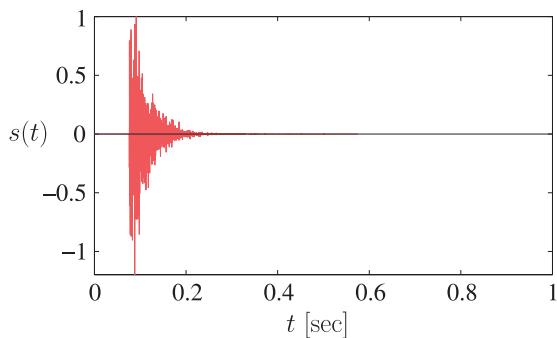
9.4 SPARSE SIGNAL REPRESENTATION

In the previous section, we brought into our discussion the need to take special care for zeros. Sparsity is an attribute that is met in a plethora of natural signals, because nature tends to be parsimonious. The notion of and need for parsimonious models was also discussed in Chapter 3, in the context of inverse problems in machine learning tasks. In this section, we will briefly present a number of application cases where the existence of zeros in a mathematical expansion is of paramount importance, hence, it justifies our search for and development of related analysis tools.

In Chapter 4, we discussed the task of echo cancellation. In a number of cases, the echo path, represented by a vector comprising the values of the impulse response samples, is a sparse one. This is the case, for example, in internet telephony and in acoustic and network environments (e.g., [3, 10, 73]). Figure 9.5 shows the impulse response of such an echo path. The impulse response of the echo path is of short duration; however, the delay with which it appears is not known. So, in order to model it, one has to use a long impulse response, yet only a relatively small number of the coefficients will be significant and the rest will be close to zero. Of course, one could ask, why not use an LMS or an RLS,

**FIGURE 9.4**

Output-input graph for the SCAD and nonnegative garrote rules with parameters $\alpha = 3.7$, and $\lambda_{\text{SCAD}} = \lambda_{\text{garr}} = 1$. Observe that both rules smooth out the discontinuity associated with the hard thresholding rule. Notice, also, that the SCAD rule removes the bias associated with the soft thresholding rule for large values of the input variable. On the contrary, the garrote thresholding rule allows some bias for large input values, which diminishes as λ_{garr} gets smaller and smaller.

**FIGURE 9.5**

The impulse response function of an echo-path in a telephone network. Observe that although it is of relatively short duration, it is not a priori known where exactly in time it will occur.

and eventually the significant coefficients will be identified? The answer is that this turns out not to be the most efficient way to tackle such problems, because the convergence of the algorithm can be very slow. In contrast, if one embeds, somehow, into the problem the a priori information concerning the existence of (almost) zero coefficients, then the convergence speed can be significantly increased and also better error floors can be attained.

A similar situation occurs in wireless communication systems, which involve multipath channels. A typical application is in high-definition television (HDTV) systems that the involved communications

channels consist of a few nonnegligible coefficients, some of which may have quite large time delays with respect to the main signal (see, e.g., [4, 32, 52, 77]). If the information signal is transmitted at high symbol rates through such a dispersive channel, then the introduced intersymbol interference (ISI) has a span of several tens up to hundreds of symbol intervals. This in turn implies that quite long channel estimators are required at the receiver's end in order to reduce effectively the ISI component of the received signal, although only a small part of it has values substantially different to zero. The situation is even more demanding whenever the channel frequency response exhibits deep nulls. More recently, sparsity has been exploited in channel estimation for multicarrier systems, both for single antenna as well as for multiple-input-multiple-output (MIMO) systems [46, 47]. A thorough, in-depth treatment related to sparsity in multipath communication systems is provided in [5].

Another example, which might be more widely known, is that of signal compression. It turns out that if the signal modalities with which we communicate (e.g., speech) and also sense the world (e.g., images, audio) are transformed into a suitably chosen domain then they are sparsely represented; only a relatively small number of the signal components in this domain are large, and the rest are close to zero. As an example, Figure 9.6a shows an image and Figure 9.6b the plot of the magnitude of the obtained discrete cosine transform (DCT) components, which are computed by writing the corresponding image array as a vector in lexicographic order. Note that more than 95% of the total energy is contributed by only 5% of the largest components. This is at the heart of any compression technique. Only the large coefficients are chosen to be coded and the rest are considered to be zero. Hence, significant gains are obtained in memory/bandwidth requirements while storing/transmitting such signals, without much perceptual loss. Depending on the modality, different transforms are used. For example, in JPEG-2000, an image array, represented in terms of a vector that contains the intensity of the gray levels of the image

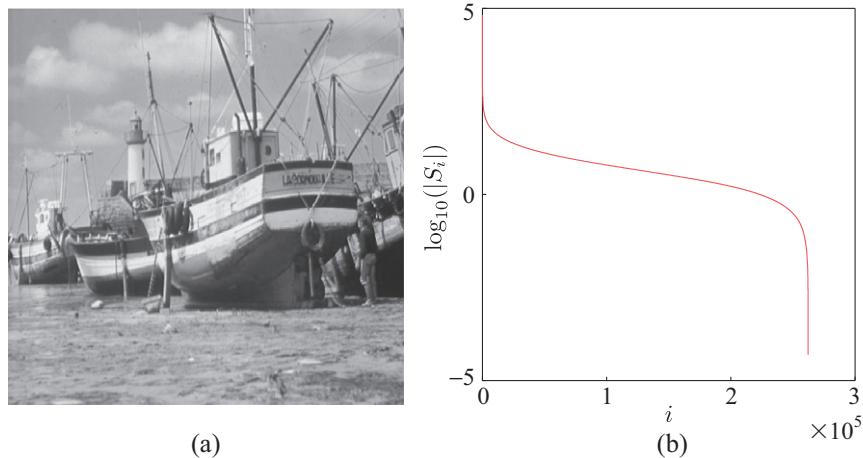


FIGURE 9.6

(a) A 512×512 pixel image and (b) the magnitude of its DCT components in descending order and logarithmic scale. Note that more than 95% of the total energy is contributed by only 5% of the largest components.

pixels, is transformed via the discrete wavelet transform (DWT) and results in a transformed vector that comprises only a few large components.

Let

$$\mathbf{S} = \Phi^H \mathbf{s}, \mathbf{s}, \mathbf{S} \in \mathbb{C}^l, \quad (9.14)$$

where \mathbf{s} is the vector of the “raw” signal samples, \mathbf{S} is the (complex-valued) vector of the transformed ones, and Φ is the $l \times l$ transformation matrix. Often, this is an orthonormal/unitary matrix, $\Phi^H \Phi = I$. Basically, a transform is nothing more than a projection of a vector on a new set of coordinate axes, which comprise the columns of the transformation matrix Φ . Celebrated examples of such transforms are the wavelet, the discrete Fourier (DFT), and the discrete cosine (DCT) transforms (e.g., [86]). In such cases, where the transformation matrix is orthonormal, one can write that

$$\mathbf{s} = \Psi \mathbf{S}, \quad (9.15)$$

where $\Psi = \Phi$. Equation (9.14) is known as the *analysis* and (9.15) as the *synthesis* equation.

Compression via such transforms exploits the fact that many signals in nature, which are rich in context, can be *compactly* represented in an appropriately chosen basis, depending on the modality of the signal. Very often, the construction of such bases tries to “imitate” the sensory systems that the human brain has developed in order to sense these signals; and we know that nature (in contrast to modern humans) does not like to waste resources. A standard compression task comprises the following stages: (a) Obtain the l components of \mathbf{S} via the analysis step (9.14); (b) keep the k most significant of them; (c) code these values, as well as their respective locations in the transformed vector \mathbf{S} ; and (d) obtain the (approximate) original signal \mathbf{s} when needed (after storage or transmission), via the synthesis Eq. (9.15), where in place of \mathbf{S} only its k most significant components are used, which are the ones that were coded, while the rest are set equal to zero. However, there is something unorthodox in this process of compression as it has been practiced until very recently. One processes (transforms) large signal vectors of l coordinates, where l in practice can be quite large, and then uses only a small percentage of the transformed coefficients, while the rest are simply ignored. Moreover, one has to store/transmit the location of the respective large coefficients that are finally coded. A natural question that is raised is the following: Because \mathbf{S} in the synthesis equation is (approximately) sparse, can one compute it via an alternative path than the analysis equation in (9.14)? The issue here is to investigate whether one could use a more informative way of sampling the available raw data so that less than l samples/observations are sufficient to recover all the necessary information. The ideal case would be to recover it via a set of k such samples, because this is the number of the significant free parameters. On the other hand, if this sounds a bit extreme, can one obtain N ($k < N \ll l$) such signal-related measurements, from which \mathbf{s} can eventually be retrieved? It turns out that such an approach is possible and it leads to the solution of an *underdetermined* system of linear equations, under the constraint that the unknown target vector is a sparse one.

The importance of such techniques becomes even more apparent when, instead of an orthonormal basis, as discussed before, a more general type of expansion is adopted, in terms of what is known as *overcomplete dictionaries*. A dictionary [65] is a collection of parameterized waveforms, which are discrete-time signal samples, represented as vectors $\psi_i \in \mathbb{C}^l, i \in \mathcal{I}$, where \mathcal{I} is an integer index set. For example, the columns of a DFT or a DWT matrix comprise a dictionary. These are two examples of what are known as *complete* dictionaries, which consist of l (orthonormal) vectors, that is, a number equal to the length of the signal vector. However, in many cases in practice, using such dictionaries is

very restrictive. Let us take, for example, a segment of audio signal, from a news media or a video, that needs to be processed. This consists, in general, of different types of signals, namely speech, music, and environmental sounds. For each type of these signals, different signal vectors may be more appropriate in the expansion for the analysis. For example, music signals are characterized by a strong harmonic content and the use of sinusoids seems to be best for compression, while for speech signals a Gabor type signal expansion (sinusoids of various frequencies weighted by sufficiently narrow pulses at different locations in time [31, 86]), may be a better choice. The same applies when one deals with an image. Different parts of an image, such as parts that are smooth or contain sharp edges, may demand a different expansion vector set for obtaining the best overall performance. The more recent tendency, in order to satisfy such needs, is to use *overcomplete* dictionaries. Such dictionaries can be obtained, for example, by concatenating different dictionaries together, for example, a DFT and a DWT matrix to result in a combined $l \times 2l$ transformation matrix. Alternatively, a dictionary can be “trained” in order to effectively represent a set of available signal exemplars, a task that is often referred to as dictionary learning [75, 78, 89, 99]. While using such overcomplete dictionaries, the synthesis equation takes the form

$$\mathbf{s} = \sum_{i \in \mathcal{I}} \theta_i \boldsymbol{\psi}_i. \quad (9.16)$$

Note that, now, the analysis is an ill-posed problem, because the elements $\{\boldsymbol{\psi}_i\}_{i \in \mathcal{I}}$ (usually called *atoms*) of the dictionary are not linearly independent, and there is not a unique set of coefficients $\{\theta_i\}_{i \in \mathcal{I}}$ that generates \mathbf{s} . Moreover, we expect most of these coefficients to be (nearly) zero. Note that, in such cases, the cardinality of \mathcal{I} is larger than l . This necessarily leads to underdetermined systems of equations with infinite many solutions. The question that is now raised is whether we can exploit the fact that most of these coefficients are known to be zero, in order to come up with a unique solution. If yes, under which conditions is such a solution possible? We will return to the task of learning dictionaries in [Chapter 19](#).

Besides the previous examples, there are a number of cases where an underdetermined system of equations is the result of our inability to obtain a sufficiently large number of measurements, due to physical and technical constraints. This is the case in MRI imaging, which will be presented in more detail later in the chapter.

9.5 IN SEARCH OF THE SPARSEST SOLUTION

Inspired by the discussion in the previous section, we now turn our attention to the task of solving underdetermined systems of equations by imposing the sparsity constraint on the solution. We will develop the theoretical setup in the context of regression and we will adhere to the notation that has been adopted for this task. Moreover, we will focus on the real-valued data case in order to simplify the presentation. The theory can be readily extended to the more general complex-valued data case (see, e.g., [64, 98]). We assume that we are given a set of observations/measurments, $\mathbf{y} := [y_1, y_2, \dots, y_N]^T \in \mathbb{R}^N$, according to the linear model

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta}, \quad \mathbf{y} \in \mathbb{R}^N, \quad \boldsymbol{\theta} \in \mathbb{R}^l, \quad l > N, \quad (9.17)$$

where \mathbf{X} is the $N \times l$ input matrix, which is assumed to be of full row rank, that is, $\text{rank}(\mathbf{X}) = N$. Our starting point is the noiseless case. The linear system of equations in (9.17) is an underdetermined one

and accepts an infinite number of solutions. The set of possible solutions lies in the intersection of the N hyperplanes² in the l -dimensional space,

$$\left\{ \boldsymbol{\theta} \in \mathbb{R}^l : y_n = \mathbf{x}_n^T \boldsymbol{\theta} \right\}, \quad n = 1, 2, \dots, N.$$

We know from geometry that the intersection of N nonparallel hyperplanes (which in our case is guaranteed by the fact that X has been assumed to be full row rank, hence \mathbf{x}_n , $n = 1, 2, \dots, N$, are linearly independent) is a plane of dimensionality $l - N$ (e.g., the intersection of two (nonparallel) (hyper)planes in the three-dimensional space is a straight line, that is, a plane of dimensionality equal to one). In a more formal way, the set of all possible solutions, to be denoted as Θ , is an *affine* set. An affine set is the translation of a linear subspace by a constant vector. Let us pursue this a bit further, because we will need it later on.

Let the null space of X be the set $\text{null}(X)$ (sometimes, denoted as $\mathcal{N}(X)$), defined as the linear subspace

$$\text{null}(X) = \left\{ \mathbf{z} \in \mathbb{R}^l : X\mathbf{z} = \mathbf{0} \right\}.$$

Obviously, if $\boldsymbol{\theta}_0$ is a solution to (9.17), that is, $\boldsymbol{\theta}_0 \in \Theta$, then it is easy to verify that $\forall \boldsymbol{\theta} \in \Theta, X(\boldsymbol{\theta} - \boldsymbol{\theta}_0) = \mathbf{0}$, or $\boldsymbol{\theta} - \boldsymbol{\theta}_0 \in \text{null}(X)$. As a result,

$$\Theta = \boldsymbol{\theta}_0 + \text{null}(X),$$

and Θ is an affine set. We also know from linear algebra basics (and it is easy to show it; [Problem 9.9](#)), that the null space of a full row rank matrix, $N \times l$, $l > N$, is a subspace of dimensionality $l - N$. [Figure 9.7](#) illustrates the case for one measurement sample in the two-dimensional space, $l = 2$ and

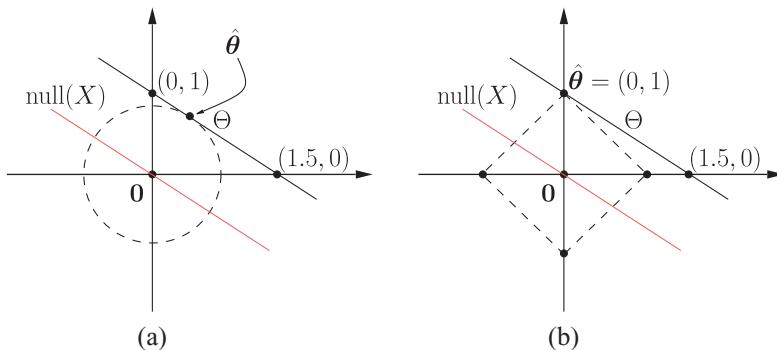


FIGURE 9.7

The set of solutions Θ is an affine set (gray line), which is a translation of the $\text{null}(X)$ subspace (red line). (a) The ℓ_2 norm minimizer. The dotted circle corresponds to the smallest ℓ_2 ball that intersects the set Θ . As such, the intersection point, $\hat{\boldsymbol{\theta}}$, is the ℓ_2 norm minimizer of the task in (9.18). Notice that the vector $\hat{\boldsymbol{\theta}}$ contains no zero component. (b) The ℓ_1 norm minimizer. The dotted rhombus corresponds to the smallest ℓ_1 ball that intersects Θ . Hence, the intersection point, $\hat{\boldsymbol{\theta}}$, is the solution of the constrained ℓ_1 minimization task of (9.21). Notice that the obtained estimate $\hat{\boldsymbol{\theta}} = (0, 1)$ contains a zero.

² In \mathbb{R}^l , a hyperplane is of dimension $l - 1$. A plane has dimension lower than $l - 1$.

$N = 1$. The set of solutions Θ is a straight line, which is the translation of the linear subspace crossing the origin (the $\text{null}(X)$). Therefore, if one wants to select a *single* point among all the points that lie in the affine set of solutions, Θ , then an extra constraint/a priori knowledge has to be imposed.

In the sequel, three such possibilities are examined.

The ℓ_2 norm minimizer

Our goal now becomes to pick a point in (the affine set) Θ that corresponds to the minimum ℓ_2 norm. This is equivalent to solving the following constrained task:

$$\begin{aligned} \min_{\theta \in \mathbb{R}^l} \quad & \|\theta\|_2^2 \\ \text{s.t.} \quad & \mathbf{x}_n^T \theta = y_n, \quad n = 1, 2, \dots, N. \end{aligned} \quad (9.18)$$

We already know from [Section 6.4](#) (and one can rederive it by employing Lagrange multipliers; [Problem 9.10](#)) that the previous optimization task accepts a *unique* solution given in closed form as

$$\hat{\theta} = X^T (XX^T)^{-1} \mathbf{y}. \quad (9.19)$$

The geometric interpretation of this solution is provided in [Figure 9.7a](#), for the case of $l = 2$ and $N = 1$. The radius of the Euclidean norm ball keeps increasing, until it touches the plane that contains the solutions. This point is the one with the minimum ℓ_2 norm or, equivalently, the point that lies closest to the origin. Equivalently, the point $\hat{\theta}$ can be seen as the (metric) projection of $\mathbf{0}$ onto Θ .

Minimizing the ℓ_2 norm in order to solve a linear set of underdetermined equations has been used in various applications. The closest to us is in the context of determining the unknown coefficients in an expansion using an overcomplete dictionary of functions (vectors) [35]. A main drawback of this method is that it is not sparsity preserving. There is no guarantee that the solution in (9.19) will give zeros even if the true model vector θ has zeros. Moreover, the method is *resolution limited* [26]. This means that even if there may be a sharp contribution of specific atoms in the dictionary, this is not portrayed in the obtained solution. This is a consequence of the fact that the information provided by XX^T is a global one, containing all atoms of the dictionary in an “averaging” fashion, and the final result tends to smooth out the individual contributions, especially when the dictionary is overcomplete.

The ℓ_0 norm minimizer

Now we turn our attention to the ℓ_0 norm (once more, it is pointed out that this is an abuse of the definition of the norm, as stated before), and we make sparsity our new flag under which a solution will be obtained. The task now becomes

$$\begin{aligned} \min_{\theta \in \mathbb{R}^l} \quad & \|\theta\|_0 \\ \text{s.t.} \quad & \mathbf{x}_n^T \theta = y_n, \quad n = 1, 2, \dots, N, \end{aligned} \quad (9.20)$$

that is, from all the points that lie on the plane of all possible solutions find the *sparsest* one, that is, the one with the least number of nonzero elements. As a matter of fact, such an approach is within the spirit of *Occam’s razor* rule—it corresponds to the smallest number of parameters that can explain the obtained observations. The points that are now raised are:

- Is a solution to this problem unique, and under which conditions?
- Can a solution be obtained with low enough complexity in realistic time?

We postpone the answer to the first question until later. As for the second one, the news is not good. Minimizing the ℓ_0 norm under a set of linear constraints is a task of combinatorial nature, and as a matter of fact, the problem is, in general, NP-hard [72]. The way to approach the problem is to consider all possible combinations of zeros in $\boldsymbol{\theta}$, removing the respective columns of X in (9.17), and check whether the system of equations is satisfied; keep as solutions the ones with the smallest number of nonzero elements. Such a searching technique exhibits complexity of an exponential dependence on l . Figure 9.7a illustrates the two points ((1.5, 0) and (0, 1)) that comprise the solution set of minimizing the ℓ_0 norm for the single measurement (constraint) case.

The ℓ_1 norm minimizer

The current task is now given by

$$\begin{aligned} \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \quad & \|\boldsymbol{\theta}\|_1 \\ \text{s.t.} \quad & \mathbf{x}_n^T \boldsymbol{\theta} = y_n, \quad n = 1, 2, \dots, N. \end{aligned} \quad (9.21)$$

Figure 9.7b illustrates the geometry. The ℓ_1 ball is increased until it touches the affine set of the possible solutions. For this specific geometry, the solution is the point (0, 1), which is a sparse solution. In our discussion in Section 9.2, we saw that the ℓ_1 norm is the one, out of all $\ell_p, p \geq 1$ norms, that bears some similarity with the sparsity-promoting (nonconvex) $\ell_p, p < 1$ “norms.” Also, we have commented that the ℓ_1 norm encourages zeros when the respective values are small. In the sequel, we will state one lemma that establishes this zero-favoring property in a more formal way. The ℓ_1 norm minimizer is also known as *Basis Pursuit* and it was suggested for decomposing a vector signal in terms of the atoms of an overcomplete dictionary [26].

The ℓ_1 minimizer can be brought into the standard linear programming (LP) form and then can be solved by recalling any related method; the simplex method and the more recent interior point methods are two possibilities (see, e.g., [14, 33]). Indeed, consider the LP task

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T \mathbf{x}, \\ \text{s.t.} \quad & \mathbf{A} \mathbf{x} = \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

To verify that our ℓ_1 minimizer can be cast in the previous form, notice first that any l -dimensional vector $\boldsymbol{\theta}$ can be decomposed as

$$\boldsymbol{\theta} = \mathbf{u} - \mathbf{v}, \quad \mathbf{u} \geq \mathbf{0}, \mathbf{v} \geq \mathbf{0}.$$

Indeed, this holds true if, for example,

$$\mathbf{u} := \boldsymbol{\theta}_+, \quad \mathbf{v} := (-\boldsymbol{\theta})_+,$$

where \mathbf{x}_+ stands for the vector obtained after keeping the positive components of \mathbf{x} and setting the rest equal to zero. Moreover, notice that

$$\|\boldsymbol{\theta}\|_1 = [1, 1, \dots, 1] \begin{bmatrix} \boldsymbol{\theta}_+ \\ (-\boldsymbol{\theta})_+ \end{bmatrix} = [1, 1, \dots, 1] \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix}.$$

Hence, our ℓ_1 minimization task can be recast in the LP form, if

$$\mathbf{c} := [1, 1, \dots, 1]^T, \quad \mathbf{x} := [\mathbf{u}^T, \mathbf{v}^T]^T,$$

$$\mathbf{A} := [X, -X], \quad \mathbf{b} := \mathbf{y}.$$

Characterization of the ℓ_1 norm minimizer

Lemma 9.1. An element θ in the affine set, Θ , of the solutions of the underdetermined linear system (9.17), has minimal ℓ_1 norm if and only if the following condition is satisfied:

$$\left| \sum_{i: \theta_i \neq 0} \operatorname{sgn}(\theta_i) z_i \right| \leq \sum_{i: \theta_i = 0} |z_i|, \quad \forall z \in \operatorname{null}(X). \quad (9.22)$$

Moreover, the ℓ_1 minimizer is unique if and only if the inequality in (9.22) is a strict one for all $z \neq 0$ (see, e.g., [74] and Problem 9.11).

Remarks 9.2.

- The previous lemma has a very interesting and important consequence. If $\hat{\theta}$ is the *unique* minimizer of (9.21), then

$$\operatorname{card}\{i : \hat{\theta}_i = 0\} \geq \dim(\operatorname{null}(X)), \quad (9.23)$$

where $\operatorname{card}\{\cdot\}$ denotes the cardinality of a set. In words, the number of zero coordinates of the unique minimizer cannot be smaller than the dimension of the null space of X . Indeed, if this is not the case, then the unique minimizer could have less zeros than the dimensionality of $\operatorname{null}(X)$. This means that we can always find a $z \in \operatorname{null}(X)$, which has zeros in the same locations where the coordinates of the unique minimizer are zero, and at the same time it is not identically zero, that is, $z \neq 0$ (Problem 9.12). However, this would violate (9.22), which in the case of uniqueness holds as a strict inequality.

Definition 9.1. A vector θ is called k -sparse if it has *at most* k nonzero components.

Remarks 9.3.

- If the minimizer of (9.21) is *unique*, then it is a k -sparse vector with

$$k \leq N.$$

This is a direct consequence of the Remark 9.2, and the fact that for the matrix X ,

$$\dim(\operatorname{null}(X)) = l - \operatorname{rank}(X) = l - N.$$

Hence, the number of the nonzero elements of the unique minimizer must be at most equal to N .

If one resorts to geometry, all the previously stated results become crystal clear.

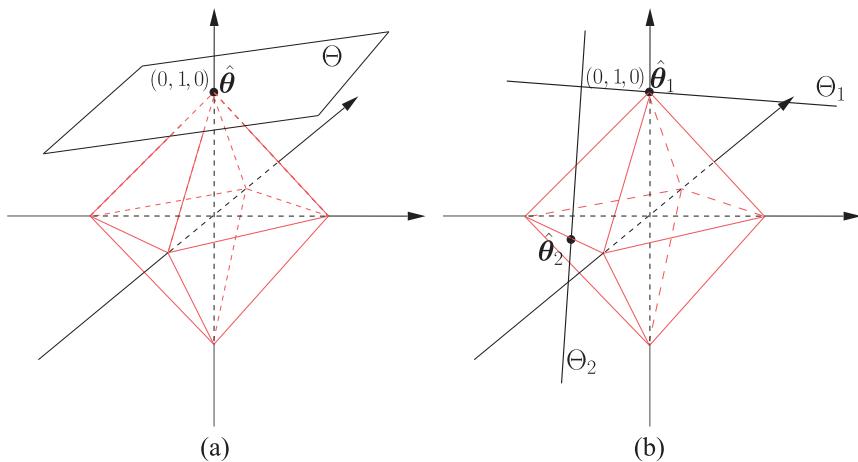
Geometric interpretation

Assume that our target solution resides in the three-dimensional space and that we are given one measurement

$$y_1 = \mathbf{x}_1^T \theta = x_{11}\theta_1 + x_{12}\theta_2 + x_{13}\theta_3.$$

Then the solution lies in the two-dimensional (hyper)plane, which is described by the previous equation. To get the minimal ℓ_1 solution we keep increasing the size of the ℓ_1 ball³ (the set of all points that have

³ Observe that in the three-dimensional space the ℓ_1 ball looks like a diamond.

**FIGURE 9.8**

(a) The ℓ_1 ball intersecting with a plane. The only possible scenario, for the existence of a unique common intersecting point of the ℓ_1 ball with a plane in the Euclidean \mathbb{R}^3 space, is for the point to be located at one of the vertices of the ℓ_1 ball, that is, to be a 1-sparse vector. (b) The ℓ_1 ball intersecting with lines. In this case, the sparsity level of the unique intersecting point is relaxed; it could be a 1- or a 2-sparse vector.

equal ℓ_1 norm) until it touches this plane. The only way that these two geometric objects have a single point in common (unique solution) is when they meet at a vertex of the diamond. This is shown in Figure 9.8a. In other words, the resulting solution is 1-sparse, having two of its components equal to zero. This complies with the finding stated in Remark 9.3, because now $N = 1$. For any other orientation of the plane, this will either cut across the ℓ_1 ball or will share with the diamond an edge or a side. In both cases, there will be infinite solutions.

Let us now assume that we are given an extra measurement,

$$y_2 = x_{21}\theta_1 + x_{22}\theta_2 + x_{23}\theta_3.$$

The solution now lies in the intersection of the two previous planes, which is a straight line. However, now, we have more alternatives for a unique solution. A line, for example, Θ_1 , can either touch the ℓ_1 ball at a vertex (1-sparse solution) or, as shown in Figure 9.8b, it can touch the ℓ_1 ball at one of its edges, for example, Θ_2 . The latter case corresponds to a solution that lies on a two-dimensional subspace, hence it will be a 2-sparse vector. This also complies with the findings stated in Remark 9.3, because in this case we have $N = 2$, $l = 3$, and the sparsity level for a unique solution can be either 1 or 2.

Note that uniqueness is associated with the particular geometry and orientation of the affine set, which is the set of all possible solutions of the underdetermined system of equations. For the case of the squared ℓ_2 norm, the solution is always unique. This is a consequence of the (hyper)spherical shape formed by the Euclidean norm. From a mathematical point of view, the squared ℓ_2 norm is a strictly convex function. This is not the case for the ℓ_1 norm, which is convex, albeit not a strictly convex function (Problem 9.13).

Example 9.2. Consider a sparse vector parameter $[0, 1]^T$, which we assume to be unknown. We will use one measurement to *sense* it. Based on this single measurement, we will use the ℓ_1 minimizer of (9.21) to recover its true value. Let us see what happens.

We will consider three different values of the “sensing” (input) vector \mathbf{x} in order to obtain the measurement $y = \mathbf{x}^T \boldsymbol{\theta}$: a) $\mathbf{x} = [\frac{1}{2}, 1]^T$, b) $\mathbf{x} = [1, 1]^T$, and c) $\mathbf{x} = [2, 1]^T$. The resulting measurement, after sensing $\boldsymbol{\theta}$ by \mathbf{x} , is $y = 1$ for all three previous cases.

Case a: The solution will lie on the straight line

$$\Theta = \left\{ [\theta_1, \theta_2]^T \in \mathbb{R}^2 : \frac{1}{2}\theta_1 + \theta_2 = 1 \right\},$$

which is shown in Figure 9.9a. For this setting, expanding the ℓ_1 ball, this will touch the straight line (our solution’s affine set) at the vertex $[0, 1]^T$. This is a unique solution, hence it is sparse, and it coincides with the true value.

Case b: The solution lies on the straight line

$$\Theta = \left\{ [\theta_1, \theta_2]^T \in \mathbb{R}^2 : \theta_1 + \theta_2 = 1 \right\},$$

which is shown in Figure 9.9b. For this setup, there is an infinite number of solutions, including two sparse ones.

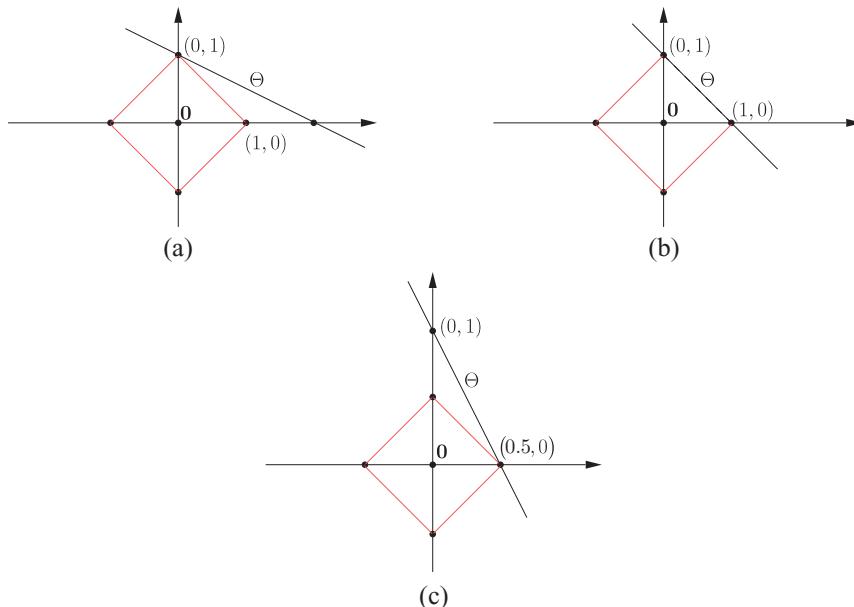


FIGURE 9.9

(a) Sensing with $\mathbf{x} = [\frac{1}{2}, 1]^T$, (b) sensing with $\mathbf{x} = [1, 1]^T$, (c) sensing with $\mathbf{x} = [2, 1]^T$. The choice of the sensing vector \mathbf{x} is crucial to unveiling the true sparse solution $(0,1)$. Only the sensing vector $\mathbf{x} = [\frac{1}{2}, 1]^T$ identifies uniquely the desired $(0,1)$.

Case c: The affine set of solutions is described by

$$\Theta = \left\{ [\theta_1, \theta_2]^T \in \mathbb{R}^2 : 2\theta_1 + \theta_2 = 1 \right\},$$

which is sketched in [Figure 9.9c](#). The solution in this case is sparse, but it is not the correct one.

This example is quite informative. *If we sense (measure) our unknown parameter vector with appropriate sensing (input) data, the use of the ℓ_1 norm can unveil the true value of the parameter vector, even if the system of equations is underdetermined, provided that the true parameter is sparse.* This becomes our new goal; to investigate whether what we have just said can be generalized, and under which conditions it holds true. In such a case, the choice of the regressors (which we called sensing vectors) and hence the input matrix (which we will refer to more and more frequently as the sensing matrix) acquire extra significance. It is not enough for the designer to care only for the rank of the matrix, that is, the linear independence of the sensing vectors. One has to make sure that the corresponding affine set of the solutions has such an orientation so that the touch with the ℓ_1 ball (as this increases from zero to meet this plane) is a “gentle” one; that is, they meet at a single point, and more important at the correct one, which is the point that represents the true value of the sparse parameter vector.

Remarks 9.4.

- Often in practice, the columns of the input matrix, X , are normalized to unit ℓ_2 norm. Although ℓ_0 norm is insensitive to the values of the nonzero components of θ , this is not the case with the ℓ_1 and ℓ_2 norms. Hence, while trying to minimize the respective norms and at the same time fulfill the constraints, components that correspond to columns of X with high energy (norm) are favored over the rest. Hence, the latter become more popular candidates to be pushed to zero. In order to avoid such situations, the columns of X are normalized to unity by dividing each element of the column vector by the respective (Euclidean) norm.

9.6 UNIQUENESS OF THE ℓ_0 MINIMIZER

Our first goal is to derive *sufficient* conditions that guarantee uniqueness of the ℓ_0 minimizer, which has been defined in [Section 9.5](#).

Definition 9.2. The *spark* of a full row rank $N \times l$ ($l \geq N$) matrix, X , denoted as $\text{spark}(X)$, is the *smallest* number of its linearly dependent columns.

According to the previous definition, *any* $m < \text{spark}(X)$ columns of X is, necessarily, *linearly independent*. The spark of a square, $N \times N$, full rank matrix is equal to $N + 1$.

Remarks 9.5.

- In contrast to the rank of a matrix, which can be easily determined, its spark can only be obtained by resorting to a combinatorial search over all possible combinations of the columns of the respective matrix (see, e.g., [[15](#), [37](#)]). The notion of the spark was used in the context of sparse representation, under the name *Uniqueness Representation Property*, in [[53](#)]. The name “spark” was coined in [[37](#)]. An interesting discussion relating this matrix index with indices used in other disciplines, is given in [[15](#)].
- Note that the notion of “spark” is related to the notion of the minimum Hamming weight of a linear code in coding theory (e.g., [[60](#)]).

Example 9.3. Consider the following matrix

$$X = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

The matrix has rank equal to 4 and spark equal to 3. Indeed, any pair of columns is linearly independent. On the other hand, the first, the second, and the fifth columns are linearly dependent. The same is also true for the combination of the second, third, and sixth columns. Also, the maximum number of linearly independent columns is four.

Lemma 9.2. *If $\text{null}(X)$ is the null space of X , then*

$$\|\theta\|_0 \geq \text{spark}(X), \quad \forall \theta \in \text{null}(X), \theta \neq \mathbf{0}.$$

Proof: To derive a contradiction, assume that there exists a $\theta \in \text{null}(X)$, $\theta \neq \mathbf{0}$, such that $\|\theta\|_0 < \text{spark}(X)$. Because by definition $X\theta = \mathbf{0}$, there exists a number of $\|\theta\|_0$ columns of X that are linearly dependent. However, this contradicts the minimality of $\text{spark}(X)$, and the claim of Lemma 9.2 is established.

Lemma 9.3. *If a linear system of equations, $X\theta = \mathbf{y}$, has a solution that satisfies*

$$\|\theta\|_0 < \frac{1}{2} \text{spark}(X),$$

then this is the sparsest possible solution. In other words, this is, necessarily, the unique solution of the ℓ_0 minimizer.

Proof: Consider any other solution $\mathbf{h} \neq \theta$. Then, $\theta - \mathbf{h} \in \text{null}(X)$, that is,

$$X(\theta - \mathbf{h}) = \mathbf{0}.$$

Thus, according to Lemma 9.2,

$$\text{spark}(X) \leq \|\theta - \mathbf{h}\|_0 \leq \|\theta\|_0 + \|\mathbf{h}\|_0. \quad (9.24)$$

Observe that although the ℓ_0 “norm” is not a true norm, it can be readily verified by simple inspection and reasoning that the triangular property is satisfied. Indeed, by adding two vectors together, the resulting number of nonzero elements will always be at most equal to the total number of nonzero elements of the two vectors. Therefore, if $\|\theta\|_0 < \frac{1}{2} \text{spark}(X)$, then (9.24) suggests that

$$\|\mathbf{h}\|_0 > \frac{1}{2} \text{spark}(X) > \|\theta\|_0.$$

Remarks 9.6.

- Lemma 9.3 is a very interesting result. We have a sufficient condition to check whether a solution is the unique optimal in a generally NP-hard problem. Of course, although this is nice from a theoretical point of view, it is not of much use by itself, because the related bound (the spark) can only be obtained after a combinatorial search. In the next section, we will see that we can relax the bound by involving another index in place of the spark, which can be easily computed.
- An obvious consequence of the previous lemma is that if the unknown parameter vector is a sparse one with k nonzero elements, then if matrix X is chosen in order to have $\text{spark}(X) > 2k$, the true

parameter vector is necessarily the sparsest one that satisfies the set of equations, and the (unique) solution to the ℓ_0 minimizer.

- In practice, the goal is to sense the unknown parameter vector by a matrix that has a spark as high as possible, so that the previously stated sufficiency condition covers a wide range of cases. For example, if the spark of the input matrix is equal to three, then one can check for optimal sparse solutions up to a sparsity level of $k = 1$. From the respective definition, it is easily seen that the values of the spark are in the range $1 < \text{spark}(X) \leq N + 1$.
- Constructing an $N \times l$ matrix X in a random manner, by generating i.i.d. entries, guarantees with high probability that $\text{spark}(X) = N + 1$; that is, any N columns of the matrix are linearly independent.

9.6.1 MUTUAL COHERENCE

Because the spark of a matrix is a number that is difficult to compute, our interest shifts to another index, which can be derived more easily and at the same time offers a useful bound on the spark. The *mutual coherence* of an $N \times l$ matrix X [65], denoted as $\mu(X)$, is defined as

$$\boxed{\mu(X) := \max_{1 \leq i < j \leq l} \frac{|\mathbf{x}_i^T \mathbf{x}_j|}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|} : \quad \text{Mutual Coherence,}} \quad (9.25)$$

where \mathbf{x}_i , $i = 1, 2, \dots, l$, denote the columns of X (notice the difference in notation between a row \mathbf{x}_i^T and a column⁴ \mathbf{x}_i of the matrix X). This number reminds us of the correlation coefficient between two random variables. Mutual coherence is bounded as $0 \leq \mu(X) \leq 1$. For a square orthogonal matrix, X , $\mu(X) = 0$. For general matrices, with $l > N$, $\mu(X)$ satisfies

$$\sqrt{\frac{l - N}{N(l - 1)}} \leq \mu(X) \leq 1,$$

which is known as the *Welch bound* [97] (Problem 9.15). For large values of l , the lower bound becomes, approximately, $\mu(X) \geq \frac{1}{\sqrt{N}}$. Common sense reasoning guides us to construct input (sensing) matrices of mutual coherence as small as possible. Indeed, the purpose of the sensing matrix is to “measure” the components of the unknown vector and “store” this information in the measurement vector \mathbf{y} . Thus, this should be done in such a way that \mathbf{y} retains as much information about the components of $\boldsymbol{\theta}$ as possible. This can be achieved if the columns of the sensing matrix, X , are as “independent” as possible. Indeed, \mathbf{y} is the result of a combination of the columns of X , each one weighted by a different component of $\boldsymbol{\theta}$. Thus, if the columns are as “independent” as possible, then the information regarding each component of $\boldsymbol{\theta}$ is contributed by a different direction, making its recovery easier. This is easier understood if X is a square orthogonal matrix. In the more general case of a nonsquare matrix, the columns should be made as “orthogonal” as possible.

Example 9.4. Assume that X is an $N \times 2N$ matrix, formed by concatenating two orthonormal bases together,

$$X = [I, W],$$

⁴ Not to be confused with the roman font used for random variables in previous chapters.

where I is the identity matrix, having as columns the vectors e_i , $i = 1, 2, \dots, N$, with elements equal to

$$\delta_{ir} = \begin{cases} 1, & \text{if } i = r, \\ 0, & \text{if } i \neq r, \end{cases}$$

for $r = 1, 2, \dots, N$. The matrix W is the orthonormal DFT matrix, defined as

$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & \dots & 1 \\ 1 & W_N & \dots & W_N^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{N-1} & \dots & W_N^{(N-1)(N-1)} \end{bmatrix},$$

where

$$W_N := \exp\left(-j\frac{2\pi}{N}\right).$$

Such an overcomplete dictionary could be used to represent signal vectors in terms of the expansion in (9.16), which comprise the sum of sinusoids with very narrow, spiky-like pulses. The inner products between any two columns of I and between any two columns of W are zero, due to orthogonality. On the other hand, it is easy to see that the inner product between any column of I and any column of W has absolute value equal to $\frac{1}{\sqrt{N}}$. Hence, the mutual coherence of this matrix is $\mu(X) = \frac{1}{\sqrt{N}}$. Moreover, observe that the spark of this matrix is $\text{spark}(X) = N + 1$.

Lemma 9.4. *For any $N \times l$ matrix X , the following inequality holds:*

$$\text{spark}(X) \geq 1 + \frac{1}{\mu(X)}. \quad (9.26)$$

The proof is given in [37] and it is based on arguments that stem from matrix theory applied on the Gram matrix, $X^T X$, of X (Problem 9.16). A “superficial” look at the previous bound is that for very small values of $\mu(X)$ the spark can be larger than $N + 1$! Looking at the proof, it is seen that in such cases the spark of the matrix attains its maximum value $N + 1$.

The result complies with common sense reasoning. The smaller the value of $\mu(X)$, the more independent the columns of X , hence the higher the value its spark is expected to be. Based on this lemma, we can now state the following theorem, first given in [37]. Combining Lemma 9.3 and (9.26), we come to the following important theorem.

Theorem 9.1. *If the linear system of equations in (9.17) has a solution that satisfies the condition*

$$\|\theta\|_0 < \frac{1}{2} \left(1 + \frac{1}{\mu(X)} \right), \quad (9.27)$$

then this solution is the sparsest one.

Remarks 9.7.

- The bound in (9.27) is “psychologically” important. It relates an easily computed bound to check whether the solution to an NP-hard task is the optimal one. However, it is not a particularly good bound and it restricts the range of values in which it can be applied. As we saw in Example 9.4, while the maximum possible value of the spark of a matrix was equal to $N + 1$, the minimum

possible value of the mutual coherence was $\frac{1}{\sqrt{N}}$. Therefore, the bound based on the mutual coherence restricts the range of sparsity, that is, $\|\theta\|_0$, where one can check optimality, to around $\frac{1}{2}\sqrt{N}$. Moreover, as the previously stated Welch bound suggests, this $\mathcal{O}(\frac{1}{\sqrt{N}})$ dependence of the mutual coherence seems to be a more general trend and not only the case for [Example 9.4](#) (see, e.g., [36]). On the other hand, as we have already stated in the [Remarks 9.6](#), one can construct random matrices with spark equal to $N + 1$; hence, using the bound based on the spark, one could expand the range of sparse vectors up to $\frac{1}{2}N$.

9.7 EQUIVALENCE OF ℓ_0 AND ℓ_1 MINIMIZERS: SUFFICIENCY CONDITIONS

We have now come to the crucial point where we will establish the conditions that guarantee the equivalence between the ℓ_1 and the ℓ_0 minimizers. Hence, under such conditions, a problem that is in general an NP-hard one *can be solved via a tractable convex optimization task*. Under these conditions, the zero value encouraging nature of the ℓ_1 norm, which has already been discussed, obtains a much higher stature; it provides the sparsest solution.

9.7.1 CONDITION IMPLIED BY THE MUTUAL COHERENCE NUMBER

Theorem 9.2. *Let the underdetermined system of equations*

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta},$$

where \mathbf{X} is an $N \times l$ ($N < l$) full row rank matrix. If a solution exists and satisfies the condition

$$\|\boldsymbol{\theta}\|_0 < \frac{1}{2} \left(1 + \frac{1}{\mu(\mathbf{X})} \right), \quad (9.28)$$

then this is the unique solution of both, the ℓ_0 as well as the ℓ_1 minimizers.

This is a very important theorem, and it was shown independently in [37, 54]. Earlier versions of the theorem addressed the special case of a dictionary comprising two orthonormal bases, [36, 48]. A proof is also summarized in [15] ([Problem 9.17](#)). This theorem established, for the first time, what it was until then empirically known: often, the ℓ_1 and ℓ_0 minimizers result in the same solution.

Remarks 9.8.

- The theory that we have presented so far is very satisfying, because it offers the theoretical framework and conditions that guarantee uniqueness of a sparse solution to an underdetermined system of equations. Now we know that under certain conditions, the solution, which we obtain by solving the convex ℓ_1 minimization task, is the (unique) sparsest one. However, from a practical point of view, the theory, which is based on mutual coherence, does not tell the whole story and falls short in predicting what happens in practice. Experimental evidence suggests that the range of sparsity levels, for which the ℓ_0 and ℓ_1 tasks give the same solution, is much wider than the range guaranteed by the mutual coherence bound. Hence, there is a lot of theoretical happening in order to improve this bound. A detailed discussion is beyond the scope of this book. In the next section, we will present one of these bounds, because it is the one that currently dominates the scene. For more details and a related discussion, the interested reader may consult, for example, [39, 49, 50].

9.7.2 THE RESTRICTED ISOMETRY PROPERTY (RIP)

Definition 9.3. For each integer $k = 1, 2, \dots$, define the *isometry constant* δ_k of an $N \times l$ matrix X as the *smallest* number such that

$$(1 - \delta_k) \|\theta\|_2^2 \leq \|X\theta\|_2^2 \leq (1 + \delta_k) \|\theta\|_2^2 : \quad \text{The RIP Condition,} \quad (9.29)$$

holds true for *all* k -sparse vectors θ .

This definition was introduced in [19]. We loosely say that matrix X obeys the RIP of order k if δ_k is not too close to one. When this property holds true, it implies that the Euclidean norm of θ is approximately *preserved*, after projecting it onto the rows of X . Obviously, if matrix X was orthonormal then $\delta_k = 0$. Of course, because we are dealing with nonsquare matrices this is not possible. However, the closer δ_k is to zero, the closer to orthonormal *all* subsets of k columns of X are. Another viewpoint of (9.29) is that X preserves Euclidean distances between k -sparse vectors. Let us consider two k -sparse vectors, θ_1, θ_2 and apply (9.29) to their difference $\theta_1 - \theta_2$, which, in general, is a $2k$ -sparse vector. Then we obtain

$$(1 - \delta_{2k}) \|\theta_1 - \theta_2\|_2^2 \leq \|X(\theta_1 - \theta_2)\|_2^2 \leq (1 + \delta_{2k}) \|\theta_1 - \theta_2\|_2^2. \quad (9.30)$$

Thus, when δ_{2k} is small enough, the Euclidean distance is preserved after projection in the lower dimensional observations' space. In words, if the RIP holds true, this means that searching for a sparse vector in the lower dimensional subspace, \mathbb{R}^N , formed by the observations, and not in the original l -dimensional space, one can still recover the vector since distances are preserved and the target vector is not "confused" with others. After projection onto the rows of X , the discriminatory power of the method is retained. It is interesting to point out that the RIP is also related to the condition number of the Grammian matrix. In [6, 19], it is pointed out that if X_r denotes the matrix that results by considering only r of the columns of X , then the RIP in (9.29) is equivalent with requiring the respective Grammian, $X_r^T X_r$, $r \leq k$, to have its eigenvalues within the interval $[1 - \delta_k, 1 + \delta_k]$. Hence, the more well conditioned the matrix, the better we dig out the information hidden in the lower dimensional space.

Theorem 9.3. Assume that for some k , $\delta_{2k} < \sqrt{2} - 1$. Then the solution to the ℓ_1 minimizer of (9.21), denoted as θ_* , satisfies the following two conditions:

$$\|\theta - \theta_*\|_1 \leq C_0 \|\theta - \theta_k\|_1, \quad (9.31)$$

and

$$\|\theta - \theta_*\|_2 \leq C_0 k^{-\frac{1}{2}} \|\theta - \theta_k\|_1, \quad (9.32)$$

for some constant C_0 . In the previously stated formulas, θ is the true (target) vector that generates the observations in (9.21) and θ_k is the vector that results from θ if we keep its k largest components and set the rest equal to zero [18, 19, 22, 23].

Hence, if the true vector is a sparse one, that is, $\theta = \theta_k$, then the ℓ_1 minimizer recovers the (unique) exact value. On the other hand, if the true vector is not a sparse one, then the minimizer results in a solution whose accuracy is dictated by a genie-aided procedure that knew in advance the locations of the k largest components of θ . This is a groundbreaking result. Moreover, it is deterministic; it is always true and not with high probability. Note that the isometry property of order $2k$ is used, because at the heart of the method lies our desire to preserve the norm of the differences between vectors.

Let us now focus on the case where there is a k -sparse vector that generates the observations, that is, $\theta = \theta_k$. Then it is shown in [18] that the condition $\delta_{2k} < 1$ guarantees that the ℓ_0 minimizer has a unique

k -sparse solution. In other words, in order to get the equivalence between the ℓ_1 and ℓ_0 minimizers, the range of values for δ_{2k} has to be decreased to $\delta_{2k} < \sqrt{2} - 1$, according to [Theorem 9.3](#). This sounds reasonable. If we relax the criterion and use ℓ_1 instead of ℓ_0 , then the sensing matrix has to be more carefully constructed. Although I will not provide the proofs of these theorems here, because their formulation is well beyond the scope of this book, it is interesting to follow what happens if $\delta_{2k} = 1$. This will give us a flavor of the essence behind the proofs. If $\delta_{2k} = 1$, the left-hand side term in [\(9.30\)](#) becomes zero. In this case, there may exist two k -sparse vectors θ_1, θ_2 such that $X(\theta_1 - \theta_2) = 0$, or $X\theta_1 = X\theta_2$. Thus, it is not possible to recover all k -sparse vectors, after projecting them in the observations space, by any method.

The previous argument also establishes a connection between RIP and the spark of a matrix. Indeed, if $\delta_{2k} < 1$, this guarantees that any number of columns of X up to $2k$ are linearly independent, because for any $2k$ -sparse θ , [\(9.29\)](#) guarantees that $\|X\theta\|_2 > 0$. This implies that $\text{spark}(X) > 2k$. A connection between RIP and the coherence is established in [\[16\]](#), where it is shown that if X has coherence $\mu(X)$, and unit norm columns, then X satisfies the RIP of order k with $\delta_k \leq (k-1)\mu(X)$.

Constructing matrices that obey the RIP of order k

It is apparent from our previous discussion that the higher the value of k , for which the RIP property of a matrix, X , holds true, the better, since a larger range of sparsity levels can be handled. Hence, a main goal toward this direction is to construct such matrices. It turns out that verifying the RIP for a matrix of a general structure is a difficult task. This reminds us of the spark of the matrix, which is also a difficult task to compute. However, it turns out that for a certain class of random matrices, the RIP can be established in an affordable way. Thus, constructing such sensing matrices has dominated the scene of related research. We will present a few examples of such matrices, which are also very popular in practice, without going into details of the proofs, because this is beyond the scope of this book. The interested reader may find this information in the related references.

Perhaps the most well-known example of a random matrix is the Gaussian one, where the entries $X(i,j)$ of the sensing matrix are i.i.d. realizations from a Gaussian pdf $\mathcal{N}(0, \frac{1}{N})$. Another popular example of such matrices is constructed by sampling i.i.d. entries from Bernoulli, or related, distributions

$$X(i,j) = \begin{cases} \frac{1}{\sqrt{N}}, & \text{with probability } \frac{1}{2}, \\ -\frac{1}{\sqrt{N}}, & \text{with probability } \frac{1}{2}, \end{cases}$$

or

$$X(i,j) = \begin{cases} +\sqrt{\frac{3}{N}}, & \text{with probability } \frac{1}{6}, \\ 0, & \text{with probability } \frac{2}{3}, \\ -\sqrt{\frac{3}{N}}, & \text{with probability } \frac{1}{6}. \end{cases}$$

Finally, one can adopt the uniform distribution and construct the columns of X by sampling uniformly at random on the unit sphere in \mathbb{R}^N . It turns out that such matrices obey the RIP of order k with overwhelming probability, provided that the number of observations, N , satisfies the inequality

$$N \geq Ck \ln(l/k), \quad (9.33)$$

where C is some constant, which depends on the isometry constant δ_k . In words, having such a matrix at our disposal, one can recover a k -sparse vector from $N < l$ observations, where N is larger than the sparsity level by an amount controlled by the inequality (9.33). More on these issues can be obtained from, for example, [6, 67].

Besides random matrices, one can construct other matrices that obey the RIP. One such example includes the partial Fourier matrices, which are formed by selecting uniformly at random N rows drawn from the $l \times l$ DFT matrix. Although the required number of samples for the RIP to be satisfied may be larger than the bound in (9.33) (see [79]), Fourier-based sensing matrices offer certain computational advantages when it comes to storage ($\mathcal{O}(N \ln l)$) and matrix-vector products ($\mathcal{O}(l \ln l)$), [20]. In [56], the case of random Toeplitz sensing matrices, containing statistical dependencies across rows is considered and it is shown that they can also satisfy the RIP with high probability. This is of particular importance in signal processing and communications applications, where it is very common for a system to be excited in its input via a time series, hence independence between successive input rows cannot be assumed. In [44, 76], the case of separable matrices is considered where the sensing matrix is the result of a Kronecker product of matrices, which satisfy the RIP individually. Such matrices are of interest for multidimensional signals, in order to exploit the sparsity structure along each one of the involved dimensions. For example, such signals may occur while trying to “encode” information associated with an event whose activity spreads across the temporal, spectral, spatial, and other domains.

In spite of their theoretical elegance, the derived bounds that determine the number of the required observations for certain sparsity levels fall short of the experimental evidence (e.g., [39]). In practice, a rule of thumb is to use N of the order of $3k$ - $5k$ [18]. For large values of l , compared to the sparsity level, the analysis in [38] suggests that we can recover most sparse signals when $N \approx 2k \ln(l/N)$. In an effort to overcome the shortcomings associated with the RIP, a number of other techniques have been proposed (e.g., [11, 30, 39, 84]). Furthermore, in specific applications, the use of an empirical study may be a more appropriate path.

Note that, in principle, the minimum number of observations that are required to recover a k -sparse vector from $N < l$ observations is $N \geq 2k$. Indeed, in the spirit of the discussion after [Theorem 9.3](#), the main requirement that a sensing matrix must fulfill is not to map two different k -sparse vectors to the same measurement vector y . Otherwise, one can never recover both vectors from their (common) observations. If we have $2k$ observations and a sensing matrix that guarantees that any $2k$ columns are linearly independent, then the previously stated requirement is satisfied. However, the bounds on the number of observations set in order for the respective matrices to satisfy the RIP are larger. This is because RIP accounts also for the stability of the recovery process. We will come to this issue in [Section 9.9](#), where we talk about *stable* embeddings.

9.8 ROBUST SPARSE SIGNAL RECOVERY FROM NOISY MEASUREMENTS

In the previous section, our focus was on recovering a sparse solution from an underdetermined system of equations. In the formulation of the problem, we assumed that there is no noise in the obtained observations. Having acquired some experience and insight from a simpler scenario, we now turn our attention to the more realistic task, where uncertainties come into the scene. One type of uncertainty may be due to the presence of noise, and our observations’ model comes back to the standard regression form

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta}, \quad (9.34)$$

where X is our familiar nonsquare $N \times l$ matrix. A sparsity-aware formulation for recovering θ from (9.34) can be cast as

$$\begin{aligned} \min_{\theta \in \mathbb{R}^l} \quad & \|\theta\|_1 \\ \text{s.t.} \quad & \|y - X\theta\|_2^2 \leq \epsilon, \end{aligned} \quad (9.35)$$

which coincides with the LASSO task given in (9.8). Such a formulation implicitly assumes that the noise is bounded and the respective range of values is controlled by ϵ . One can consider a number of different variants. For example, one possibility would be to minimize the $\|\cdot\|_0$ norm instead of the $\|\cdot\|_1$, albeit losing the computational elegance of the latter. An alternative route would be to replace the Euclidean norm in the constraints with another one.

Besides the presence of noise, one could see the previous formulation from a different perspective. The unknown parameter vector, θ , may not be exactly sparse, but it may consist of a few large components, while the rest are small and close to, yet not necessarily equal to, zero. Such a model misfit can be accommodated by allowing a deviation of y from $X\theta$.

In this relaxed setting of a sparse solution recovery, the notions of uniqueness and equivalence concerning the ℓ_0 and ℓ_1 solutions no longer apply. Instead, the issue that now gains importance is that of *stability* of the solution. To this end, we focus on the computationally attractive ℓ_1 task. The counterpart of [Theorem 9.3](#) is now expressed as follows.

Theorem 9.4. *Assume that the sensing matrix, X , obeys the RIP with $\delta_{2k} < \sqrt{2} - 1$, for some k . Then the solution θ_* of (9.35) satisfies the following ([22, 23]),*

$$\|\theta - \theta_*\|_2 \leq C_0 k^{-\frac{1}{2}} \|\theta - \theta_k\|_1 + C_1 \sqrt{\epsilon}, \quad (9.36)$$

for some constants C_1 , C_0 , and θ_k as defined in [Theorem 9.3](#).

This is also an elegant result. If the model is exact and $\epsilon = 0$ we obtain (9.32). If not, the higher the uncertainty (noise) term in the model, the higher our ambiguity about the solution. Note, also, that the ambiguity about the solution depends on how far the true model is from θ_k . If the true model is k -sparse, the first term on the right-hand side of the inequality is zero. The values of C_1, C_0 depend on δ_{2k} but they are small, for example, close to five or six, [23].

The important conclusion here is that *the LASSO formulation for solving inverse problems (which in general, as we noted in [Chapter 3](#), tend to be ill-conditioned) is a stable one and the noise is not amplified excessively during the recovery process*.

9.9 COMPRESSED SENSING: THE GLORY OF RANDOMNESS

The way in which this chapter was deployed followed, more or less, the sequence of developments that took place during the evolution of the sparsity-aware parameter estimation field. We intentionally made an effort to follow such a path, because this is also indicative of how science evolves in most cases. The starting point had a rather strong mathematical flavor: to develop conditions for the solution of an underdetermined linear system of equations, under the sparsity constraint and in a mathematically tractable way, that is, using convex optimization. In the end, the accumulation of a sequence of individual contributions revealed that the solution can be (uniquely) recovered if the unknown quantity is sensed via randomly chosen data samples. This development has, in turn, given birth to a new field

with strong theoretical interest as well as with an enormous impact on practical applications. This new emerged area is known as *compressed sensing* or *compressive sampling* (CS). Although CS builds around the LASSO and basis pursuit (and variants of them, as we will soon see), it has changed our view on how to sense and process signals efficiently.

Compressed sensing

In compressed sensing, the goal is to directly acquire as few samples as possible that encode the minimum information needed to obtain a compressed signal representation. In order to demonstrate this, let us return to the data compression example discussed in [Section 9.4](#). There, it was commented that the “classical” approach to compression was rather unorthodox, in the sense that first all (i.e., a number of l) samples of the signal are used, and then they are processed to obtain l transformed values, from which only a small subset is used for coding. In the CS setting, the procedure changes to the following one.

Let X be an $N \times l$ sensing matrix, which is applied to the (unknown) signal vector, s , in order to obtain the observations, y , and Ψ be the dictionary matrix that describes the domain where the signal s accepts a sparse representation, that is,

$$\begin{aligned} s &= \Psi\theta, \\ y &= Xs. \end{aligned} \tag{9.37}$$

Assuming that at most k of the components of θ are nonzero, this can be obtained by the following optimization task

$$\begin{aligned} \min_{\theta \in \mathbb{R}^l} \quad & \|\theta\|_1 \\ \text{s.t.} \quad & y = X\Psi\theta, \end{aligned} \tag{9.38}$$

provided that the combined matrix $X\Psi$ complies with the RIP, and the number of observations, N , is large enough, as dictated by the bound in [\(9.33\)](#). Note that s needs not be stored and can be obtained any time, once θ is known. Moreover, as we will soon discuss, there are techniques that allow observations, y_n , $n = 1, 2, \dots, N$, to be acquired directly from an analog signal $s(t)$, prior to obtaining its sample (vector) version, s ! Thus, from such a perspective, CS fuses the data acquisition and the compression steps together.

There are different ways to obtain a sensing matrix, X , that leads to a product $X\Psi$, which satisfies the RIP. It can be shown ([Problem 9.19](#)) that if Ψ is orthonormal and X is a random matrix, which is constructed as discussed at the end of [Section 9.7.2](#), then the product $X\Psi$ obeys the RIP, provided that [\(9.33\)](#) is satisfied. An alternative way to obtain a combined matrix that respects the RIP is to consider another orthonormal matrix Φ , whose columns have low coherence with the columns of Ψ (coherence between two matrices is defined in [\(9.25\)](#), where now, the place of \mathbf{x}_i is taken by a column of Φ and that of \mathbf{x}_j by a column of Ψ). For example, Φ could be the DFT matrix and $\Psi = I$, or vice versa. Then choose N rows of Φ uniformly at random to form X in [\(9.37\)](#). In other words, for such a case, the sensing matrix can be written as $R\Phi$, where R is an $N \times l$ matrix that extracts N rows uniformly at random. The notion of incoherence (low coherence) between the sensing and the basis matrices is closely related to RIP. The more incoherent the two matrices, the less the number of the required observations for the RIP to hold (e.g., [\[21, 79\]](#)). Another way to view incoherence is that the rows of Φ cannot be sparsely represented in terms of the columns of Ψ . It turns out that if the sensing matrix X is a random one,

formed as has already been described in [Section 9.7.2](#), then the RIP and the incoherence with any Ψ are satisfied with high probability.

It gets even better when we say that all the previously stated philosophy can be extended to the more general type of signals, which are not necessarily sparse or sparsely represented in terms of the atoms of a dictionary, and they are known as *compressible*. A signal vector is said to be compressible if its expansion in terms of a basis consists of just a few large coefficients θ_i and the rest are small. In other words, the signal vector is *approximately* sparse in some basis. Obviously, this is the most interesting case in practice, where exact sparsity is scarcely (if ever) met. Reformulating the arguments used in [Section 9.8](#), the CS task for this case can be cast as

$$\begin{aligned} \min_{\theta \in \mathbb{R}^l} \quad & \|\theta\|_1 \\ \text{s.t.} \quad & \|y - X\Psi\theta\|_2^2 \leq \epsilon, \end{aligned} \quad (9.39)$$

and everything that has been said in [Section 9.8](#) is also valid for this case, if in place of X we consider the product $X\Psi$.

Remarks 9.9.

- An important property in compressed sensing is that the sensing matrix, which provides the observations, may be chosen independently on the matrix Ψ , that is, the basis/dictionary in which the signal is sparsely represented. In other words, the sensing matrix can be “universal” and can be used to provide the observations for reconstructing any sparse or sparsely represented signal in any dictionary, provided RIP is not violated.
- Each measurement, y_n , is the result of an inner product of the signal vector with a row, x_n^T , of the sensing matrix, X . Assuming that the signal vector, s , is the result of a sampling process on an analog signal, $s(t)$, then y_n can be directly obtained, to a good approximation, by taking the inner product (integral) of $s(t)$ with a sensing waveform, $x_n(t)$, that corresponds to x_n . For example, if X is formed by ± 1 , as described in [Section 9.7.2](#), then the configuration shown in [Figure 9.10](#) results to y_n . An important aspect of this approach, besides avoiding computing and storing the l components of s , is that multiplying by ± 1 is a relatively easy operation. It is equivalent with changing the polarity of the signal and it can be implemented by employing inverters and mixers. It is a process that can be performed, in practice, at much higher rates than sampling. The

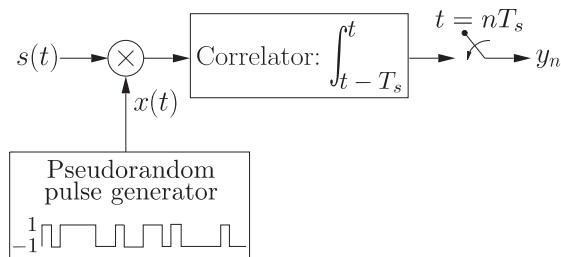


FIGURE 9.10

Sampling an analog signal $s(t)$ in order to generate the measurement y_n at the time instant n . The sampling period T_s is much lower than that required by the Nyquist sampling.

sampling system shown in Figure 9.10 is referred to as *random demodulator*, [58, 90]. It is one among the popular analog-to-digital (A/D) conversion architectures, which exploit the CS rationale in order to sample at rates much lower than those required for classical sampling. We will come back to this soon.

One of the very first CS-based acquisition systems was an imaging system called the *one pixel camera* [83], which followed an approach resembling the conventional digital CS. According to this, light of an image of interest is projected onto a random base generated by a micromirror device. A sequence of projected images is collected by a *single photodiode* and used for the reconstruction of the full image using conventional CS techniques. This was among the most catalytic examples that spread the rumor about the practical power of CS. CS is an example of common wisdom: “There is nothing more practical than a good theory!”

9.9.1 DIMENSIONALITY REDUCTION AND STABLE EMBEDDINGS

We will now shed light on what we have said so far in this chapter from a different point of view. In both cases, either when the unknown quantity was a k -sparse vector in a high-dimensional space, \mathbb{R}^l , or when the signal s was (approximately) sparsely represented in some dictionary ($s = \Psi\theta$), we chose to work in a lower dimensional space (\mathbb{R}^N), that is, the space of the observations, y . This is a typical task of dimensionality reduction, see, Chapter 19. The main task in any (linear) dimensionality reduction technique is to choose the proper matrix X , that dictates the projection to the lower dimensional space. In general, there is always a loss of information by projecting from \mathbb{R}^l to \mathbb{R}^N , with $N < l$, in the sense that we cannot recover any vector, $\theta_l \in \mathbb{R}^l$, from its projection $\theta_N \in \mathbb{R}^N$. Indeed, take any vector $\theta_{l-N} \in \text{null}(X)$, that lies in the $(l - N)$ -dimensional null space of the (full row rank) X (see Section 9.5). Then, all vectors $\theta_l + \theta_{l-N} \in \mathbb{R}^l$ share the same projection in \mathbb{R}^N . However, what we have discovered in this chapter is that if the original vector is sparse, then we can recover it exactly. This is because all the k -sparse vectors do not lie anywhere in \mathbb{R}^l , but rather in a subset of it, that is, in a *union of subspaces*, each one having dimensionality k . If the signal s is sparse in some dictionary Ψ , then one has to search for it in the union of all possible k -dimensional subspaces of \mathbb{R}^l , which are spanned by k -column vectors from Ψ [8, 62]. Of course, even in this case, where sparse vectors are involved, no projection can guarantee unique recovery. The guarantee is provided if the projection in the lower dimensional space is a *stable embedding*. A stable embedding in a lower dimensional space must guarantee that if $\theta_1 \neq \theta_2$, then their projections also remain different. Yet this is not enough. A stable embedding must guarantee that distances are (approximately) preserved; that is, vectors that lie far apart in the high-dimensional space have projections that also lie far apart. Such a property guarantees robustness to noise. The sufficient conditions, which have been derived and discussed throughout this chapter, and guarantee the recovery of a sparse vector lying in \mathbb{R}^l from its projections in \mathbb{R}^N , are conditions that guarantee stable embeddings. The RIP and the associated bound on N provide a condition on X that leads to stable embeddings. We commented on this norm-preserving property of RIP in the related section. The interesting fact that came from the theory is that we can achieve such stable embeddings via random projection matrices.

Random projections for dimensionality reduction are not new and have extensively been used in pattern recognition, clustering, and data mining (see, e.g., [1, 13, 34, 82, 86]). The advent of the big data era resparked the interest in random projection-aided data analysis algorithms (e.g., [55, 81]) for two major reasons. The first is that data processing is computationally lighter in the lower dimensional

space, because it involves operations with matrices or vectors represented with fewer parameters. Moreover, the projection of the data to lower dimensional spaces can be realized via well-structured matrices in computational cost significantly lower compared to that implied by general matrix-vector multiplications [29, 42]. The reduced computational power required by these methods renders them appealing when dealing with excessively large data volumes. The second reason is that there exist randomized algorithms, which access the data matrix a (usually fixed) number of times that is much smaller than the number of accesses performed by ordinary methods [28, 55]. This is very important whenever the full amount of data does not fit in fast memory and has to be accessed in parts from slow memory devices, such as hard discs. In such cases, the computational time is often dominated by the cost of memory access.

The spirit underlying compressed sensing has been exploited in the context of pattern recognition too. In this application, one need not return to the original high-dimensional space, after the information-digging activity in the low-dimensional subspace. Since the focus in pattern recognition is to identify the class of an object/pattern, this can be performed in the observations subspace, provided that there is no class-related information loss. In [17], it is shown, using compressed sensing arguments, that if the data is approximately linearly separable in the original high-dimensional space and the data has a sparse representation, even in an unknown basis, then projecting randomly in the observations subspace retains the structure of linear separability.

Manifold learning is another area where random projections have been recently applied. A manifold is, in general, a nonlinear k -dimensional surface, embedded in a higher dimensional (ambient) space. For example, the surface of a sphere is a two-dimensional manifold in a three-dimensional space. In [7, 95], the compressed sensing rationale is extended to signal vectors that live along a k -dimensional submanifold of the space \mathbb{R}^l . It is shown that if choosing a matrix, X , to project and a sufficient number, N , of observations, then the corresponding submanifold has a stable embedding in the observations subspace, under the projection matrix, X ; that is, pairwise Euclidean and geodesic distances are approximately preserved after the projection mapping. More on these issues can be found in the given references and in, for example, [8]. We will come to the manifold learning task in [Chapter 19](#).

9.9.2 SUB-NYQUIST SAMPLING: ANALOG-TO-INFORMATION CONVERSION

In our discussion in the Remarks presented before, we touched on a very important issue—that of going from the analog domain to the discrete one. The topic of analog-to-digital (A/D) conversion has been at the forefront of research and technology since the seminal works of Shannon, Nyquist, Whittaker, and Kotelnikof were published, see, for example, [91] for a thorough related review. We all know that if the highest frequency of an analog signal, $s(t)$, is less than $F/2$, then Shannon's theorem suggests that no loss of information is achieved if the signal is sampled, at least, at the Nyquist rate of $F = 1/T$, where T is the corresponding sampling period, and the signal can be perfectly recovered by its samples

$$s(t) = \sum_n s(nT) \operatorname{sinc}(Ft - n),$$

where $\operatorname{sinc}(\cdot)$ is the sampling function

$$\operatorname{sinc}(t) = \frac{\sin(\pi t)}{\pi t}.$$

While this has been the driving force behind the development of signal acquisition devices, the increasing complexity of emerging applications demands increasingly higher sampling rates that cannot be accommodated by today's hardware technology. This is the case, for example, in wideband communications, where conversion speeds, as dictated by Shannon's bound, have become more and more difficult to obtain. Consequently, alternatives to high rate sampling are attracting strong interest, with the goal of reducing the sampling rate by exploiting the *underlying structure* of the signals at hand. For example, in many applications, the signal comprises a few frequencies or bands, see [Figure 9.11](#) for an illustration. In such cases, sampling at the Nyquist rate is inefficient. This is an old problem investigated by a number of authors, leading to techniques that allow low rate sampling whenever the locations of the nonzero bands in the frequency spectrum are known (see, e.g., [\[61, 92, 93\]](#)). CS theory has inspired research to study cases where the locations (carrier frequencies) of the bands are not known a priori. A typical application of this kind, of high practical interest, lies within the field of cognitive radio (e.g., [\[68, 87, 102\]](#)).

The process of sampling an analog signal with a rate lower than the Nyquist one is referred to as *analog-to-information* sampling or *sub-Nyquist* sampling. Two are the most popular CS-based A/D converters. The first is the *random demodulator* (RD), which was first presented in [\[58\]](#) and later improved and theoretically developed in [\[90\]](#). RD in its basic configuration is shown in [Figure 9.10](#), and it is designed for acquiring at sub-Nyquist rates sparse multitone signals, that is, signals having a sparse DFT. This implies that the signal comprises a few frequency components, but these components are constrained to correspond to integral frequencies. This limitation was pointed out in [\[90\]](#), and potential solutions have been sought according to the general framework proposed in [\[24\]](#) and/or the heuristic approach described in [\[45\]](#). Moreover, more elaborate RD designs, such as the *random-modulation pre-integrator* (RMPI) [\[101\]](#), have the potential to deal with signals that are sparse in any domain.

Another CS-based sub-Nyquist sampling strategy that has received much attention is the *modulated wideband converter* (MWC), [\[68, 69, 71\]](#). The MWC is very efficient in acquiring multiband signals such as the one depicted in [Figure 9.11](#). This concept has also been extended to accommodate

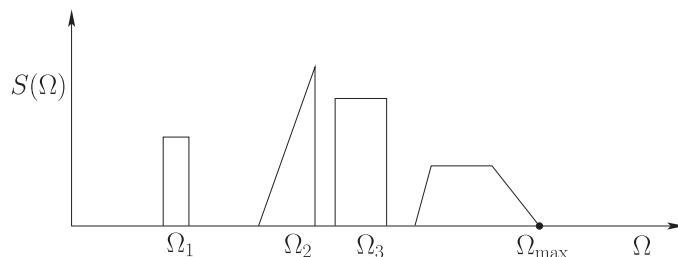


FIGURE 9.11

The Fourier transform of an analog signal, $s(t)$, which is sparse in the frequency domain; only a limited number of frequency bands contribute to its spectrum content $S(\Omega)$, where Ω stands for the angular frequency. Nyquist's theory guarantees that sampling at a frequency larger than or equal to twice the maximum Ω_{\max} is sufficient to recover the original analog signal. However, this theory does not exploit information related to the sparse structure of the signal in the frequency domain.

signals with different characteristics, such as signals consisting of short pulses [66]. An in-depth investigation, which sheds light on the similarities and differences between the RD and the MWC sampling architectures, can be found in [59].

Note that both RD and MWC sample the signal uniformly in time. In [96], a different approach is adopted, leading to much easier implementations. In particular, the preprocessing stage is avoided and nonuniformly spread in time samples are acquired directly from the raw signal. In total, less samples are obtained compared to the Nyquist sampling. Then, CS-based reconstruction is mobilized in order to recover the signal under consideration based on the values of the samples and the time information. Like in the basic RD case, the nonuniform sampling approach is suitable for signals sparse in the DFT basis. From a practical point of view, there are still a number of hardware implementation-related issues that more or less concern all the approaches above and need to be solved (see, e.g., [9, 25, 63]).

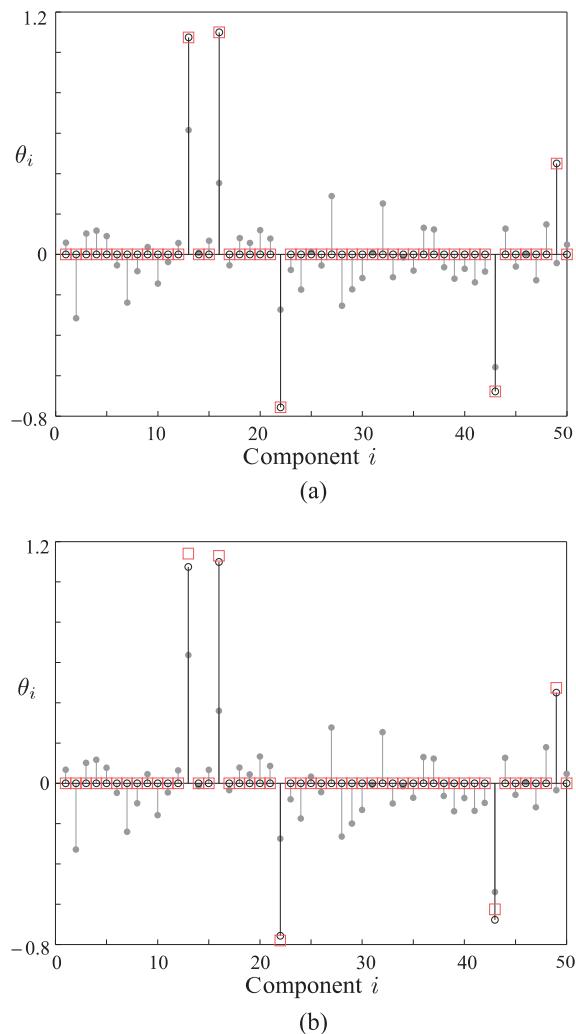
An alternative path to sub-Nyquist sampling embraces a different class of analog signals known as *multipulse* signals, that is, signals that consist of a stream of short pulses. Sparsity now refers to the time domain, and such signals may not even be bandlimited. Signals of this type can be met in a number of applications, such as in radar, ultrasound, bioimaging, and neuronal signal processing (see, e.g., [41]). An approach known as *finite rate of innovation sampling* passes an analog signal having k degrees of freedom per second through a linear time invariant filter, and then samples at a rate of $2k$ samples per second. Reconstruction is performed via rooting a high-order polynomial (see, e.g., [12, 94] and the references therein). In [66], the task of sub-Nyquist sampling is treated using CS theory arguments and an expansion in terms of Gabor functions; the signal is assumed to consist of a sum of a few pulses of finite duration, yet of unknown shape and time positions.

The task of sparsity-aware learning in the analog domain is still in its early stages, and there is a lot of ongoing activity; more on this topic can be obtained in [43, 51, 70] and the references therein.

Example 9.5. We are given a set of $N = 20$ observations stacked in the $\mathbf{y} \in \mathbb{R}^N$ vector. These were taken by applying a sensing matrix X on an “unknown” vector in \mathbb{R}^{50} , which is known to be sparse with $k = 5$ nonzero components; the location of these nonzero components in the unknown vector is not known. The sensing matrix was a random matrix with elements drawn from a normal distribution $\mathcal{N}(0, 1)$, and then the columns were normalized to unit norm. There are two scenarios for the measurements. In the first one, we are given the exact measurements, while in the second one, white Gaussian noise of variance $\sigma^2 = 0.025$ was added.

In order to recover the unknown sparse vector, the compressive sampling matching pursuit (CoSaMP, Chapter 10) algorithm was used for both scenarios.

The results are shown in Figure 9.12a and b for the noiseless and noisy scenarios, respectively. The values of the true unknown vector θ are represented with black stems topped with open circles. Note that all but five of them are zero. In Figure 9.12a, exact recovery of the unknown values is succeeded; the estimated values of $\hat{\theta}_i, i = 1, 2 \dots, 50$, are indicated with squares in red color. In the noisy case of Figure 9.12b, the resulting estimates, which are denoted with squares, deviate from the correct values. Note that estimated values very close to zero ($|\hat{\theta}| \leq 0.01$) have been omitted from the figure in order to facilitate visualization. In both figures, the stemmed gray-filled circles correspond to the minimum ℓ_2 norm LS solution. The advantages of adopting a sparsity-promoting approach to recover the solution are obvious. The CoSaMP algorithm was provided with the exact number of sparsity. The reader is advised to reproduce the example and play with different values of the parameters and see how results are affected.

**FIGURE 9.12**

(a) Noiseless case. The values of the true vector, which generated the data for [Example 9.5](#), are shown with stems topped with open circles. The recovered points are shown with squares. An exact recovery of the signal has been obtained. The stems topped with gray-filled circles correspond to the minimum Euclidean norm LS solution. (b) This figure corresponds to the noisy counterpart of that in (a). In the presence of noise, exact recovery is not possible and the higher the variance of the noise, the less accurate the results.

9.10 A CASE STUDY: IMAGE DE-NOISING

We have already discussed compressed sensing (CS) as a notable application of sparsity-aware learning. Although CS has acquired a lot of fame, a number of classical signal processing and machine learning tasks lend themselves to efficient modeling via sparsity-related arguments. Two typical examples are:

- *De-noising*: The problem in signal de-noising is that instead of the actual signal samples, $\tilde{\mathbf{y}}$, a noisy version of the corresponding observations, \mathbf{y} , are available; that is, $\mathbf{y} = \tilde{\mathbf{y}} + \boldsymbol{\eta}$, where $\boldsymbol{\eta}$ is the vector of noise samples. Under the sparse modeling framework, the unknown signal $\tilde{\mathbf{y}}$ is modeled as a sparse representation in terms of a specific known dictionary Ψ , that is, $\tilde{\mathbf{y}} = \Psi\boldsymbol{\theta}$. Moreover, the dictionary is allowed to be redundant (overcomplete). Then, the de-noising procedure is realized in two steps.

First, an estimate of the sparse representation vector, $\boldsymbol{\theta}$, is obtained via the ℓ_0 norm minimizer or via any LASSO formulation, for example,

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \|\boldsymbol{\theta}\|_1, \quad (9.40)$$

$$\text{s.t. } \|\mathbf{y} - \Psi\hat{\boldsymbol{\theta}}\|_2^2 \leq \epsilon. \quad (9.41)$$

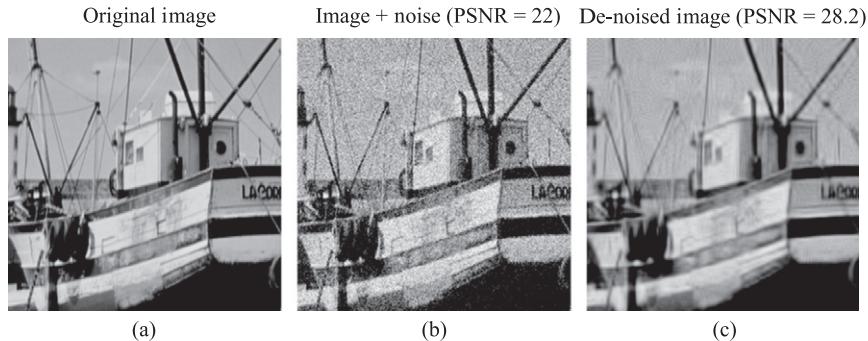
Second, the estimate of the true signal is computed as $\hat{\mathbf{y}} = \Psi\hat{\boldsymbol{\theta}}$. In [Chapter 19](#), we will study the case where the dictionary is not fixed and known, but is estimated from the data.

- *Linear inverse problems*: Such problems, which come under the more general umbrella of what is known as *signal restoration*, go one step beyond de-noising. Now, the available observations are *distorted* as well as noisy versions of the true signal samples; that is, $\mathbf{y} = H\tilde{\mathbf{y}} + \boldsymbol{\eta}$, where H is a known linear operator. For example, H may correspond to the blurring point spread function of an image, as discussed in [Chapter 4](#). Then, similar to the de-noising example, assuming that the original signal samples can be efficiently represented in terms of an overcomplete dictionary, $\hat{\boldsymbol{\theta}}$ is estimated, via any sparsity promoting method, using $H\Psi$ in place of Ψ in (9.41), and the estimate of the true signal is obtained as $\hat{\mathbf{y}} = \Psi\hat{\boldsymbol{\theta}}$.

Besides de-blurring, other applications that fall under this formulation include image inpainting, if H represents the corresponding sampling mask; inverse-Radon transform in tomography, if H comprises the set of parallel projections, and so on. See, for example, [49] for more details on this topic.

In this case study, the image de-noising task, based on the sparse and redundant formulation as discussed above, is explored. Our starting point is the 256×256 image shown in [Figure 9.13a](#). In the sequel, the image is corrupted by zero mean Gaussian noise leading to the noisy version of [Figure 9.13b](#), corresponding to peak signal-to-noise ratio (PSNR) equal to 22 dB, which is defined as

$$\text{PSNR} = 20 \log_{10} \left(\frac{m_I}{\sqrt{\text{MSE}}} \right), \quad (9.42)$$

**FIGURE 9.13**

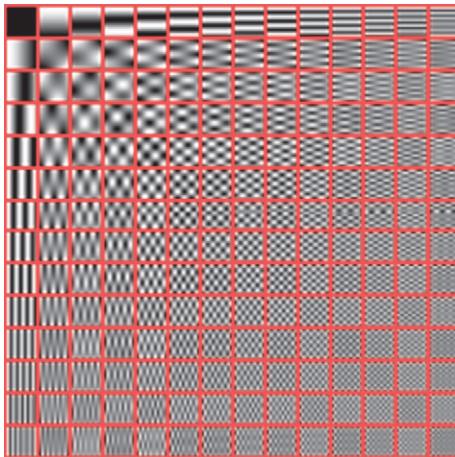
De-noising based on sparse and redundant representations.

where m_I is the maximum pixel value of the image and $\text{MSE} = \frac{1}{N_p} \|I - \tilde{I}\|_F^2$, with I and \tilde{I} being the noisy and original image matrices, N_p is equal to the total number of pixels and the Frobenius norm for matrices has been employed.

De-noising could be applied to the full image at once. However, a more efficient practice with respect to memory consumption is to split the image to patches of size much smaller than that of the image; for our case, we chose 12×12 patches. Then, de-noising is performed to each patch separately as follows: The i th patch image is reshaped in lexicographic order forming a 1-D vector, $y_i \in \mathbb{R}^{144}$. We assume that each one of the patches can be reproduced in terms of an overcomplete dictionary, as discussed before; hence, the de-noising task is equivalently formulated around (9.40)-(9.41). Denote by \tilde{y}_i the i th patch of the noise-free image. What is left is to choose a dictionary Ψ , which sparsely represents the \tilde{y}_i , and then solve for sparse θ according to (9.40)-(9.41).

It is known that images often exhibit sparse DCT transforms, so an appropriate choice for the dictionary is to fill the columns of Ψ with atoms of a redundant 2D-DCT reshaped in lexicographic order [49]. Here, 196 such atoms were used. There is a standard way to develop such a dictionary given the dimensionality of the image, and it is described in [Exercise 9.22](#). The *same* dictionary is used for *all* patches. The atoms of the dictionary, reshaped to form 12×12 blocks, are depicted in [Figure 9.14](#).

A question that naturally arises is how many patches to use. A straightforward approach is to tile the patches side by side in order to cover the whole extent of the image. This is feasible, however, it is likely to result in blocking effects at the edges of several patches. A better practice is to let the patches overlap. During the reconstruction phase ($\hat{y} = \Psi\hat{\theta}$), because each pixel is covered by more than one patch, the final value of each pixel is taken as the average of the corresponding predicted values from all the involved patches. The results of this method, for our case, are shown in [Figure 9.13c](#). The attained PSNR is higher than 28 dB.

**FIGURE 9.14**

2D-DCT Dictionary atoms, corresponding to 12×12 patch size.

PROBLEMS

9.1 If $x_i, y_i, i = 1, 2, \dots, l$, are real numbers, then prove the Cauchy-Schwarz inequality:

$$\left(\sum_{i=1}^l x_i y_i \right)^2 \leq \left(\sum_{i=1}^l x_i^2 \right) \left(\sum_{i=1}^l y_i^2 \right).$$

9.2 Prove that the ℓ_2 (Euclidean) norm is a true norm, that is, it satisfies the four conditions that define a norm.

Hint. To prove the triangle inequality, use the Cauchy-Schwarz inequality.

9.3 Prove that any function that is a norm is also a convex function.

9.4 Show Young's inequality for nonnegative real numbers a and b ,

$$ab \leq \frac{a^p}{p} + \frac{b^q}{q},$$

for $\infty > p > 1$ and $\infty > q > 1$ such that

$$\frac{1}{p} + \frac{1}{q} = 1.$$

9.5 Prove Holder's inequality for ℓ_p norms,

$$\|\mathbf{x}^T \mathbf{y}\|_1 = \sum_{i=1}^l |x_i y_i| \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q = \left(\sum_{i=1}^l |x_i|^p \right)^{1/p} \left(\sum_{i=1}^q |y_i|^q \right)^{1/q},$$

for $p \geq 1$ and $q \geq 1$ such that

$$\frac{1}{p} + \frac{1}{q} = 1.$$

Hint. Use Young's inequality.

- 9.6** Prove Minkowski's inequality,

$$\left(\sum_{i=1}^l (|x_i| + |y_i|)^p \right)^{1/p} \leq \left(\sum_{i=1}^l |x_i|^p \right)^{1/p} + \left(\sum_{i=1}^l |y_i|^p \right)^{1/p},$$

for $p \geq 1$.

Hint. Use Holder's inequality together with the identity

$$(|a| + |b|)^p = (|a| + |b|)^{p-1}|a| + (|a| + |b|)^{p-1}|b|.$$

- 9.7** Prove that for $p \geq 1$, the ℓ_p norm is a true norm.
9.8 Use a counterexample to show that any ℓ_p norm for $0 < p < 1$ is not a true norm and it violates the triangle inequality.
9.9 Show that the null space of a full row rank $N \times l$ matrix X is a subspace of dimensionality N , for $N < l$.
9.10 Show, using Lagrange multipliers, that the ℓ_2 minimizer in (9.18), accepts the closed form solution

$$\hat{\boldsymbol{\theta}} = X^T (X X^T)^{-1} \mathbf{y}.$$

- 9.11** Show that the necessary and sufficient condition for a $\boldsymbol{\theta}$ to be a minimizer of

$$\begin{aligned} &\text{minimize} && \|\boldsymbol{\theta}\|_1 \\ &\text{subject to} && X\boldsymbol{\theta} = \mathbf{y}, \end{aligned}$$

is the following

$$\left| \sum_{i:\theta_i \neq 0} \text{sign}(\theta_i) z_i \right| \leq \sum_{i:\theta_i=0} |z_i|, \quad \forall \mathbf{z} \in \text{null}(X),$$

where $\text{null}(X)$ is the null space of X . Moreover, if the minimizer is unique the previous inequality becomes a strict one.

- 9.12** Prove that if the ℓ_1 norm minimizer is unique, then the number of its components, which are identically zero, must be at least as large as the dimensionality of the null space of the corresponding input matrix.
9.13 Show that the ℓ_1 norm is a convex function (as all norms), yet it is not strictly convex. In contrast, the squared Euclidean norm is a strictly convex function.
9.14 Construct in the five-dimensional space a matrix that has (a) rank equal to five and spark equal to four, (b) rank equal to five and spark equal to three, and (c) rank and spark equal to four.
9.15 Let X be a full row rank $N \times l$ matrix, with $l > N$. Derive the Welch bound for the mutual coherence $\mu(X)$,

$$\mu(X) \geq \sqrt{\frac{l-N}{N(l-1)}}. \tag{9.43}$$

- 9.16** Let X be an $N \times l$ matrix. Then prove that its spark is bounded as

$$\text{spark}(X) \geq 1 + \frac{1}{\mu(X)},$$

where $\mu(X)$ is the mutual coherence of the matrix.

Hint. Consider the Gram matrix $X^T X$ and the following theorem, concerning positive definite matrices: An $m \times m$ matrix A is positive definite if

$$|A(i, i)| > \sum_{j=1, j \neq i}^m |A(i, j)|, \forall i = 1, 2, \dots, m,$$

see, for example, [57].

- 9.17** Show that if the underdetermined system of equations $y = X\theta$ accepts a solution such that

$$\|\theta\|_0 < \frac{1}{2} \left(1 + \frac{1}{\mu(X)} \right),$$

then the ℓ_1 minimizer is equivalent to the ℓ_0 one. Assume that the columns of X are normalized.

- 9.18** Prove that if the RIP of order k is valid for a matrix X and $\delta_k < 1$, then any $m < k$ columns of X

are necessarily linearly independent.

- 9.19** Show that if X satisfies the RIP of order k and some isometry constant δ_k so does the product $X\Psi$ if Ψ is an orthonormal matrix.

MATLAB Exercises

- 9.20** Consider an unknown 2-sparse vector θ_o , which when measured with the following sensing matrix

$$X = \begin{bmatrix} 0.5 & 2 & 1.5 \\ 2 & 2.3 & 3.5 \end{bmatrix},$$

that is, of $y = X\theta_o$, gives $y = [1.25, 3.75]^T$. Perform the following tasks in MATLAB:

(a) Based on the pseudo-inverse of X , compute $\hat{\theta}_2$, which is the ℓ_2 norm minimized solution, (9.18). Next, check that this solution $\hat{\theta}_2$ leads to zero estimation error (up to machine precision). Is $\hat{\theta}_2$ a 2-sparse vector such as the true unknown vector θ_o , and if it is not, how is it possible to lead to zero estimation error? (b) Solve the ℓ_0 minimization task described in (9.20) (exhaustive search) for all possible 1- and 2-sparse solutions and get the best one, $\hat{\theta}_o$. Does $\hat{\theta}_o$ lead to zero estimation error (up to machine precision)? (c) Compute and compare the ℓ_2 norms of $\hat{\theta}_2$ and $\hat{\theta}_o$. Which is the smaller one? Was this result expected?

- 9.21** Generate in MATLAB a sparse vector $\theta \in \mathbb{R}^l$, $l = 100$, with its first 5 components taking random values drawn from a normal distribution, $\mathcal{N}(0, 1)$ and the rest being equal to zero. Build, also, a sensing matrix X with $N = 30$ rows having samples normally distributed $\mathcal{N}(0, \frac{1}{\sqrt{N}})$, in order to get 30 observations based on the linear regression model $y = X\theta$. Then perform the following tasks: (a) Use the function “solvelasso.m”⁵, or any other LASSO implementation you prefer, in order to reconstruct θ from y and X . (b) Repeat the experiment with different realizations of X in order to compute the probability of correct reconstruction

⁵ It can be found in the SparseLab MATLAB toolbox, which is freely available from <http://sparselab.stanford.edu/>

(assume the reconstruction is exact when $\|\mathbf{y} - X\boldsymbol{\theta}\|_2 < 10^{-8}$). (c) Construct another sensing matrix X having $N = 30$ rows taken uniformly at random from the $l \times l$ DCT matrix, which can be obtained via the built-in MATLAB function “dctmtx.m”. Compute the probability of reconstruction when this DCT-based sensing matrix is used and confirm that results similar to those in question (b) are obtained. (d) Repeat the same experiment with matrices of the form

$$X(i,j) = \begin{cases} + \sqrt{\frac{\sqrt{p}}{N}}, & \text{with probability } \frac{1}{2\sqrt{p}}, \\ 0, & \text{with probability } 1 - \frac{1}{\sqrt{p}}, \\ - \sqrt{\frac{\sqrt{p}}{N}}, & \text{with probability } \frac{1}{2\sqrt{p}}, \end{cases}$$

for p equal to 1, 9, 25, 36, 64 (make sure that at each row and each column of X has at least a nonzero component). Give an explanation why the probability of reconstruction falls as p increases (observe that both the sensing matrix and the unknown vector are sparse).

- 9.22** This exercise reproduces the de-noising results of the case study in [Section 9.10](#), where the image depicting the boat can be downloaded from the book website. First, extract from the image all the possible sliding patches of size 12×12 using the im2col.m Matlab function. Confirm that $(256 - 12 + 1)^2 = 60,025$ patches in total are obtained. Next, a dictionary in which all the patches are sparsely represented needs to be designed.

Specifically, the dictionary atoms are going to be those corresponding to the 2D redundant DCT transform, which are obtained as follows [49]:

- a) Consider vectors $\mathbf{d}_i = [d_{i,1}, d_{i,2}, \dots, d_{i,12}]^T$, $i = 0, \dots, 13$, being the sampled sinusoids of the form

$$d_{i,t+1} = \cos\left(\frac{t\pi i}{14}\right), \quad t = 0, \dots, 11.$$

Then make a (12×14) matrix \tilde{D} , having as columns the vectors \mathbf{d}_i normalized to unit norm. D resembles a redundant DCT matrix.

- b) construct the $(12^2 \times 14^2)$ dictionary Ψ according to $\Psi = D \otimes D$, where \otimes denotes Kronecker product. Built in this way, the resulting atoms correspond to atoms related to the overcomplete 2D-DCT transform [49].

As a next step, de-noise each image patch separately. In particular, assuming that \mathbf{y}_i is the i th patch reshaped in column vector, use the function “solvelasso.m”⁶, or any other suitable algorithm you prefer, in order to estimate a sparse vector $\boldsymbol{\theta}_i \in \mathbb{R}^{196}$ and obtain the corresponding de-noised vector as $\hat{\mathbf{y}}_i = \Psi\boldsymbol{\theta}_i$. Finally, average the values of the overlapped patches in order to form the full de-noised image.

⁶ It can be found in the SparseLab MATLAB toolbox, which is freely available from <http://sparselab.stanford.edu/>

REFERENCES

- [1] D. Achlioptas, Database-friendly random projections, in: Proceedings of the Symposium on Principles of Database Systems (PODS), ACM Press, 2001, pp. 274-281.
- [2] A. Antoniadis, Wavelet methods in statistics: some recent developments and their applications, *Stat. Surv.* 1 (2007) 16-55.
- [3] J. Arenas-Garcia, A.R. Figueiras-Vidal, Adaptive combination of proportionate filters for sparse echo cancellation, *IEEE Trans. Audio Speech Language Process.* 17(6) (2009) 1087-1098.
- [4] S. Ariyavitsakul, N.R. Sollenberger, L.J. Greenstein, Tap-selectable decision feedback equalization, *IEEE Trans. Commun.* 45(12) (1997) 1498-1500.
- [5] W.U. Bajwa, J. Haupt, A.M. Sayeed, R. Nowak, Compressed channel sensing: a new approach to estimating sparse multipath channels, *Proc. IEEE* 98(6) (2010) 1058-1076.
- [6] R.G. Baraniuk, M. Davenport, R. DeVore, M.B. Wakin, A simple proof of the restricted isometry property for random matrices, *Construct. Approximat.* 28 (2008) 253-263.
- [7] R. Baraniuk, M. Wakin, Random projections of smooth manifolds, *Foundat. Comput. Math.* 9(1) (2009) 51-77.
- [8] R. Baraniuk, V. Cevher, M. Wakin, Low-dimensional models for dimensionality reduction and signal recovery: a geometric perspective, *Proc. IEEE* 98(6) (2010) 959-971.
- [9] S. Becker, Practical compressed sensing: modern data acquisition and signal processing, Ph.D. thesis, Caltech, 2011.
- [10] J. Benesty, T. Gansler, D.R. Morgan, M.M. Sondhi, S.L. Gay, *Advances in Network and Acoustic Echo Cancellation*, Springer-Verlag, Berlin, 2001.
- [11] P. Bickel, Y. Ritov, A. Tsybakov, Simultaneous analysis of LASSO and Dantzig selector, *Ann. Stat.* 37(4) (2009) 1705-1732.
- [12] T. Blu, P.L. Dragotti, M. Vetterli, P. Marziliano, L. Coulot, Sparse sampling of signal innovations, *IEEE Signal Process. Mag.* 25(2) (2008) 31-40.
- [13] A. Blum, Random projection, margins, kernels and feature selection, in: *Lecture Notes on Computer Science (LNCS)*, 2006, pp. 52-68.
- [14] S. Boyd, L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
- [15] A.M. Bruckstein, D.L. Donoho, M. Elad, From sparse solutions of systems of equations to sparse modeling of signals and images, *SIAM Rev.* 51(1) (2009) 34-81.
- [16] T.T. Cai, G. Xu, J. Zhang, On recovery of sparse signals via ℓ_1 minimization, *IEEE Trans. Informat. Theory* 55(7) (2009) 3388-3397.
- [17] R. Calderbank, S. Jeafarpour, R. Schapire, Compressed learning: Universal sparse dimensionality reduction and learning in the measurement domain, *Tech. Rep.*, Rice University, 2009.
- [18] E.J. Candès, J. Romberg, Practical signal recovery from random projections, in: *Proceedings of the SPIE 17th Annual Symposium on Electronic Imaging*, Bellingham, WA, 2005.
- [19] E.J. Candès, T. Tao, Decoding by linear programming, *IEEE Trans. Informat. Theory* 51(12) (2005) 4203-4215.
- [20] E. Candès, J. Romberg, T. Tao, Robust uncertainty principles: exact signal reconstruction from highly incomplete Fourier information, *IEEE Trans. Informat. Theory* 52(2) (2006) 489-509.
- [21] E. Candès, T. Tao, Near optimal signal recovery from random projections: Universal encoding strategies, *IEEE Trans. Informat. Theory* 52(12) (2006) 5406-5425.
- [22] E.J. Candès, J. Romberg, T. Tao, Stable recovery from incomplete and inaccurate measurements, *Commun. Pure Appl. Math.* 59(8) (2006) 1207-1223.
- [23] E.J. Candès, M.B. Wakin, An introduction to compressive sampling, *IEEE Signal Process. Mag.* 25(2) (2008) 21-30.
- [24] E.J. Candès, Y.C. Eldar, D. Needell, P. Randall, Compressed sensing with coherent and redundant dictionaries, *Appl. Comput. Harmonic Anal.* 31(1) (2011) 59-73.

- [25] F. Chen, A.P. Chandrakasan, V.M. Stojanovic, Design and analysis of hardware efficient compressed sensing architectures for compression in wireless sensors, *IEEE Trans. Solid State Circuits* 47(3) (2012) 744-756.
- [26] S. Chen, D.L. Donoho, M. Saunders, Atomic decomposition by basis pursuit, *SIAM J. Sci. Comput.* 20(1) (1998) 33-61.
- [27] J.F. Claerbout, F. Muir, Robust modeling with erratic data, *Geophysics* 38(5) (1973) 826-844.
- [28] K.L. Clarkson, D.P. Woodruff, Numerical linear algebra in the streaming model, in: *Proceedings of the 41st annual ACM symposium on Theory of computing*, ACM, 2009, pp. 205-214.
- [29] K.L. Clarkson, D.P. Woodruff, Low rank approximation and regression in input sparsity time, in: *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, ACM, 2013, pp. 81-90.
- [30] A. Cohen, W. Dahmen, R. DeVore, Compressed sensing and best k -term approximation, *J. Amer. Math. Soc* 22(1) (2009) 211-231.
- [31] R.R. Coifman, M.V. Wickerhauser, Entropy-based algorithms for best basis selection, *IEEE Trans. Informat. Theory* 38(2) (1992) 713-718.
- [32] S.F. Cotter, B.D. Rao, Matching pursuit based decision-feedback equalizers, in: *IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Istanbul, Turkey, 2000.
- [33] G.B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.
- [34] S. Dasgupta, Experiments with random projections, in: *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, Morgan-Kaufmann, San Francisco, CA, USA, 2000, pp. 143-151.
- [35] I. Daubechies, Time-frequency localization operators: a geometric phase space approach, *IEEE Trans. Informat. Theory* 34(4) (1988) 605-612.
- [36] D.L. Donoho, X. Huo, Uncertainty principles and ideal atomic decomposition, *IEEE Trans. Informat. Theory* 47(7) (2001) 2845-2862.
- [37] D.L. Donoho, M. Elad, Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization, in: *Proceedings of National Academy of Sciences*, 2003, pp. 2197-2202.
- [38] D.L. Donoho, J. Tanner, Counting faces of randomly projected polytopes when the projection radically lowers dimension, *Tech. Rep. 2006-11*, Stanford University, 2006.
- [39] D.L. Donoho, J. Tanner, Precise undersampling theorems, *Proc. IEEE* 98(6) (2010) 913-924.
- [40] D.L. Donoho, B.F. Logan, Signal recovery and the large sieve, *SIAM J. Appl. Math.* 52(2) (1992) 577-591.
- [41] P.L. Dragotti, M. Vetterli, T. Blu, Sampling moments and reconstructing signals of finite rate of innovation: Shannon meets Strang-Fix, *IEEE Trans. Signal Process.* 55(5) (2007) 1741-1757.
- [42] P. Drineas, M.W. Mahoney, S. Muthukrishnan, T. Sarlós, Faster least squares approximation, *Numer. Math.* 117(2) (2011) 219-249.
- [43] M.F. Duarte, Y. Eldar, Structured compressed sensing: from theory to applications, *IEEE Trans. Signal Process.* 59(9) (2011) 4053-4085.
- [44] M.F. Duarte, R.G. Baraniuk, Kronecker compressive sensing, *IEEE Trans. Image Process.* 21(2) (2012) 494-504.
- [45] M.F. Duarte, R.G. Baraniuk, Spectral compressive sensing, *Appl. Comput. Harmonic Anal.* 35(1) (2013) 111-129.
- [46] D. Eiwen, G. Taubock, F. Hlawatsch, H.G. Feichtinger, Group sparsity methods for compressive channel estimation in doubly dispersive multicarrier systems, in: *Proceedings IEEE SPAWC*, Marrakech, Morocco, June 2010.
- [47] D. Eiwen, G. Taubock, F. Hlawatsch, H. Rauhut, N. Czink, Multichannel-compressive estimation of doubly selective channels in MIMO-OFDM systems: Exploiting and enhancing joint sparsity, in: *Proceedings International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Dallas, TX, 2010.
- [48] M. Elad, A.M. Bruckstein, A generalized uncertainty principle and sparse representations in pairs of bases, *IEEE Trans. Informat. Theory* 48(9) (2002) 2558-2567.

- [49] M. Elad, *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*, Springer, 2010.
- [50] Y.C. Eldar, G. Kutyniok, *Compressed Sensing: Theory and Applications*, Cambridge University Press, 2012.
- [51] Y.C. Eldar, *Sampling Theory: Beyond Bandlimited Systems*, Cambridge University Press, 2014.
- [52] M. Ghosh, Blind decision feedback equalization for terrestrial television receivers, *Proc. IEEE* 86(10) (1998) 2070-2081.
- [53] I.F. Gorodnitsky, B.D. Rao, Sparse signal reconstruction from limited data using FOCUSS: a re-weighted minimum norm algorithm, *IEEE Trans. Signal Process.* 45(3) (1997) 600-614.
- [54] R. Gribonval, M. Nielsen, Sparse decompositions in unions of bases, *IEEE Trans. Informat. Theory* 49(12) (2003) 3320-3325.
- [55] N. Halko, P.G. Martinsson, J.A. Tropp, Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions, *SIAM Rev.* 53(2) (2011) 217-288.
- [56] J. Haupt, W.U. Bajwa, G. Raz, R. Nowak, Toeplitz compressed sensing matrices with applications to sparse channel estimation, *IEEE Trans. Informat. Theory* 56(11) (2010) 5862-5875.
- [57] R.A. Horn, C.R. Johnson, *Matrix Analysis*, Cambridge University Press, New York, 1985.
- [58] S. Kirolos, J.N. Laska, M.B. Wakin, M.F. Duarte, D. Baron, T. Ragheb, Y. Massoud, R.G. Baraniuk, Analog to information conversion via random demodulation, in: *Proceedings of the IEEE Dallas/CAS Workshop on Design, Applications, Integration and Software*, Dallas, USA, 2006, pp. 71-74.
- [59] M. Lexa, M. Davies, J. Thompson, Reconciling compressive sampling systems for spectrally sparse continuous-time signals, *IEEE Trans. Signal Process.* 60(1) (2012) 155-171.
- [60] S. Lin, D.C. Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1983.
- [61] Y.-P. Lin, P.P. Vaidyanathan, Periodically nonuniform sampling of bandpass signals, *IEEE Trans. Circuits Syst. II* 45(3) (1998) 340-351.
- [62] Y.M. Lu, M.N. Do, Sampling signals from a union of subspaces, *IEEE Signal Process. Mag.* 25(2) (2008) 41-47.
- [63] P. Maechler, N. Felber, H. Kaeslin, A. Burg, Hardware-efficient random sampling of Fourier-sparse signals, in: *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 2012.
- [64] A. Maleki, L. Anitori, Z. Yang, R. Baraniuk, Asymptotic analysis of complex LASSO via complex approximate message passing (CAMP), *IEEE Trans. Informat. Theory*, 59(7) (2013) 4290-4308.
- [65] S. Mallat, S. Zhang, Matching pursuit in a time-frequency dictionary, *IEEE Trans. Signal Process.* 41 (1993) 3397-3415.
- [66] E. Matusiak, Y.C. Eldar, Sub-Nyquist sampling of short pulses, *IEEE Trans. Signal Process.* 60(3) (2012) 1134-1148.
- [67] S. Mendelson, A. Pajor, N. Tomczak-Jaegermann, Uniform uncertainty principle for Bernoulli and sub-Gaussian ensembles, *Construct. Approximat.* 28 (2008) 277-289.
- [68] M. Mishali, Y.C. Eldar, A. Elron, Xampling: analog data compression, in: *Proceedings Data Compression Conference*, Snowbird, Utah, USA, 2010.
- [69] M. Mishali, Y. Eldar, From theory to practice: sub-Nyquist sampling of sparse wideband analog signals, *IEEE J. Selected Topics Signal Process.* 4(2) (2010) 375-391.
- [70] M. Mishali, Y.C. Eldar, Sub-Nyquist sampling, *IEEE Signal Process. Mag.* 28(6) (2011) 98-124.
- [71] M. Mishali, Y.C. Eldar, A. Elron, Xampling: signal acquisition and processing in union of subspaces, *IEEE Trans. Signal Process.* 59(10) (2011) 4719-4734.
- [72] B.K. Natarajan, Sparse approximate solutions to linear systems, *SIAM J. Comput.* 24 (1995) 227-234.
- [73] P.A. Naylor, J. Cui, M. Brookes, Adaptive algorithms for sparse echo cancellation, *Signal Process.* 86 (2004) 1182-1192.

- [74] A.M. Pinkus, On ℓ_1 -Approximation, Cambridge Tracts in Mathematics, vol. 93, Cambridge University Press, 1989.
- [75] Q. Qiu, V.M. Patel, P. Turaga, R. Chellappa, Domain adaptive dictionary learning, in: Proceedings of the European Conference on Computer Vision (ECCV), Florence, Italy, 2012.
- [76] Y. Rivenson, A. Stern, Compressed imaging with a separable sensing operator, IEEE Signal Process. Lett. 16(6) (2009) 449-452.
- [77] A. Rondogiannis, K. Berberidis, Efficient decision feedback equalization for sparse wireless channels, IEEE Trans. Wireless Commun. 2(3) (2003) 570-581.
- [78] R. Rubinstein, A. Bruckstein, M. Elad, Dictionaries for sparse representation modeling, Proceed. IEEE 98(6) (2010) 1045-1057.
- [79] M. Rudelson, R. Vershynin, On sparse reconstruction from Fourier and Gaussian measurements, Commun. Pure Appl. Math. 61(8) (2008) 1025-1045.
- [80] F. Santosa, W.W. Symes, Linear inversion of band limited reflection seismograms, SIAM J. Sci. Comput. 7(4) (1986) 1307-1330.
- [81] T. Sarlos, Improved approximation algorithms for large matrices via random projections, in: Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on, IEEE, 2006, pp. 143-152.
- [82] P. Saurabh, C. Boutsidis, M. Magdon-Ismail, P. Drineas, Random projections for support vector machines, in: Proceedings 16th International Conference on Artificial Intelligence and Statistics (AISTATS) Scottsdale, AZ, USA, 2013.
- [83] D. Takhar, V. Bansal, M. Wakin, M. Duarte, D. Baron, K.F. Kelly, R.G. Baraniuk, A compressed sensing camera: New theory and an implementation using digital micromirrors, in: Proceedings on Computational Imaging (SPIE), San Jose, CA, 2006.
- [84] G. Tang, A. Nehorai, Performance analysis of sparse recovery based on constrained minimal singular values, IEEE Trans. Signal Process. 59(12) (2011) 5734-5745.
- [85] H.L. Taylor, S.C. Banks, J.F. McCoy, Deconvolution with the ℓ_1 norm, Geophysics 44(1) (1979) 39-52.
- [86] S. Theodoridis, K. Koutroumbas, Pattern Recognition, fourth ed., Academic Press, 2009.
- [87] Z. Tian, G.B. Giannakis, Compressed sensing for wideband cognitive radios, in: Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP), 2007, pp. 1357-1360.
- [88] R. Tibshirani, Regression shrinkage and selection via the LASSO, J. Royal. Statist. Soc. B. 58(1) (1996) 267-288.
- [89] I. Tosić, P. Frossard, Dictionary learning, IEEE Signal Process. Mag. 28(2) (2011) 27-38.
- [90] J.A. Tropp, J.N. Laska, M.F. Duarte, J.K. Romberg, G. Baraniuk, Beyond Nyquist: efficient sampling of sparse bandlimited signals, IEEE Trans. Informat. Theory 56(1) (2010) 520-544.
- [91] M. Unser, Sampling: 50 years after Shannon, Proc. IEEE 88(4) (2000) 569-587.
- [92] R.G. Vaughan, N.L. Scott, D.R. White, The theory of bandpass sampling, IEEE Trans. Signal Process. 39(9) (1991) 1973-1984.
- [93] R. Venkataramani, Y. Bresler, Perfect reconstruction formulas and bounds on aliasing error in sub-Nyquist nonuniform sampling of multiband signals, IEEE Trans. Informat. Theory 46(6) (2000) 2173-2183.
- [94] M. Vetterli, P. Marzilliano, T. Blu, Sampling signals with finite rate of innovation, IEEE Trans. Signal Process. 50(6) (2002) 1417-1428.
- [95] M. Wakin, Manifold-based signal recovery and parameter estimation from compressive measurements, 2008, preprint: <http://arxiv.org/abs/1002.1247>.
- [96] M. Wakin, S. Becker, E. Nakamura, M. Grant, E. Sovero, D. Ching, J. Yoo, J. Romberg, A. Emami-Neyestanak, E. Candes, A non-uniform sampler for wideband spectrally-sparse environments, IEEE Trans. on Emerging and Selected Topics in Circuits and Systems 2(3) (2012) 516-529.
- [97] L.R. Welch, Lower bounds on the maximum cross correlation of signals, IEEE Trans. Informat. Theory 20(3) (1974) 397-399.

- [98] S. Wright, R. Nowak, M. Figueiredo, Sparse reconstruction by separable approximation, *IEEE Trans. Signal Process.* 57(7) (2009) 2479-2493.
- [99] M. Yaghoobi, L. Daudet, M. Davies Parametric dictionary design for sparse coding, *IEEE Trans. Signal Process.* 57(12) (2009) 4800-4810.
- [100] Y. Ye, *Interior Point Methods: Theory and Analysis*, Wiley, New York, 1997.
- [101] J. Yoo, S. Becker, M. Monge, M. Loh, E. Candès, A. Emami-Neyestanak, Design and implementation of a fully integrated compressed-sensing signal acquisition system, in: 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2012, pp. 5325-5328.
- [102] Z. Yu, S. Hoyos, B.M. Sadler, Mixed-signal parallel compressed sensing and reception for cognitive radio, in: Proceedings IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP), 2008, pp. 3861-3864.

SPARSITY-AWARE LEARNING: ALGORITHMS AND APPLICATIONS

10

CHAPTER OUTLINE

10.1 Introduction	456
10.2 Sparsity-Promoting Algorithms	456
10.2.1 Greedy Algorithms	457
<i>OMP Can Recover Optimal Sparse Solutions: Sufficiency Condition</i>	459
<i>The LARS Algorithm</i>	460
<i>Compressed Sensing Matching Pursuit (CSMP) Algorithms</i>	461
10.2.2 Iterative Shrinkage/Thresholding (IST) Algorithms	462
10.2.3 Which Algorithm?: Some Practical Hints	468
10.3 Variations on the Sparsity-Aware Theme	473
10.4 Online Sparsity-Promoting Algorithms	481
10.4.1 LASSO: Asymptotic Performance	481
10.4.2 The Adaptive Norm-Weighted LASSO	483
10.4.3 Adaptive CoSaMP (AdCoSaMP) Algorithm	485
10.4.4 Sparse Adaptive Projection Subgradient Method (SpAPSM)	486
<i>Projection onto the Weighted ℓ_1 Ball</i>	488
10.5 Learning Sparse Analysis Models	491
10.5.1 Compressed Sensing for Sparse Signal Representation in Coherent Dictionaries	493
10.5.2 Cosparsity	494
10.6 A Case Study: Time-Frequency Analysis	496
<i>Gabor Transform and Frames</i>	496
<i>Time-Frequency Resolution</i>	498
<i>Gabor Frames</i>	499
<i>Time-Frequency Analysis of Echolocation Signals Emitted by Bats</i>	499
10.7 Appendix to Chapter 10: Some Hints from the Theory of Frames	503
Problems	506
<i>MATLAB Exercises</i>	507
References	508

10.1 INTRODUCTION

This chapter is the follow-up to the previous one concerning sparsity-aware learning. The emphasis now is on the algorithmic front. Following the theoretical advances concerning sparse modeling, a true scientific happening occurred in trying to derive algorithms tailored for the efficient solution of the related constrained optimization tasks. Our goal is to present the main directions that have been followed and to provide in a more explicit form some of the most popular algorithms. We will discuss batch as well as online algorithms. This chapter can also be considered as a complement to Chapter 8, where some aspects of convex optimization were introduced; a number of algorithms discussed there are also appropriate for tasks involving sparsity-related constraints/regularization.

Besides describing various algorithmic families, some variants of the basic sparsity-promoting ℓ_1 and ℓ_0 norms are discussed. Also, typical examples are shown and a case study concerning time-frequency analysis is given. Finally, some more recent theoretical advances concerning the discussion of “synthesis vs. analysis” models are provided.

10.2 SPARSITY-PROMOTING ALGORITHMS

In the previous chapter, our emphasis was on highlighting some of the most important aspects underlying the theory of sparse signal/parameter vector recovery from an underdetermined set of linear equations. We now turn our attention to the algorithmic aspects of the problem (e.g., [52, 54]). The issue now becomes that of discussing *efficient* algorithmic schemes, which can achieve the recovery of the unknown set of parameters. In Sections 9.3 and 9.5, we saw that the constrained ℓ_1 norm minimization (basis pursuit) can be solved via linear programming techniques and the LASSO task via convex optimization schemes. However, such general purpose techniques tend to be inefficient, because they often require many iterations to converge, and the respective computational resources can be excessive for practical applications, especially in high-dimensional spaces, \mathbb{R}^l . As a consequence, a huge research effort has been invested with the goal of developing efficient algorithms that are tailored to these specific tasks. Our aim here is to provide the reader with some general trends and philosophies that characterize the related activity. We will focus on the most commonly used and cited algorithms, which at the same time are structurally simple, so the reader can follow them, without deeper knowledge of optimization. Moreover, these algorithms involve, in one way or another, arguments that are directly related to points and notions we have already used while presenting the theory; thus, they can also be exploited from a pedagogical point of view in order to strengthen the reader’s understanding of the topic. We start our review with the class of batch algorithms, where all data are assumed to be available prior to the application of the algorithm, and then we will move on to online/time-adaptive schemes. Furthermore, our emphasis is on algorithms that are appropriate for any sensing matrix. This is stated in order to point out that in the literature, efficient algorithms have also been developed for specific forms of highly structured sensing matrices, and exploiting their particular structure can lead to reduced computational demands [61, 93].

There are three rough types of families along which this algorithmic activity has grown: (a) greedy algorithms, (b) iterative shrinkage schemes, and (c) convex optimization techniques. We have used the word rough because in some cases, it may be difficult to assign an algorithm to a specific family.

10.2.1 GREEDY ALGORITHMS

Greedy algorithms have a long history; see, for example, [114] for a comprehensive list of references. In the context of dictionary learning, a greedy algorithm known as *matching pursuit* was introduced in [88]. A greedy algorithm is built upon a series of *locally optimal single-term* updates. In our context, the goals are (a) to unveil the “active” columns of the sensing matrix X , that is, those columns that correspond to the nonzero locations of the unknown parameters; and (b) to estimate the respective sparse parameter vector. The set of indices that correspond to the nonzero vector components is also known as the *support*. To this end, the set of active columns of X (and the support) is increased by one at each iteration step. In the sequel, an updated estimate of the unknown sparse vector is obtained. Let us assume that, at the $(i - 1)$ th iteration step, the algorithm has selected the columns denoted as $\mathbf{x}_{j_1}, \mathbf{x}_{j_2}, \dots, \mathbf{x}_{j_{i-1}}$, with $j_1, j_2, \dots, j_{i-1} \in \{1, 2, \dots, l\}$. These indices are the elements of the currently available support, $S^{(i-1)}$. Let $X^{(i-1)}$ be the $N \times (i - 1)$ matrix, having $\mathbf{x}_{j_1}, \mathbf{x}_{j_2}, \dots, \mathbf{x}_{j_{i-1}}$ as its columns. Let also the current estimate of the solution be $\boldsymbol{\theta}^{(i-1)}$, which is a $(i - 1)$ -sparse vector, with zeros at all locations with index outside the support.

Algorithm 10.1 (Orthogonal matching pursuit (OMP)).

The algorithm is initialized with $\boldsymbol{\theta}^{(0)} := \mathbf{0}$, $\mathbf{e}^{(0)} := \mathbf{y}$, and $S^{(0)} = \emptyset$. At iteration step i , the following computational steps are performed:

1. Select the column \mathbf{x}_{j_i} of X , which is *maximally* correlated to (forms the least angle with) the respective error vector, $\mathbf{e}^{(i-1)} := \mathbf{y} - X\boldsymbol{\theta}^{(i-1)}$, that is,

$$\mathbf{x}_{j_i} : j_i := \arg \max_{j=1,2,\dots,l} \frac{|\mathbf{x}_j^T \mathbf{e}^{(i-1)}|}{\|\mathbf{x}_j\|_2}.$$

2. Update the support and the corresponding set of active columns: $S^{(i)} = S^{(i-1)} \cup \{j_i\}$, and $X^{(i)} = [X^{(i-1)}, \mathbf{x}_{j_i}]$.
3. Update the estimate of the parameter vector: Solve the least-squares (LS) problem that minimizes the norm of the error, using the active columns of X only, that is,

$$\tilde{\boldsymbol{\theta}} := \arg \min_{z \in \mathbb{R}^i} \left\| \mathbf{y} - X^{(i)} z \right\|_2^2.$$

Obtain $\boldsymbol{\theta}^{(i)}$ by inserting the elements of $\tilde{\boldsymbol{\theta}}$ in the respective locations (j_1, j_2, \dots, j_i) , which comprise the support (the rest of the elements of $\boldsymbol{\theta}^{(i)}$ retain their zero values).

4. Update the error vector

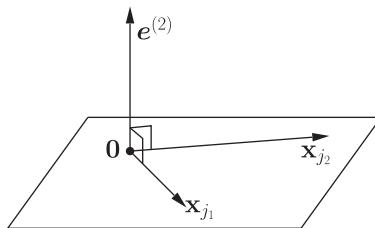
$$\mathbf{e}^{(i)} := \mathbf{y} - X\boldsymbol{\theta}^{(i)}.$$

The algorithm terminates if the norm of the error becomes less than a preselected user-defined constant, ϵ_0 . The following observations are in order.

Remarks 10.1.

- Because $\boldsymbol{\theta}^{(i)}$, in Step 3, is the result of an LS task, we know from Chapter 6 that the error vector is orthogonal to the subspace spanned by the active columns involved, that is,

$$\mathbf{e}^{(i)} \perp \text{span} \{ \mathbf{x}_{j_1}, \dots, \mathbf{x}_{j_i} \}.$$

**FIGURE 10.1**

The error vector at the i th iteration is orthogonal to the subspace spanned by the currently available set of active columns. Here is an illustration for the case of the three-dimensional Euclidean space \mathbb{R}^3 , and for $i = 2$.

This guarantees that in the next step, taking the correlation of the columns of X with $e^{(i)}$, none of the previously selected columns will be reselected; they result to zero correlation, being orthogonal to $e^{(i)}$; see [Figure 10.1](#).

- The column, which has maximal correlation (maximum absolute value of the inner product) with the currently available error vector, is the one that maximally reduces (compared to any other column) the ℓ_2 norm of the error, when y is approximated by linearly combining the currently available active columns. This is the point where the heart of the greedy strategy beats. This minimization is with respect to a *single term*, keeping the rest fixed, as they have been obtained from the previous iteration steps ([Problem 10.1](#)).
- Starting with all the components being zero, if the algorithm stops after k_0 iteration steps, the result will be a k_0 -sparse solution.
- Note that there is no optimality in this searching strategy. The only guarantee is that the ℓ_2 norm of the error vector is decreased at every iteration step. In general, there is no guarantee that the algorithm can obtain a solution close to the true one; (see, for example, [\[38\]](#)). However, under certain constraints on the structure of X , performance bounds can be obtained; see, for example, [\[37, 115, 123\]](#).
- The complexity of the algorithm amounts to $\mathcal{O}(k_0 l N)$ operations, which are contributed by the computations of the correlations, plus the demands raised by the solution of the LS task in Step 3, whose complexity depends on the specific algorithm used. The k_0 is the sparsity level of the delivered solution and, hence, the total number of iteration steps that are performed.

Another more qualitative argument that justifies the selection of the columns based on their correlation with the error vector is the following. Assume that the matrix X is orthonormal. Let $y = X\theta$. Then, y lies in the subspace spanned by the active columns of X , that is, those that correspond to the nonzero components of θ . Hence, the rest of the columns are orthogonal to y , because X is assumed to be orthonormal. Taking the correlation of y , at the first iteration step, with all the columns, it is certain that one among the active columns will be chosen. The inactive columns result in zero correlation. A similar argument holds true for all subsequent steps, because all the activity takes place in a subspace that is orthogonal to all the inactive columns of X . In the more general case, where X is not orthonormal, we can still use the correlation as a measure that quantifies geometric similarity. The smaller the

correlation/magnitude of the inner product, the more orthogonal the two vectors. This brings us back to the notion of mutual coherence, which is a measure of the maximum correlation (least angle) among the columns of X .

OMP can recover optimal sparse solutions: sufficiency condition

We have already stated that, in general, there are no guarantees that OMP will recover optimal solutions. However, when the unknown vector is sufficiently sparse, with respect to the structure of the sensing matrix X , then OMP can exactly solve the ℓ_0 minimization task in (9.20) and recover the solution in k_0 steps, where k_0 is the sparsest solution that satisfies the associated linear set of equations.

Theorem 10.1. *Let the mutual coherence (Section 9.6.1) of the sensing matrix, X , be $\mu(X)$. Assume, also, that the linear system, $y = X\theta$, accepts a solution such as*

$$\|\theta\|_0 < \frac{1}{2} \left(1 + \frac{1}{\mu(X)} \right). \quad (10.1)$$

Then, OMP guarantees recovery of the sparsest solution in $k_0 = \|\theta\|_0$ steps.

We know from Section 9.6.1 that under the previous condition, any other solution will be necessarily less sparse. Hence, there is a unique way to represent y in terms of k_0 columns of X . Without harming generality, let us assume that the true support corresponds to the first k_0 columns of X , that is,

$$y = \sum_{j=1}^{k_0} \theta_j x_j, \quad \theta_j \neq 0, \quad \forall j \in \{1, \dots, k_0\}.$$

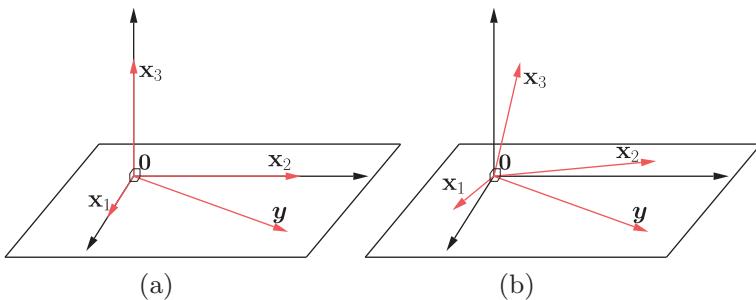
The theorem is a direct consequence of the following proposition.

Proposition 10.1. *If the condition (10.1) holds true, then the OMP algorithm will never select a column with index outside the true support; see, for example, [115], (Problem 10.2). In a more formal way, this is expressed as*

$$j_i = \arg \max_{j=1,2,\dots,l} \frac{\left| \mathbf{x}_j^T \mathbf{e}^{(i-1)} \right|}{\|\mathbf{x}_j\|_2} \in \{1, \dots, k_0\}.$$

A geometric interpretation of this proposition is the following: if the angles formed between all the possible pairs among the columns of X close to 90° (columns almost orthogonal) in the \mathbb{R}^l space, which guarantees that $\mu(X)$ is small enough, then y will lean more (form a smaller angle) toward any one of the active columns that contribute to its formation, compared to the rest that are inactive and do not participate in the linear combination that generates y . Figure 10.2 illustrates the geometry, for the extreme case of mutually orthogonal vectors (Figure 10.2a), and for the more general case where the vectors are not orthogonal, yet the angle between any pair of columns is close enough to 90° (Figure 10.2b).

In a nutshell, the previous proposition guarantees that, during the first iteration, a column corresponding to the true support will be selected. In a similar way, this is also true for all subsequent iterations. In the second step, another column, different from the previously selected column (as has already been stated), will be chosen. At step k_0 , the last remaining active column corresponding to the true support is selected, and this necessarily results to zero error. To this end, it suffices to set ϵ_0 equal to zero.

**FIGURE 10.2**

(a) In the case of an orthogonal matrix, the measurement vector \mathbf{y} will be orthogonal to any inactive column; here, \mathbf{x}_3 . (b) In the more general case, it is expected to “lean” closer (form smaller angles) to the active than to the inactive columns.

The LARS algorithm

The least angle regression (LARS) algorithm, [48], shares the first two steps with OMP. It selects j_i to be an index outside the currently available active set in order to maximize the correlation with the residual vector. However, instead of performing an LS fit to compute the nonzero components of $\boldsymbol{\theta}^{(i)}$, these are computed so that the residual will be equicorrelated with all the columns in the active set, that is,

$$|\mathbf{x}_j^T(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}^{(i)})| = \text{constant}, \quad \forall j \in S^{(i)},$$

where we have assumed that the columns of X are normalized, as is common in practice (recall, also, the [Remarks 9.4](#)). In other words, in contrast to the OMP, where the error vector is forced to be orthogonal to the active columns, LARS demands this error form equal angles with each one of them. Like OMP, it can be shown that, provided the target vector is sufficiently sparse and under incoherence of the columns of X , LARS can exactly recover the sparsest solution, [116].

A further small modification leads to the LARS-LASSO algorithm. According to this version, a previously selected index in the active set can be removed at a later stage. This gives the algorithm the potential to “recover” from a previously bad decision. Hence, this modification departs from the strict rationale that defines the greedy algorithms. It turns out that this version solves the LASSO optimization task. This algorithm is the same as the one suggested in [99] and it is known as a *homotopy* algorithm. Homotopy methods are based on a continuous transformation from one optimization task to another. The solutions to this sequence of tasks lie along a continuous parameterized path. The idea is that while the optimization tasks may be difficult to solve by themselves, one can trace this path of solutions by slowly varying the parameters. For the LASSO task, it is the λ parameter that is varying; see, for example, [4, 86, 104]. Take as an example the LASSO task in its regularized version in (9.6). For $\lambda = 0$, the task minimizes the ℓ_2 error norm and for $\lambda \rightarrow \infty$ the task minimizes the parameter vector’s ℓ_1 norm, and for this case the solution tends to zero. It turns out that the solution path, as λ changes from large to small values, is polygonal. Vertices on this solution path correspond to vectors having nonzero elements only on a subset of entries. This subset remains unchanged until λ reaches the next critical value, which corresponds to a new vertex of the polygonal path and to a new subset of potential nonzero values. Thus, the solution is obtained via this sequence of steps along this polygonal path.

Compressed sensing matching pursuit (CSMP) algorithms

Strictly speaking, the algorithms to be discussed here are not greedy, yet as stated in [93], they are at heart greedy algorithms. Instead of performing a single term optimization per iteration step, in order to increase the support by one, as is the case with OMP, these algorithms attempt to obtain first an estimate of the support and then use this information to compute a LS estimate of the target vector, constrained on the respective active columns. The quintessence of the method lies in the near-orthogonal nature of the sensing matrix, assuming that this obeys the RIP condition.

Assume that X obeys the RIP for some small enough value δ_k and sparsity level, k , of the unknown vector. Let, also, that the measurements are exact, that is, $\mathbf{y} = X\boldsymbol{\theta}$. Then, $X^T\mathbf{y} = X^TX\boldsymbol{\theta} \approx \boldsymbol{\theta}$, due to the near-orthogonal nature of X . Therefore, intuition indicates that it is not unreasonable to select, in the first iteration step, the t (a user-defined parameter) largest in magnitude components of $X^T\mathbf{y}$ as indicative of the nonzero positions of the sparse target vector. This reasoning carries on for all subsequent steps, where, at the i th iteration, the place of \mathbf{y} is taken by the residual $\mathbf{e}^{(i-1)} := \mathbf{y} - X\boldsymbol{\theta}^{(i-1)}$, where $\boldsymbol{\theta}^{(i-1)}$ indicates the estimate of the target vector at the $(i-1)$ th iteration. Basically, this could be considered as a generalization of the OMP. However, as we will soon see, the difference between the two mechanisms is more substantial.

Algorithm 10.2 (The CSMP scheme).

1. Select the value of t .
2. Initialize the algorithm: $\boldsymbol{\theta}^{(0)} = \mathbf{0}$, $\mathbf{e}^{(0)} = \mathbf{y}$.
3. For $i = 1, 2, \dots$, execute the following;

- (a) Obtain the current support:

$$S^{(i)} := \text{supp}(\boldsymbol{\theta}^{(i-1)}) \cup \left\{ \begin{array}{c} \text{indices of the } t \text{ largest in magnitude} \\ \text{components of } X^T\mathbf{e}^{(i-1)} \end{array} \right\}.$$

- (b) Select the active columns: Construct $X^{(i)}$ to comprise the active columns of X in accordance to $S^{(i)}$. Obviously, $X^{(i)}$ is an $N \times r$ matrix, where r denotes the cardinality of the support set $S^{(i)}$.
- (c) Update the estimate of the parameter vector: solve the LS task

$$\tilde{\boldsymbol{\theta}} := \arg \min_{\mathbf{z} \in \mathbb{R}^r} \|\mathbf{y} - X^{(i)}\mathbf{z}\|_2^2.$$

Obtain $\hat{\boldsymbol{\theta}}^{(i)} \in \mathbb{R}^l$ having the r elements of $\tilde{\boldsymbol{\theta}}$ in the respective locations, as indicated by the support, and the rest of the elements being zero.

- (d) $\boldsymbol{\theta}^{(i)} := H_k(\hat{\boldsymbol{\theta}}^{(i)})$. The mapping H_k denotes the *hard thresholding* function; that is, it returns a vector with the k largest in magnitude components of the argument, and the rest are forced to zero.
- (e) Update the error vector: $\mathbf{e}^{(i)} = \mathbf{y} - X\boldsymbol{\theta}^{(i)}$.

The algorithm requires as input the sparsity level k . Iterations carry on until a halting criterion is met. The value of t , which determines the largest in magnitude values in Steps 1 and 3a, depends on the specific algorithm. In CoSaMP (compressive sampling matching pursuit [93]), $t = 2k$ (Problem 10.3), and in the SP (subspace pursuit [33]), $t = k$.

Having stated the general scheme, a major difference with OMP becomes readily apparent. In OMP, only one column is selected per iteration step. Moreover, this remains in the active set for all subsequent steps. If, for some reason, this was not a good choice, the scheme cannot recover from such a bad

decision. In contrast, the support and, hence, the active columns of X are continuously updated in CSMP, and the algorithm has the ability to correct a previously bad decision, as more information is accumulated and iterations progress. In [33], it is shown that if the measurements are exact ($\mathbf{y} = X\boldsymbol{\theta}$), then SP can recover the k -sparse true vector in a finite number of iteration steps, provided that X satisfies the RIP with $\delta_{3k} < 0.205$. If the measurements are noisy, performance bounds have been derived, which hold true for $\delta_{3k} < 0.083$. For the CoSaMP, performance bounds have been derived for $\delta_{4k} < 0.1$.

10.2.2 ITERATIVE SHRINKAGE/THRESHOLDING (IST) ALGORITHMS

This family of algorithms also have a long history; see, for example, [44, 69, 70, 73]. However, in the “early” days, most of the developed algorithms had some sense of heuristic flavor, without establishing a clear bridge with optimizing a cost function. Later attempts were substantiated by sound theoretical arguments concerning issues such as convergence and convergence rate [31, 34, 50, 56].

The general form of this algorithmic family has a striking resemblance to the classical linear algebra iterative schemes for approximating the solution of large linear systems of equations, known as *stationary iterative* or *iterative relaxation* methods. The classical Gauss-Seidel and Jacobi algorithms (e.g., [65]), in numerical analysis can be considered members of this family. Given a linear system of l equations with l unknowns, $\mathbf{z} = \mathbf{Ax}$, the basic iteration at step i has the following form:

$$\begin{aligned}\mathbf{x}^{(i)} &= (\mathbf{I} - \mathbf{Q}\mathbf{A})\mathbf{x}^{(i-1)} + \mathbf{Q}\mathbf{z} \\ &= \mathbf{x}^{(i-1)} + \mathbf{Q}\mathbf{e}^{(i-1)}, \quad \mathbf{e}^{(i-1)} := \mathbf{z} - \mathbf{Ax}^{(i-1)},\end{aligned}$$

which does not come as a surprise. It is of the same form as most of the iterative schemes for numerical solutions! The matrix \mathbf{Q} is chosen in order to guarantee convergence, and different choices lead to different algorithms with their pros and cons. It turns out that this algorithmic form can also be applied to underdetermined systems of equations, $\mathbf{y} = \mathbf{X}\boldsymbol{\theta}$, with a “minor” modification, which is imposed by the sparsity constraint of the target vector. This leads to the following general form of iterative computation:

$$\boldsymbol{\theta}^{(i)} = T_i(\boldsymbol{\theta}^{(i-1)} + \mathbf{Q}\mathbf{e}^{(i-1)}), \quad \mathbf{e}^{(i-1)} = \mathbf{y} - \mathbf{X}\boldsymbol{\theta}^{(i-1)},$$

starting from an initial guess of $\boldsymbol{\theta}^{(0)}$ (usually $\boldsymbol{\theta}^{(0)} = \mathbf{0}$, $\mathbf{e}^{(0)} = \mathbf{y}$). In certain cases, \mathbf{Q} can be made to be iteration-dependent. The function $T_i(\cdot)$ is a nonlinear thresholding function that is applied *entrywise*, that is, *component-wise*. Depending on the specific scheme, this can be either the hard thresholding function, denoted as H_k , or the soft thresholding function, denoted as S_α . Hard thresholding, as we already know, keeps the k largest components of a vector unaltered and sets the rest equal to zero. Soft thresholding was introduced in Section 9.3. All components with magnitude less than a threshold value, α , are forced to zero and the rest are reduced in magnitude by α ; that is, the j th component of a vector, $\boldsymbol{\theta}$, after soft thresholding becomes

$$(S_\alpha(\boldsymbol{\theta}))_j = \text{sgn}(\theta_j)(|\theta_j| - \alpha)_+.$$

Depending on (a) the choice of T_i , (b) the specific value of the parameter k or α , and (c) the matrix \mathbf{Q} , different instances occur. The most common choice for \mathbf{Q} is $\mu\mathbf{X}^T$, and the generic form of the main iteration becomes

$$\boldsymbol{\theta}^{(i)} = T_i(\boldsymbol{\theta}^{(i-1)} + \mu\mathbf{X}^T\mathbf{e}^{(i-1)}),$$

(10.2)

where μ is a relaxation (user-defined) parameter, which can also be left to vary with each iteration step. The choice of X^T is intuitively justified, once more, by the near-orthogonal nature of X . For the first iteration step and for a linear system of the form $y = X\theta$, starting from a zero initial guess, we have $X^T y = X^T X \theta \approx \theta$ and we are close to the solution.

Although intuition is most important in scientific research, it is not enough, by itself, to justify decisions and actions. The generic scheme in (10.2) has been reached from different paths, following different perspectives that lead to different choices of the involved parameters. Let us spend some more time on that, with the aim of making the reader more familiar with techniques that address optimization tasks of nondifferentiable loss functions. The term in the parenthesis in (10.2) coincides with the gradient descent iteration step if the cost function were the unregularized LS loss, that is,

$$J(\theta) = \frac{1}{2} \|y - X\theta\|_2^2.$$

In this case, the gradient descent rationale leads to

$$\begin{aligned}\theta^{(i-1)} - \mu \frac{\partial J(\theta^{(i-1)})}{\partial \theta} &= \theta^{(i-1)} - \mu X^T (X\theta^{(i-1)} - y) \\ &= \theta^{(i-1)} + \mu X^T e^{(i-1)}.\end{aligned}$$

The gradient descent can alternatively be viewed as the result of minimizing a regularized version of the linearized cost function (verify it),

$$\begin{aligned}\theta^{(i)} = \arg \min_{\theta \in \mathbb{R}^l} \left\{ J(\theta^{(i-1)}) + (\theta - \theta^{(i-1)})^T \frac{\partial J(\theta^{(i-1)})}{\partial \theta} \right. \\ \left. + \frac{1}{2\mu} \|\theta - \theta^{(i-1)}\|_2^2 \right\}. \quad (10.3)\end{aligned}$$

One can adopt this view of the gradient descent philosophy as a kick-off point to minimize iteratively the following LASSO task,

$$\min_{\theta \in \mathbb{R}^l} \left\{ L(\theta, \lambda) = \frac{1}{2} \|y - X\theta\|_2^2 + \lambda \|\theta\|_1 = J(\theta) + \lambda \|\theta\|_1 \right\}.$$

The difference now is that the loss function comprises two terms: one that is smooth (differentiable) and a nonsmooth one. Let the current estimate be $\theta^{(i-1)}$. The updated estimate is obtained by

$$\begin{aligned}\theta^{(i)} = \arg \min_{\theta \in \mathbb{R}^l} \left\{ J(\theta^{(i-1)}) + (\theta - \theta^{(i-1)})^T \frac{\partial J(\theta^{(i-1)})}{\partial \theta} \right. \\ \left. + \frac{1}{2\mu} \|\theta - \theta^{(i-1)}\|_2^2 + \lambda \|\theta\|_1 \right\},\end{aligned}$$

which, after ignoring constants, is equivalently written as

$$\theta^{(i)} = \arg \min_{\theta \in \mathbb{R}^l} \left\{ \frac{1}{2} \|\theta - \tilde{\theta}\|_2^2 + \lambda \mu \|\theta\|_1 \right\} \quad (10.4)$$

where

$$\tilde{\theta} := \theta^{(i-1)} - \mu \frac{\partial J(\theta^{(i-1)})}{\partial \theta}. \quad (10.5)$$

Following exactly the same steps as those that led to the derivation of (9.13) from (9.6) (after replacing $\hat{\theta}_{LS}$ with $\tilde{\theta}$), we obtain

$$\theta^{(i)} = S_{\lambda, \mu}(\tilde{\theta}) = S_{\lambda, \mu} \left(\theta^{(i-1)} - \mu \frac{\partial J(\theta^{(i-1)})}{\partial \theta} \right) \quad (10.6)$$

$$= S_{\lambda, \mu} \left(\theta^{(i-1)} + \mu X^T e^{(i-1)} \right). \quad (10.7)$$

This is very interesting and practically useful. The only effect of the presence of the nonsmooth ℓ_1 norm in the loss function is an extra simple thresholding operation, which as we know is an operation performed *individually* on each component. It can be shown (e.g., [11, 95]), that this algorithm converges to a minimizer θ_* of the LASSO (9.6), provided that $\mu \in (0, 1/\lambda_{\max}(X^T X))$, where $\lambda_{\max}(\cdot)$ denotes the maximum eigenvalue of $X^T X$. The convergence rate is dictated by the rule

$$L(\theta^{(i)}, \lambda) - L(\theta_*, \lambda) \approx O(1/i),$$

which is known as *sublinear* global rate of convergence. Moreover, it can be shown that

$$L(\theta^{(i)}, \lambda) - L(\theta_*, \lambda) \leq \frac{C \|\theta^{(0)} - \theta_*\|_2^2}{2i}.$$

The latter result indicates that if one wants to achieve an accuracy of ϵ , then this can be obtained by at most $\left\lfloor \frac{C \|\theta^{(0)} - \theta_*\|_2^2}{2\epsilon} \right\rfloor$ iterations, where $\lfloor \cdot \rfloor$ denotes the floor function.

In [34], (10.2) was obtained from a nearby corner, building upon arguments from the classical *proximal-point* methods in optimization theory (e.g., [105]). The original LASSO regularized cost function is modified to the *surrogate objective*,

$$J(\theta, \tilde{\theta}) = \frac{1}{2} \|y - X\theta\|_2^2 + \lambda \|\theta\|_1 + \frac{1}{2} d(\theta, \tilde{\theta}),$$

where

$$d(\theta, \tilde{\theta}) := c \|\theta - \tilde{\theta}\|_2^2 - \|X\theta - X\tilde{\theta}\|_2^2.$$

If c is appropriately chosen (larger than the largest eigenvalue of $X^T X$), the surrogate objective is guaranteed to be strictly convex. Then it can be shown (Problem 10.4) that the minimizer of the surrogate objective is given by

$$\hat{\theta} = S_{\lambda/c} \left(\tilde{\theta} + \frac{1}{c} X^T (y - X\tilde{\theta}) \right). \quad (10.8)$$

In the iterative formulation, $\tilde{\theta}$ is selected to be the previously obtained estimate; in this way, one tries to keep the new estimate close to the previous one. The procedure readily results to our generic scheme in (10.2), using soft thresholding with parameter λ/c . It can be shown that such a strategy converges to a minimizer of the original LASSO problem. The same algorithm was reached in [56], using *majorization-minimization* techniques from optimization theory. So, from this perspective, the IST family has strong ties with algorithms that belong to the convex optimization category.

In [118], the *sparse reconstruction by separable approximation* (SpaRSA) algorithm is proposed, which is a modification of the standard IST scheme. The starting point is (10.3); however, the

multiplying factor, $\frac{1}{2\mu}$, instead of being constant is now allowed to change from iteration to iteration according to a rule. This results in a speedup in the convergence of the algorithm. Moreover, inspired by the homotopy family of algorithms, where λ is allowed to vary, SpaRSA can be extended to solve a sequence of problems that are associated with a corresponding sequence of values of λ . Once a solution has been obtained for a particular value of λ , it can be used as a “warm-start” for a nearby value. Solutions can therefore be computed for a range of values, at a small extra computational cost, compared to solving for a single value from a “cold start.” This technique abides with the *continuation strategy*, which has been used in the context of other algorithms as well (e.g., [66]). Continuation has been shown to be a very successful tool to increase the speed of convergence.

An interesting variation of the basic IST scheme has been proposed in [11], which improves the convergence rate to $O(1/i^2)$, by only a simple modification with almost no extra computational burden. The scheme is known as *fast iterative shrinkage-thresholding algorithm* (FISTA). This scheme is an evolution of [96], which introduced the basic idea for the case of differentiable costs, and consists of the following steps:

$$\boldsymbol{\theta}^{(i)} = S_{\lambda, \mu} \left(\mathbf{z}^{(i)} + \mu X^T (\mathbf{y} - X\mathbf{z}^{(i)}) \right),$$

$$\mathbf{z}^{(i+1)} := \boldsymbol{\theta}^{(i)} + \frac{t_i - 1}{t_{i+1}} (\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{(i-1)}),$$

where

$$t_{i+1} := \frac{1 + \sqrt{1 + 4t_i^2}}{2},$$

with initial points $t_1 = 1$ and $\mathbf{z}^{(1)} = \boldsymbol{\theta}^{(0)}$. In words, in the thresholding operation, $\boldsymbol{\theta}^{(i-1)}$ is replaced by $\mathbf{z}^{(i)}$, which is a specific linear combination of two successive updates of $\boldsymbol{\theta}$. Hence, at a marginal increase of the computational cost, a substantial increase in convergence speed is achieved.

In [17] the hard thresholding version has been used, with $\mu = 1$, and the thresholding function H_k uses the sparsity level k of the target solution that is assumed to be known. In a later version, [19], the relaxation parameter is left to change so that, at each iteration step, the error is maximally reduced. It has been shown that the algorithm converges to a local minimum of the cost function $\|\mathbf{y} - X\boldsymbol{\theta}\|_2$, under the constraint that $\boldsymbol{\theta}$ is a k -sparse vector. Moreover, the latter version is a stable one and it results to a near optimal solution if a form of RIP is fulfilled.

A modified version of the generic scheme given in (10.2), which evolves along the lines of [84], obtains the updates component-wise, one vector component at a time. Thus, a “full” iteration consists of l steps. The algorithm is known as *coordinate descent* and its basic iteration has the form (Problem 10.5),

$$\theta_j^{(i)} = S_{\lambda/\|\mathbf{x}_j\|_2^2} \left(\theta_j^{(i-1)} + \frac{\mathbf{x}_j^T \boldsymbol{\epsilon}^{(i-1)}}{\|\mathbf{x}_j\|_2^2} \right), \quad j = 1, 2, \dots, l. \quad (10.9)$$

This algorithm replaces the constant c , in the previously reported soft thresholding algorithm, with the norm of the respective column of X , if the columns of X are not normalized to unit norm. It has been shown that the parallel coordinate descent algorithm also converges to a LASSO minimizer of (9.6) [50]. Improvements of the algorithm, using line search techniques to determine the most descent direction for each iteration, have also been proposed; see [124].

The main contribution to the complexity for the iterative shrinkage algorithmic family comes from the two matrix-vector products, which amounts to $\mathcal{O}(NI)$, unless X has a special structure (e.g., DFT), that can be exploited to reduce the load.

In [85], the two stage thresholding (TST) scheme is presented, which brings together arguments from the iterative shrinkage family and the OMP. This algorithmic scheme involves two stages of thresholding. The first step is exactly the same as in (10.2). However, this is now used only for determining “significant” nonzero locations, just as in compressed sensing matching pursuit (CSMP) algorithms, presented in the previous subsection. Then, an LS problem is solved to provide the updated estimate, under the constraint of the available support. This is followed by a second step of thresholding. The thresholding operations in the two stages can be different. If hard thresholding, H_k , is used in both steps, this results to the algorithm proposed in [58]. For this latter scheme, convergence and performance bounds are derived if the RIP holds for $\delta_{3k} < 0.58$. In other words, the basic difference between the TST and CSMP approaches is that, in the latter case, the most significant nonzero coefficients are obtained by looking at the correlation term $X^T \mathbf{e}^{(i-1)}$ and in the TST family at $\boldsymbol{\theta}^{(i-1)} + \mu X^T \mathbf{e}^{(i-1)}$. The differences among different approaches can be minor and the crossing lines between the different algorithmic categories are not necessarily crispy and clear. However, from a practical point of view, sometimes small differences may lead to substantially improved performance.

In [41], the IST algorithmic framework was treated as a *message passing* algorithm in the context of graphical models (Chapter 15), and the following modified recursion was obtained:

$$\boldsymbol{\theta}^{(i)} = T_i \left(\boldsymbol{\theta}^{(i-1)} + X^T \mathbf{z}^{(i-1)} \right), \quad (10.10)$$

$$\mathbf{z}^{(i-1)} = \mathbf{y} - X \boldsymbol{\theta}^{(i-1)} + \frac{1}{\alpha} \overline{\mathbf{z}^{(i-2)} T'_i (\boldsymbol{\theta}^{(i-2)} + X^T \mathbf{z}^{(i-2)})}, \quad (10.11)$$

where $\alpha = \frac{N}{l}$, the overbar denotes the average over all the components of the corresponding vector and T'_i denotes the respective derivative of the component-wise thresholding rule. The extra term on the right-hand side in (10.11), which now appears, turns out to provide a performance improvement of the algorithm, compared to the IST family, with respect to the undersampling-sparsity trade-off (Section 10.2.3). Note that T_i is iteration-dependent and it is controlled via the definition of certain parameters. A parameterless version of it has been proposed in [91]. A detailed treatment on the message passing algorithms can be found in [2].

Remarks 10.2.

- The iteration in (10.6) bridges the IST algorithmic family with another powerful tool in convex optimization, which builds upon the notion of *proximal mapping* or *Moreau envelopes* (see Chapter 8 and, e.g., [32, 105]). Given a convex function $h : \mathbb{R}^l \rightarrow \mathbb{R}$, and a $\mu > 0$, the proximal mapping, $\text{Prox}_{\mu h} : \mathbb{R}^l \mapsto \mathbb{R}^l$, with respect to h , and of index μ , is defined as the (unique) minimizer

$$\text{Prox}_{\mu h}(\mathbf{x}) := \arg \min_{\mathbf{v} \in \mathbb{R}^l} \left\{ h(\mathbf{v}) + \frac{1}{2\mu} \|\mathbf{x} - \mathbf{v}\|_2^2 \right\}, \quad \forall \mathbf{x} \in \mathbb{R}^l. \quad (10.12)$$

Let us now assume that we want to minimize a convex function, which is given as the sum

$$f(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + h(\boldsymbol{\theta}),$$

where J is convex and differentiable, and h is also convex, but not necessarily smooth. Then, it can be shown (Section 8.14) that the following iterations converge to a minimizer of f ,

$$\boldsymbol{\theta}^{(i)} = \text{Prox}_{\mu h} \left(\boldsymbol{\theta}^{(i-1)} - \mu \frac{\partial J(\boldsymbol{\theta}^{(i-1)})}{\partial \boldsymbol{\theta}} \right), \quad (10.13)$$

where $\mu > 0$, and it can also be made iteration dependent, that is, $\mu_i > 0$. If we now use this scheme to minimize our familiar cost,

$$J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_1,$$

we obtain (10.6); this is so because the proximal operator of $h(\boldsymbol{\theta}) := \lambda \|\boldsymbol{\theta}\|_1$ is shown [31, 32], Section 8.13 to be identical to the soft thresholding operator, that is,

$$\text{Prox}_h(\boldsymbol{\theta}) = S_\lambda(\boldsymbol{\theta}).$$

In order to feel more comfortable with this operator, note that if $h(\boldsymbol{x}) \equiv 0$, its proximal operator is equal to \boldsymbol{x} , and in this case (10.13) becomes our familiar gradient descent algorithm.

- All the nongreedy algorithms that have been discussed so far have been developed to solve the task defined in the formulation (9.6). This is mainly because this is an easier task to solve; once λ has been fixed, it is an unconstrained optimization task. However, there are algorithms that have been developed to solve the alternative formulations.

The NESTA algorithm has been proposed in [12] and solves the task in its (9.8) formulation.

Adopting this path can have an advantage because ϵ may be given as an estimate of the uncertainty associated with the noise, which can readily be obtained in a number of practical applications. In contrast, selecting a priori the value for λ is more intricate. In [28], the value $\lambda = \sigma_\eta \sqrt{2 \ln l}$, where σ_η is the noise standard deviation, is argued to have certain optimality properties; however, this argument hinges on the assumption of the orthogonality of X . NESTA relies heavily on Nesterov's generic scheme [96], hence its name. The original Nesterov's algorithm performs a constrained minimization of a smooth convex function $f(\boldsymbol{\theta})$, that is,

$$\min_{\boldsymbol{\theta} \in Q} f(\boldsymbol{\theta}),$$

where Q is a convex set, and in our case this is associated with the quadratic constraint in (9.8). The algorithm consists of three basic steps. The first one, involves an auxiliary variable, and is similar with the step in (10.3), i.e.,

$$\boldsymbol{w}^{(i)} = \arg \min_{\boldsymbol{\theta} \in Q} \left\{ \left(\boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)} \right)^T \frac{\partial f(\boldsymbol{\theta}^{(i-1)})}{\partial \boldsymbol{\theta}} + \frac{L}{2} \left\| \boldsymbol{\theta} - \boldsymbol{\theta}^{(i-1)} \right\|_2^2 \right\}, \quad (10.14)$$

where L is an upper bound on the Lipschitz coefficient, which the gradient of f has to satisfy. The difference with (10.3) is that the minimization is now a constrained one. However, Nesterov has also added a second step involving another auxiliary variable, $\boldsymbol{z}^{(i)}$, which is computed in a similar way as $\boldsymbol{w}^{(i)}$, but the linearized term is now replaced by a weighted cumulative gradient,

$$\sum_{k=0}^{i-1} \alpha_k \left(\boldsymbol{\theta} - \boldsymbol{\theta}^{(k)} \right)^T \frac{\partial f(\boldsymbol{\theta}^{(k)})}{\partial \boldsymbol{\theta}}.$$

The effect of this term is to smooth out the “zigzagging” of the path toward the solution, whose effect is to increase significantly the convergence speed. The final step of the scheme involves an averaging of the previously obtained variables,

$$\boldsymbol{\theta}^{(i)} = t_i \mathbf{z}^{(i)} + (1 - t_i) \mathbf{w}^{(i)}.$$

The values of the parameters α_k , $k = 0, \dots, i - 1$, and t_i result from the theory so that convergence is guaranteed. As was the case with its close relative FISTA, the algorithm enjoys an $O(1/i^2)$ convergence rate. In our case, where the function to be minimized, $\|\boldsymbol{\theta}\|_1$, is not smooth, NESTA uses a smoothed prox-function of it. Moreover, it turns out that close-form updates are obtained for $\mathbf{z}^{(i)}$ and $\mathbf{w}^{(i)}$. If X is chosen in order to have orthonormal rows, the complexity per iteration is $O(l)$ plus the computations needed for performing the product $X^T X$, which is the most computationally thirsty part. However, this complexity can substantially be reduced if the sensing matrix is chosen to be a submatrix of a unitary transform, which admits fast matrix-vector product computations (e.g., a subsampled DFT matrix). For example, for the case of a subsampled DFT matrix, the complexity amounts to $O(l)$ plus the load to perform the two fast fourier transforms (FFT). Moreover, the continuation strategy can also be employed to accelerate convergence. In [12], it is demonstrated that NESTA exhibits good accuracy results, while retaining a complexity that is competitive with algorithms developed around the (9.6) formulation and scales in an affordable way for large-size problems. Furthermore, NESTA, and in general Nesterov's scheme, enjoy a generality that allows their use for other optimization tasks as well.

- The task in (9.7) has been considered in [14] and [99]. In the former, the algorithm comprises a projection on the ℓ_1 ball $\|\boldsymbol{\theta}\|_1 \leq \rho$ (see also Section 10.4.4) per iteration step. The most computationally dominant part of the algorithm consists of matrix-vector products. In [99], a homotopy algorithm is derived for the same task, where now the bound ρ becomes the homotopy parameter that is left to vary. This algorithm is also referred to as the LARS-LASSO, as has already been reported.

10.2.3 WHICH ALGORITHM?: SOME PRACTICAL HINTS

We have already discussed a number of algorithmic alternatives to obtain solutions to the ℓ_0 or ℓ_1 norm minimization tasks. Our focus was on schemes whose computational demands are rather low and that scale well to very large problem sizes. We have not touched more expensive methods such as interior point methods for solving the ℓ_1 convex optimization task. A review of such methods is provided in [72]. Interior point methods evolve along the Newton-type recursion and their complexity per iteration step is at least of the order $\mathcal{O}(l^3)$. As is most often the case, there is a trade-off. Schemes of higher complexity tend to result in enhanced performance. However, such schemes become impractical in problems of large size. Some examples of other algorithms that were not discussed can be found in [14, 35, 118, 121]. Talking about complexity, it has to be pointed out that what really matters at the end is not so much the complexity per iteration step, but the overall required resources in computer time/memory for the algorithm to converge to a solution within a specified accuracy. For example, an algorithm may be of low complexity per iteration step, but it may need an excessive number of iterations to converge.

Computational load is only one among a number of indices that characterize the performance of an algorithm. Throughout the book so far, we have considered a number of other performance measures, such as convergence rate, tracking speed (for the adaptive algorithms), and stability with respect to the presence of noise and/or finite word length computations. No doubt, all these performance measures are also of interest here, too. However, there is an additional aspect that is of particular importance when quantifying performance of sparsity-promoting algorithms. This is related to the *undersampling-sparsity trade-off* or the *phase transition curve*.

One of the major issues on which we focused in [Chapter 9](#) was to derive and present the conditions that guarantee uniqueness of the ℓ_0 minimization and its equivalence with the ℓ_1 minimization task, under an underdetermined set of measurements/observations, $y = X\theta$, for the recovery of sparse enough signals/vectors. While discussing the various algorithms in this section, we reported a number of different RIP-related conditions that some of the algorithms have to satisfy in order to recover the target sparse vector. As a matter of fact, it has to be admitted that this was quite confusing, because each algorithm had to satisfy its own conditions. In addition, in practice, these conditions are not easily to be verified. Although such results are no doubt important to establish convergence, and make us more confident, and help us better understand why and how an algorithm works, one needs further experimental evidence in order to establish good performance bounds for an algorithm. Moreover, all the conditions we have dealt with, including coherence and RIP, are sufficient conditions. In practice, it turns out that sparse signal recovery is possible with sparsity levels much higher than those predicted by the theory, for given N and l . Hence, proposing a new algorithm or selecting an algorithm from an available palette, one has to demonstrate experimentally the range of sparsity levels that can be recovered by the algorithm, as a percentage of the number of measurements and the dimensionality. Thus, in order to select an algorithm, one should cast her/his vote for the algorithm that, for given l and N , has the potential to recover k -sparse vectors with k being as high as possible for most of the cases, that is, with *high probability*.

[Figure 10.3](#) illustrates the type of curve that is expected to result in practice. The vertical axis is the probability of exact recovery of a target k -sparse vector and the horizontal axis shows the ratio k/N , for a given number of measurements, N , and the dimensionality of the ambient space, l . Three curves are shown. The red ones correspond to the same algorithm, for two different values of the dimensionality, l , and the gray one corresponds to another algorithm. Curves of this shape are expected to result from experiments of the following setup. Assume that we are given a sparse vector, θ_o , with k nonzero components in the l -dimensional space. Using a sensing matrix X , we generate N measurements $y = X\theta_o$. The experiment is repeated a number of M times, each time using a different realization of the sensing matrix and a different k -sparse vector. For each instance, the algorithm is run to recover

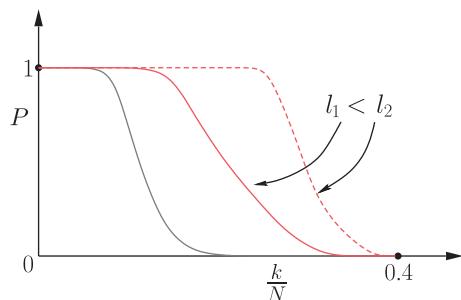


FIGURE 10.3

For any algorithm, the transition between the regions of 100% success and of complete failure is very sharp. For the algorithm corresponding to the red curve, this transition occurs at higher sparsity values and, from this point of view, it is a better algorithm than the one associated with the gray curve. Also, given an algorithm, the higher the dimensionality the higher the sparsity level where this transition occurs, as indicated by the two red curves.

the target sparse vector. This is not always possible. We count the number, m , of successful recoveries, and compute the corresponding percentage of successful recovery (probability), m/M , which is plotted on the vertical axis of [Figure 10.3](#). The procedure is repeated for a different value of k , $1 \leq k \leq N$. A number of issues now jump onto the stage: (a) how one selects the ensemble of sensing matrices and (b) how one selects the ensemble of sparse vectors. There are different scenarios, and some typical examples are described next.

1. The $N \times l$ sensing matrices X are formed by:
 - (a) Different i.i.d. realizations with elements drawn from a Gaussian $\mathcal{N}(0, 1/N)$.
 - (b) Different i.i.d. realizations from the uniform distribution on the unit sphere in \mathbb{R}^N , which is also known as the uniform spherical ensemble.
 - (c) Different i.i.d. realizations with elements drawn from Bernoulli type distributions.
 - (d) Different i.i.d. realizations of partial Fourier matrices, each time using a different set of N rows.
2. The k -sparse target vector θ_o is formed by selecting the locations of (at most) k nonzero elements randomly, by “tossing a coin” with probability $p = k/l$, and filling the values of the nonzero elements according to a statistical distribution (e.g., Gaussian, uniform, double exponential, Cauchy).

Other scenarios are also possible. Some authors set all nonzero values to one [16], or to ± 1 , with the randomness imposed on the choice of the sign. It must be stressed that the performance of an algorithm may vary significantly under different experimental scenarios, and this may be indicative of the stability of an algorithm. In practice, a user may be interested in a specific scenario that is more representative of the available data.

Looking at [Figure 10.3](#), the following conclusions are in order. In all curves, there is a sharp transition between two levels, from the 100% success to the 0% success. Moreover, the higher the dimensionality, the sharper the transition. This has also been shown theoretically in [40]. For the algorithm corresponding to the red curves, this transition occurs at higher values of k , compared to the algorithm that generates the curve drawn in gray. Provided that the computational complexity of the “red” algorithm can be accommodated by the resources that are available for a specific application, this seems to be the more sensible choice between the two algorithms. However, if the resources are limited, concessions are unavoidable.

Another way to “interrogate” and demonstrate the performance of an algorithm, with respect to its robustness to the range of values of sparsity levels that can be successfully recovered, is via the *phase transition curve*. To this end define

- $\alpha := \frac{N}{l}$, which is a normalized measure of the problem indeterminacy,
- $\beta := \frac{k}{N}$, which is a normalized measure of sparsity.

In the sequel, plot a graph having $\alpha \in [0, 1]$ in the horizontal axis and $\beta \in [0, 1]$ in the vertical one. For each point, (α, β) , in the $[0, 1] \times [0, 1]$ region, compute the probability of the algorithm to recover a k -sparse target vector. In order to compute the probability, one has to adopt one of the previously stated scenarios. In practice, one has to form a grid of points that cover densely enough the region $[0, 1] \times [0, 1]$ in the graph. Use a varying intensity level scale to color the corresponding (α, β) point. Black corresponds to probability one and red to probability zero. [Figure 10.4](#) illustrates the type of graph that is expected to be recovered in practice for large values of l ; that is, the transition from the

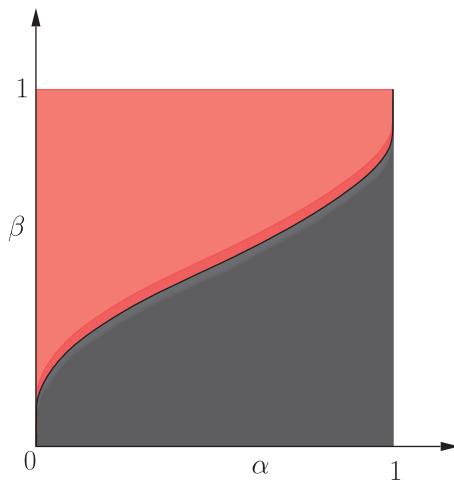
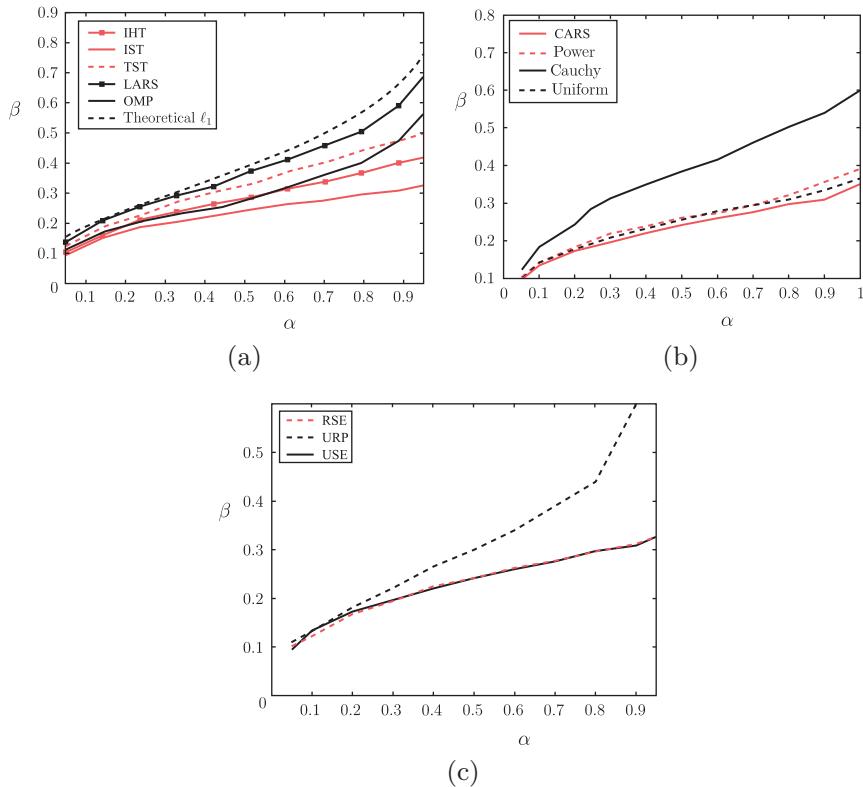


FIGURE 10.4

Typical phase transition behavior of a sparsity-promoting algorithm. Black corresponds to 100% success of recovering the sparsest solution, and red to 0%. For high-dimensional spaces, the transition is very sharp, as is the case in the figure. For lower dimensionality values, the transition from black to red is smoother and involves a region of varying color intensity.

region (phase) of “success” (black) to that of “fail” (red) is very sharp. As a matter of fact, there is a curve that separates the two regions. The theoretical aspects of this curve have been studied in the context of combinatorial geometry in [40] for the asymptotic case, $l \rightarrow \infty$, and in [42] for finite values of l . Observe that the larger the value of α (larger percentage of measurements), the larger the value of β at which the transition occurs. This is in line with what we have said so far in this chapter, and the problem gets increasingly difficult as one moves up and to the left in the graph. In practice, for smaller values of l , the transition region from red to black is smoother, and it gets narrower as l increases. In such cases, one can draw an approximate curve that separates the “success” and “fail” regions, using regression techniques (see, e.g., [85]).

The reader may already be aware of the fact that, so far, we have avoided talking about the performance of individual algorithms. We have just discussed some “typical” behavior that algorithms tend to exhibit in practice. What the reader might have expected is to discuss comparative performance tests and draw related conclusions. We have not done that because we feel that it is too early in time to have “definite” performance conclusions, and this field is still in an early stage. Most authors compare their newly suggested algorithm with a few other algorithms, usually within a certain algorithmic family and, more important, under some specific scenarios, where the advantages of the newly suggested algorithm are documented. However, the performance of an algorithm can change significantly by changing the experimental scenario under which the tests are carried out. The most comprehensive comparative performance study so far has been carried out in [85]. However, even in this one, the scenario of exact measurements has been considered and there are no experiments concerning the robustness of individual algorithms to the presence of noise. It is important to say that this study involved

**FIGURE 10.5**

(a) The obtained phase transition curves for different algorithms under the same experimental scenario, together with the theoretical one. (b) Phase transition curve for the IST algorithm under different experimental scenarios for generating the target sparse vector. (c) The phase transition for the IST algorithms under different experimental scenarios for generating the sensing matrix X .

a huge effort of computation. We will comment on some of the findings from this study, which will also reveal to the reader that different experimental scenarios can significantly affect the performance of an algorithm.

Figure 10.5a shows the obtained phase transition curves for (a) the iterative hard thresholding (IHT); (b) the iterative soft thresholding (IST) scheme of (10.2); (c) the two-stage-thresholding (TST) scheme, as discussed earlier; (d) the LARS algorithm; and (e) the OMP algorithm, together with the theoretically obtained one using ℓ_1 minimization. All algorithms were tuned with the optimal values, with respect to the required user-defined parameters, after extensive experimentation. The results in the figure correspond to the uniform spherical scenario for the generation of the sensing matrices. Sparse vectors were generated according to the ± 1 scenario for the nonzero coefficients. The interesting observation is that, although the curves deviate from each other as they move to larger values of β , for smaller values, the differences in their performance become less and less. This is also true for computationally simple

schemes such as the IHT one. The performance of LARS is close to the optimal one. However, this comes at the cost of computational increase. The required computational time for achieving the same accuracy, as reported in [85], favors the TST algorithm. In some cases, LARS required excessively longer time to reach the same accuracy, in particular when the sensing matrix was the partial Fourier one and fast schemes to perform matrix vector products can be exploited. For such matrices, the thresholding schemes (IHT, IST, TST) exhibited a performance that scales very well to large-size problems.

Figure 10.5b indicates the phase transition curve for one of the algorithms (IST) as we change the scenarios for generating the sparse (target) vectors, using different distributions: (a) ± 1 , with equiprobable selection of signs (constant amplitude random selection (CARS)); (b) double exponential (power); (c) Cauchy; and (d) uniform in $[-1, 1]$. This is indicative and typical for other algorithms as well, with some of them being more sensitive than others. Finally, Figure 10.5c shows the transition curves for the IST algorithm by changing the sensing matrix generation scenario. Three curves are shown corresponding to (a) uniform spherical ensemble (USE); (b) random sign ensemble (RSE), where the elements are ± 1 with signs uniformly distributed; and (c) the uniform random projection (URP) ensemble. Once more, one can observe the possible variations that are expected due to the use of different matrix ensembles. Moreover, changing ensembles affects each algorithm in a different way.

Concluding this section, it must be emphasized that the field of algorithmic development is still an ongoing research field, and it is early to come up with definite and concrete comparative performance conclusions. Moreover, besides the algorithmic front, existing theories often fall short in predicting what is observed in practice, with respect to their phase transition performance. For a related discussion, see, for example, [43].

10.3 VARIATIONS ON THE SPARSITY-AWARE THEME

In our tour so far, we have touched a number of aspects of sparsity-aware learning that come from mainstream theoretical developments. However, a number of variants have appeared and have been developed with the goal of addressing problems of a more special structure and/or proposing alternatives, which can be beneficial in boosting the performance in practice by serving the needs of specific applications. These variants focus either on the regularization term in (9.6), on the misfit-measuring term or, both. Once more, research activity in this direction is dense, and our purpose is to simply highlight possible alternatives and make the reader aware of the various possibilities that spring from the basic theory.

In a number of tasks, it is a priori known that the nonzero coefficients in the target signal/vector occur in groups and they are not randomly spread in all possible positions. A typical example is the echo path in internet telephony, where the nonzero coefficients of the impulse response tend to cluster together; see Figure 9.5. Other examples of “structured” sparsity can be traced in DNA microarrays, MIMO channel equalization, source localization in sensor networks, magnetoencephalography, or in neuroscience problems (e.g., [1, 9, 10, 60, 101]). As is always the case in machine learning, being able to incorporate a priori information into the optimization can only be of benefit for improving performance, because the estimation task is externally assisted in its effort to search for the target solution.

The *group LASSO* [8, 59, 97, 98, 117, 122] addresses the task where it is a priori known that the nonzero components occur in groups. The unknown vector θ is divided into L groups, that is,

$$\theta^T = [\theta_1^T, \dots, \theta_L^T]^T,$$

each of them of a predetermined size, s_i , $i = 1, 2, \dots, L$, with $\sum_{i=1}^L s_i = l$. The regression model can then be written as

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\eta} = \sum_{i=1}^L \mathbf{X}_i \boldsymbol{\theta}_i + \boldsymbol{\eta},$$

where each \mathbf{X}_i is a submatrix of \mathbf{X} comprising the corresponding s_i columns. The solution of the group LASSO is given by the following LS regularized task:

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \left(\left\| \mathbf{y} - \sum_{i=1}^L \mathbf{X}_i \boldsymbol{\theta}_i \right\|_2^2 + \lambda \sum_{i=1}^L \sqrt{s_i} \|\boldsymbol{\theta}_i\|_2 \right), \quad (10.15)$$

where $\|\boldsymbol{\theta}_i\|_2$ is the Euclidean norm (not the squared one) of $\boldsymbol{\theta}_i$, that is,

$$\|\boldsymbol{\theta}_i\|_2 = \sqrt{\sum_{j=1}^{s_i} |\theta_{i,j}|^2}.$$

In other words, the individual components of $\boldsymbol{\theta}$, which contribute to the formation of the ℓ_1 norm in the standard LASSO formulation, are now replaced by the square root of the energy of each individual block. In this setting, it is not the individual components but *blocks* of them that are forced to zero, when their contribution to the LS misfit measuring term is not significant. Sometimes, this type of regularization is coined as the ℓ_1/ℓ_2 regularization. An example of an ℓ_1/ℓ_2 ball for $\boldsymbol{\theta} \in \mathbb{R}^3$ can be seen in [Figure 10.6b](#) in comparison with the corresponding ℓ_1 ball depicted in [Figure 10.6a](#).

Beyond the conventional group LASSO, often referred to as *block sparsity*, research effort has been dedicated to the development of learning strategies incorporating more elaborate *structured sparse* models. There are two major reasons for such directions. First, in a number of applications the unknown set of parameters, $\boldsymbol{\theta}$, exhibit a structure that cannot be captured by the block sparse model. Second, even for cases where $\boldsymbol{\theta}$ is block sparse, standard grouped ℓ_1 norms require information about the partitioning of $\boldsymbol{\theta}$. This can be rather restrictive in practice. The adoption of overlapping groups has been proposed as a possible solution. Assuming that every coefficient belongs to at least one group, such models lead to optimization tasks that, in many cases, are not hard to solve, for example, by resorting to proximal methods [6, 7]. Moreover, by using properly defined overlapping groups [71], the allowed sparsity patterns can be constrained to form *hierarchical* structures, such as connected and rooted trees and subtrees that are met for example, in, multiscale (wavelet) decompositions. In [Figure 10.6c](#), an example of an ℓ_1/ℓ_2 ball for overlapping groups is shown.

Besides the previously stated directions, extensions of the compressed sensing principles to cope with structured sparsity led to the *model-based* compressed sensing [10, 26]. The (k, C) model allows the significant coefficients of a k -sparse signal to appear in at most C clusters, whose size is unknown. In [Section 9.9](#), it was commented that searching for a k -sparse solution takes place in a union of subspaces, each one of dimensionality k . Imposing a certain structure on the target solution restricts the searching in a subset of these subspaces and leaves a number of these out of the game. This obviously facilitates the optimization task. In [27], structured sparsity is considered in terms of graphical models, and in [110] the C-HiLasso group sparsity model was introduced, which allows each block to have a sparse structure itself. Theoretical results that extend the RIP to the block RIP have been developed and reported, see,

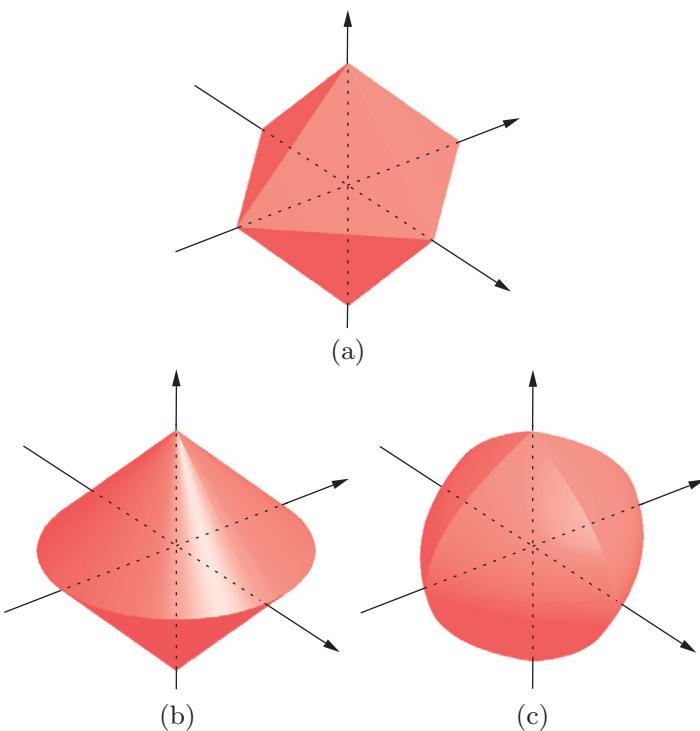


FIGURE 10.6

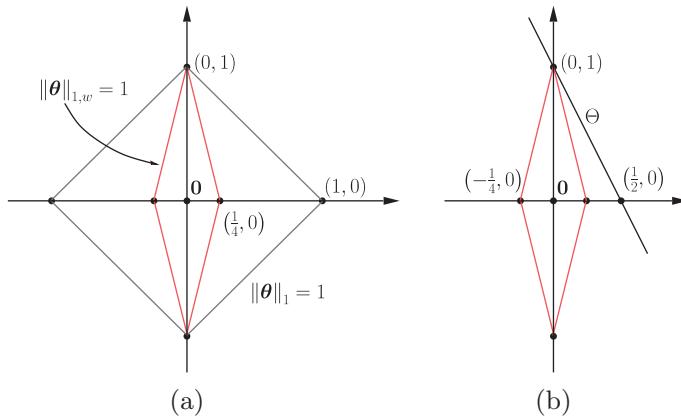
Representation of balls of $\theta \in \mathbb{R}^3$ corresponding to: (a) the ℓ_1 norm; (b) an ℓ_1/ℓ_2 with nonoverlapping groups; one group comprises $\{\theta_1, \theta_2\}$, and the other one $\{\theta_3\}$; (c) the ℓ_1/ℓ_2 with overlapping groups comprising $\{\theta_1, \theta_2, \theta_3\}$, $\{\theta_1\}$, and $\{\theta_3\}$, [6].

for example, [18, 83] and in the algorithmic front, proper modifications of greedy algorithms have been proposed in order to provide structured sparse solutions [53].

In [24], it is suggested to replace the ℓ_1 norm by a weighted version of it. To justify such a choice, let us recall Example 9.2 and the case where the “unknown” system was sensed using $x = [2, 1]^T$. We have seen that by “blowing” up the ℓ_1 ball, the wrong sparse solution was obtained. Let us now replace the ℓ_1 norm in (9.21) with its weighted version

$$\|\theta\|_{1,w} := w_1|\theta_1| + w_2|\theta_2|, \quad w_1, w_2 > 0,$$

and set $w_1 = 4$ and $w_2 = 1$. Figure 10.7a shows the isovalue curve $\|\theta\|_{1,w} = 1$, together with that resulting from the standard ℓ_1 norm. The weighted one is sharply “pinched” around the vertical axis, and the larger the value of w_1 , compared to that of w_2 , the sharper the corresponding ball will be. Figure 10.7b shows what happens when “blowing” the weighted ℓ_1 ball. It will first touch the point $(0, 1)$, which is the true solution. Basically, what we have done is “squeeze” the ℓ_1 ball to be aligned more to the axis that contains the (sparse) solution. For the case of our example, any weight $w_1 > 2$ would do the job.

**FIGURE 10.7**

(a) The isovalue curves for the ℓ_1 and the weighted ℓ_1 norms for the same value. The weighted ℓ_1 is sharply pinched around one of the axes, depending on the weights. (b) Adopting to minimize the weighted ℓ_1 norm for the setup of Figure 9.9c, the correct sparse solution is obtained.

Consider now the general case of a weighted norm

$$\|\theta\|_{1,w} := \sum_{j=1}^l w_j |\theta_j|, \quad w_j > 0, : \quad \text{Weighted } \ell_1 \text{ Norm.} \quad (10.16)$$

The ideal choice of the weights would be

$$w_j = \begin{cases} \frac{1}{|\theta_{o,j}|}, & \theta_{o,j} \neq 0, \\ \infty, & \theta_{o,j} = 0, \end{cases}$$

where θ_o is the target true vector, and where we have silently assumed that $0 \cdot \infty = 0$. In other words, the smaller a coefficient, the larger the respective weight becomes. This is justified, because large weighting will force respective coefficients toward zero during the minimization process. Of course, in practice the values of the true vector are not known, so it is suggested to use their estimates during each iteration of the minimization procedure. The resulting scheme is of the following form.

Algorithm 10.3.

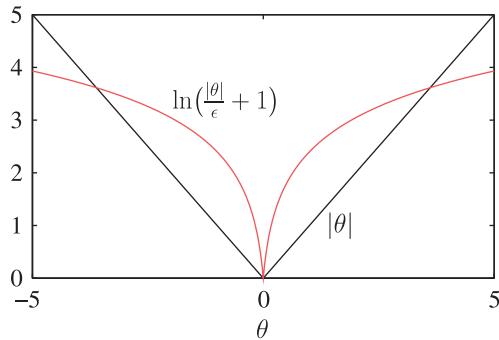
1. Initialize weights to unity, $w_j^{(0)} = 1, j = 1, 2, \dots, l$.
2. Minimize the weighted ℓ_1 norm,

$$\begin{aligned} \theta^{(i)} &= \arg \min_{\theta \in \mathbb{R}^l} \|\theta\|_{1,w} \\ \text{s.t.} \quad \mathbf{y} &= X\theta. \end{aligned}$$

3. Update the weights

$$w_j^{(i+1)} = \frac{1}{|\theta_j^{(i)}| + \epsilon}, \quad j = 1, 2, \dots, l.$$

4. Terminate when a stopping criterion is met, otherwise return to step 2.

**FIGURE 10.8**

One-dimensional graphs of the ℓ_1 norm and the logarithmic regularizer $\ln\left(\frac{|\theta|}{\epsilon} + 1\right) = \ln(|\theta| + \epsilon) - \ln\epsilon$, with $\epsilon = 0.1$. The term $\ln\epsilon$ was subtracted for illustration purposes only and does not affect the optimization. Notice the nonconvex nature of the logarithmic regularizer.

The constant ϵ is a small user-defined parameter to guarantee stability when the estimates of the coefficients take very small values. Note that if the weights have constant preselected values, the task retains its convex nature; this is no longer true when the weights are changing. It is interesting to point out that this intuitively motivated weighting scheme can result if the ℓ_1 norm is replaced by $\sum_{j=1}^l \ln(|\theta_j| + \epsilon)$ as the regularizing term of (9.6). Figure 10.8 shows the respective graph, in the one-dimensional space together with that of the ℓ_1 norm. The graph of the logarithmic function reminds us of the ℓ_p , $p < 0 < 1$ “norms” and the comments made in Section 9.2. This is no longer a convex function, and the iterative scheme given before is the result of a majorization-minimization procedure in order to solve the resulting nonconvex task [24] (Problem 10.6).

The concept of iterative weighting, as used before, has also been applied in the context of the *iterative reweighted LS algorithm*. Observe that the ℓ_1 norm can be written as

$$\|\boldsymbol{\theta}\|_1 = \sum_{j=1}^l |\theta_j| = \boldsymbol{\theta}^T \mathcal{W}_{\boldsymbol{\theta}} \boldsymbol{\theta},$$

where

$$\mathcal{W}_{\boldsymbol{\theta}} = \begin{bmatrix} \frac{1}{|\theta_1|} & 0 & \cdots & 0 \\ 0 & \frac{1}{|\theta_2|} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{|\theta_l|} \end{bmatrix},$$

and where in the case of $\theta_i = 0$, for some $i \in \{1, 2, \dots, l\}$, the respective coefficient of $\mathcal{W}_{\boldsymbol{\theta}}$ is defined to be 1. If $\mathcal{W}_{\boldsymbol{\theta}}$ were a constant weighting matrix, that is, $\mathcal{W}_{\boldsymbol{\theta}} := \mathcal{W}_{\tilde{\boldsymbol{\theta}}}$, for some fixed $\tilde{\boldsymbol{\theta}}$, then obtaining the minimum

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \|\mathbf{y} - X\boldsymbol{\theta}\|_2^2 + \lambda \boldsymbol{\theta}^T \mathcal{W}_{\tilde{\boldsymbol{\theta}}} \boldsymbol{\theta},$$

is straightforward and similar to the ridge regression. In the iterative reweighted scheme, \mathcal{W}_θ is replaced by $\mathcal{W}_{\tilde{\theta}^{(i)}}$, formed by using the respected estimates of the coefficients, which have been obtained from the previous iteration, that is, $\tilde{\theta} := \theta^{(i)}$, as we did before. In the sequel, each iteration solves a weighted ridge regression task.

The *focal underdetermined system solver* (FOCUSS) algorithm, [64], was the first one to use the concept of iterative-reweighted-least-squares (IRLS) to represent ℓ_p , $p \leq 1$ as a weighted ℓ_2 norm in order to find a sparse solution to an underdetermined system of equations. This algorithm is also of historical importance, because it is among the very first ones to emphasize the importance of sparsity; moreover, it provides comprehensive convergence analysis as well as a characterization of the stationary points of the algorithm. Variants of this basic iterative weighting scheme have also been proposed (see, e.g., [35] and the references therein).

In [126], the *elastic net* regularization penalty was introduced, which combines the ℓ_2 and ℓ_1 concepts together in a trade-off fashion, that is,

$$\lambda \sum_{i=1}^l (\alpha \theta_i^2 + (1 - \alpha) |\theta_i|),$$

where α is a user-defined parameter controlling the influence of each individual term. The idea behind the elastic net is to combine the advantages of the LASSO and the ridge regression. In problems where there is a group of variables in x that are highly correlated, LASSO tends to select one of the corresponding coefficients in θ , and set the rest to zero in a rather arbitrary fashion. This can be understood by looking carefully at how the greedy algorithms work. When sparsity is used to select the most important of the variables in x (feature selection), it is better to select all the relevant components in the group. If one knew which of the variables are correlated, he/she could form a group and then use the group LASSO. However, if this is not known, involving the ridge regression offers a remedy to the problem. This is because the ℓ_2 penalty in ridge regression tends to shrink the coefficients associated with correlated variables toward each other (e.g., [68]). In such cases, it would be better to work with the elastic net rationale that involves LASSO and ridge regression in a combined fashion.

In [23], the LASSO task is modified by replacing the squared error term with one involving correlations, and the minimization task becomes

$$\begin{aligned} \hat{\theta} : \quad & \min_{\theta \in \mathbb{R}^l} \|\theta\|_1 \\ \text{s.t.} \quad & \|X^T(y - X\theta)\|_\infty \leq \epsilon, \end{aligned}$$

where ϵ is related to l and the noise variance. This task is known as the *Dantzig selector*. That is, instead of constraining the energy of the error, the constraint now imposes an upper limit to the correlation of the error vector with any of the columns of X . In [5, 15], it is shown that under certain conditions, the LASSO estimator and the Dantzig selector become identical.

Total variation (TV) [107] is a closely related to ℓ_1 sparsity-promoting notion that has been widely used in image processing. Most of the grayscale image arrays, $I \in \mathbb{R}^{l \times l}$, consist of slowly varying pixel intensities except at the edges. As a consequence, the discrete gradient of an image array will be approximately sparse (compressible). The discrete directional derivatives of an image array are defined pixel-wise as

$$\nabla_x(I)(i,j) := I(i+1,j) - I(i,j), \quad \forall i \in \{1, 2, \dots, l-1\}, \quad (10.17)$$

$$\nabla_y(I)(i,j) := I(i,j+1) - I(i,j), \quad \forall j \in \{1, 2, \dots, l-1\}, \quad (10.18)$$

and

$$\nabla_x(I)(l,j) := \nabla_y(I)(i,l) := 0, \quad \forall i, j \in \{1, 2, \dots, l-1\}. \quad (10.19)$$

The discrete gradient transform

$$\nabla : \mathbb{R}^{l \times l} \rightarrow \mathbb{R}^{l \times 2l},$$

is defined in terms of a matrix form as

$$\nabla(I)(i,j) := [\nabla_x(I)(i,j), \nabla_y(I)(i,j)], \quad \forall i, j \in \{1, 2, \dots, l\}. \quad (10.20)$$

The total variation of the image array is defined as the ℓ_1 norm of the *magnitudes* of the elements of the discrete gradient transform, that is,

$$\|I\|_{TV} := \sum_{i=1}^l \sum_{j=1}^l \|\nabla(I)(i,j)\|_2 = \sum_{i=1}^l \sum_{j=1}^l \sqrt{\nabla_x(I)^2(i,j) + \nabla_y(I)^2(i,j)}. \quad (10.21)$$

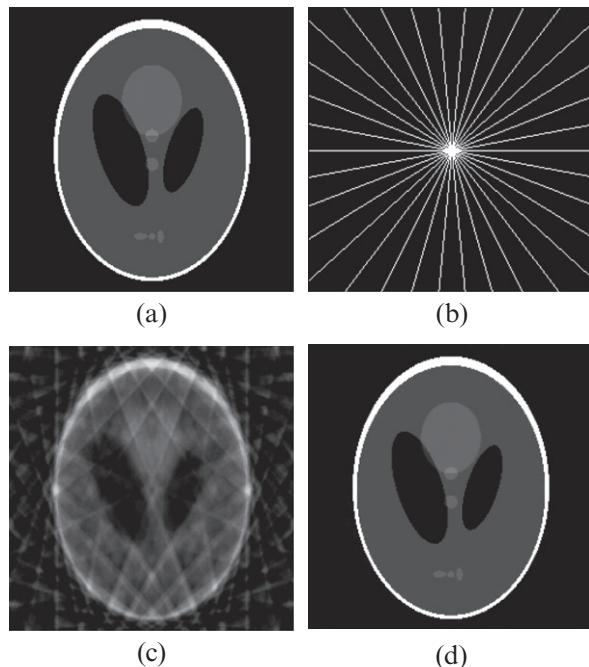
Note that this is a mixture of ℓ_2 and ℓ_1 norms. The sparsity promoting optimization around the total variation is defined as

$$\begin{aligned} I_* &\in \arg \min_I \|I\|_{TV} \\ \text{s.t.} \quad &\|y - \mathcal{F}(I)\|_2 \leq \epsilon, \end{aligned} \quad (10.22)$$

where $y \in \mathbb{R}^N$ is the observations vector and $\mathcal{F}(I)$ denotes the result in vectorized form of the application of a linear operator on I . For example, this could be the result of the action of a partial two-dimensional DFT on the image. Subsampling of the DFT matrix as a means of forming sensing matrices has already been discussed in [Section 9.7.2](#). The task in (10.22) retains its convex nature and it basically expresses our desire to reconstruct an image, that is as smooth as possible, given the available observations. The NESTA algorithm can be used for solving the total variation minimization task; besides it, other efficient algorithms for this task can be found in, for example, [63, 120].

It has been shown in [22] for the exact measurements case ($\epsilon = 0$), and in [94] for the erroneous measurements case, that conditions and bounds that guarantee recovery of an image array from the task in (10.22) can be derived and are very similar to those we have discussed for the case of the ℓ_1 norm.

Example 10.1 (*Magnetic resonance imaging (MRI)*). In contrast to ordinary imaging systems, which directly acquire pixel samples, MRI scanners sense the image in an encoded form. Specifically, MRI scanners sample components in the spatial frequency domain, known as “*k*-space” in MRI nomenclature. If all the components in this transform domain were available, one could apply the inverse 2D-DFT to recover the exact MR image in the pixel domain. Sampling in the *k*-space is realized along particular trajectories in a number of successive acquisitions. This process is time consuming, merely due to physical constraints. As a result, techniques for efficient image recovery from a *limited number of observations* is of high importance, because they can reduce the required acquisition time for performing

**FIGURE 10.9**

(a) The original Shepp-Logan image phantom. (b) The white lines indicate the directions across which the sampling in the spatial Fourier transform were obtained. (c) The recovered image after applying the inverse DFT, having first filled with zeros the missing values in the DFT transform. (d) The recovered image using the total variation minimization approach.

the measurements. Long acquisition times are not only inconvenient but even impossible, because the patients have to stay still for long time intervals. Thus, MRI was among the very first applications where compressed sensing found its way to offering its elegant solutions.

[Figure 10.9a](#) shows the “famous” Shepp-Logan phantom, and the goal is to recover it via a limited number of (measurements) samples in its frequency domain. The MRI measurements are taken across 17 radial lines in the spatial frequency domain, as shown in [Figure 10.9b](#). A “naive” approach to recovering the image from this limited number of measuring samples would be to adopt a zero-filling rationale for the missing components. The recovered image according to this technique is shown in [Figure 10.9c](#). [Figure 10.9d](#) shows the recovered image using the approach of minimizing the total variation, as explained before. Observe that the results for this case are astonishingly good. The original image is almost perfectly recovered. The constrained minimization was performed via the NESTA algorithm. Note that if the minimization of the ℓ_1 norm of the image array were used in place of the total variation, the results would not be as good; the phantom image is sparse in the discrete gradient domain, because it contains large sections that share constant intensities.

10.4 ONLINE SPARSITY-PROMOTING ALGORITHMS

In this section, online schemes for sparsity-aware learning are presented. There are a number of reasons that one has to resort to such schemes. As has already been noted in previous chapters, in various signal processing tasks the data arrive sequentially. Under such a scenario, using batch processing techniques to obtain an estimate of an unknown target parameter vector would be highly inefficient, because the number of training points keeps increasing. Such an approach is prohibited for real-time applications. Moreover, time-recursive schemes can easily incorporate the notion of adaptivity, when the learning environment is not stationary but undergoes changes as time evolves. Besides signal processing applications, there are an increasing number of machine learning applications where online processing is of paramount importance, such as bioinformatics, hyperspectral imaging, and data mining. In such applications, the number of training points easily amounts to a few thousand up to hundreds of thousand points. Concerning the dimensionality of the ambient (feature) space, one can claim numbers that lie in similar ranges. For example, in [82], the task is to search for sparse solutions in feature spaces with dimensionality as high as 10^9 having access to data sets as large as 10^7 points. Using batch techniques, in a single computer is out of the question with today's technology.

The setting that we have adopted for this section is the same as that used in previous chapters (e.g., [Chapters 5](#) and [6](#)). We assume that there is an unknown parameter vector that generates data according to the standard regression model

$$y_n = \mathbf{x}_n^T \boldsymbol{\theta} + \eta_n, \quad \forall n,$$

and the training samples are received sequentially (y_n, \mathbf{x}_n) , $n = 1, 2, \dots$. In the case of a stationary environment, we would expect our algorithm to converge asymptotically, as $n \rightarrow \infty$, to or “near to” the true parameter vector that gives birth to the observations, y_n , when it is sensed by \mathbf{x}_n . For time varying environments, the algorithms should be able to track the underlying changes as time goes by. Before we proceed, a comment is important. Because the time index, n , is left to grow, all we have said in the previous sections with respect to underdetermined systems of equations, loses its meaning. Sooner or later we are going to have more observations than the dimension of the space in which the data live. Our major concern here becomes the issue of asymptotic convergence for the case of stationary environments. The obvious question that is now raised is why not use a standard algorithm (e.g., LMS, RLS, or APSM), because we know that these algorithms converge to, or near enough in some sense, the solution (i.e., the algorithm will identify the zeros asymptotically)? The answer is that if such algorithms are modified to be aware of the underlying sparsity, convergence is significantly speeded up; in real-life applications, one does not have the “luxury” of waiting a long time for the solution. In practice, a good algorithm should be able to provide a good enough solution, and in the case of sparse solutions to *obtain the support*, after a reasonably small number of iteration steps.

In [Chapter 5](#), we commented on attempts to modify classical online schemes (for example, the proportionate LMS) in order to consider sparsity. However, these algorithms were of a rather ad hoc nature. In this section, the powerful theory around the ℓ_1 norm regularization will be used to obtain sparsity-promoting time adaptive schemes.

10.4.1 LASSO: ASYMPTOTIC PERFORMANCE

When presenting the basic principles of parameter estimation in [Chapter 3](#), the notions of bias, variance, and consistency, which are related to the performance of an estimator, were introduced. In a number

of cases, such performance measures were derived asymptotically. For example, we have seen that the maximum likelihood estimator is asymptotically unbiased and consistent. In Chapter 6, we saw that the LS estimator is also asymptotically consistent. Moreover, under the assumption that the noise samples are i.i.d., the LS estimator, $\hat{\theta}_n$, based on n observations, is itself a random vector that satisfies the \sqrt{n} -estimation consistency, that is,

$$\sqrt{n}(\hat{\theta}_n - \theta_o) \xrightarrow{d} \mathcal{N}(\mathbf{0}, \sigma^2 \Sigma^{-1}),$$

where θ_o is the true vector that generates the observations, Σ denotes the variance of the noise source, and Σ is the covariance matrix $\mathbb{E}[\mathbf{x}\mathbf{x}^T]$ of the input sequence, which has been assumed to be zero mean and the limit denotes convergence in distribution.

The LASSO in its (9.6) formulation is the task of minimizing the ℓ_1 norm regularized version of the LS cost. However, nothing has been said so far about the statistical properties of this estimator. The only performance measure that we referred to was the error norm bound given in (9.36). However, this bound, although important in the context for which it was proposed, does not provide much statistical information. Since the introduction of the LASSO estimator, a number of papers have addressed problems related to its statistical performance (see, e.g., [45, 55, 74, 127]).

When dealing with sparsity-promoting estimators such as the LASSO, two crucial issues emerge: (a) whether the estimator, even asymptotically, can obtain the support, if the true vector parameter is a sparse one; and (b) to quantify the performance of the estimator with respect to the estimates of the nonzero coefficients, that is, those coefficients whose index belongs to the support. Especially for LASSO, the latter issue becomes to study whether LASSO behaves as well as the unregularized LS with respect to these nonzero components. This task was addressed for the first time and in a more general setting in [55]. Let the support of the true, yet unknown, k -sparse parameter vector θ_o be denoted as S . Let also $\Sigma|_S$ be the $k \times k$ covariance matrix $\mathbb{E}[\mathbf{x}|_S \mathbf{x}|_S^T]$, where $\mathbf{x}|_S \in \mathbb{R}^k$ is the random vector that contains only the k components of \mathbf{x} , with indices in the support S . Then, we say that an estimator satisfies asymptotically the *oracle properties* if:

- $\lim_{n \rightarrow \infty} \text{Prob} \left\{ S_{\hat{\theta}_n} = S \right\} = 1$. This is known as *support consistency*.
- $\sqrt{n}(\hat{\theta}_{n|S} - \theta_{o|S}) \xrightarrow{d} \mathcal{N}(\mathbf{0}, \sigma^2 \Sigma_{|S}^{-1})$. This is the \sqrt{n} -estimation consistency.

We denote as $\theta_{o|S}$ and $\hat{\theta}_{n|S}$ the k -dimensional vectors that result from θ_o , $\hat{\theta}_n$, respectively, if we keep the components whose indices lie in the support S . In other words, according to the oracle properties, a good sparsity-promoting estimator should be able to predict, asymptotically, the true support and its performance with respect to the nonzero components should be as good as that of a genie-aided LS estimator, which is informed in advance of the positions of the nonzero coefficients.

Unfortunately, the LASSO estimator *cannot* satisfy simultaneously both conditions. It has been shown [55, 74, 127] that

- For support consistency, the regularization parameter $\lambda := \lambda_n$ should be time varying such that

$$\lim_{n \rightarrow \infty} \frac{\lambda_n}{\sqrt{n}} = \infty, \quad \lim_{n \rightarrow \infty} \frac{\lambda_n}{n} = 0.$$

That is, λ_n must grow faster than \sqrt{n} , but slower than n .

- For \sqrt{n} -consistency, λ_n must grow as

$$\lim_{n \rightarrow \infty} \frac{\lambda_n}{\sqrt{n}} = 0,$$

that is, it grows slower than \sqrt{n} .

The previous two conditions are conflicting and the LASSO estimator cannot comply with the two oracle conditions simultaneously. The proofs of the previous two points are somewhat technical and are not given here. The interested reader can obtain them from the previously given references. However, before we proceed, it is instructive to see why the regularization parameter has to grow more slowly than n , in any case. Without being too rigorous mathematically, recall that the LASSO solution comes from Eq. (9.6). This can be written as

$$\mathbf{0} \in -\frac{2}{n} \sum_{i=1}^n \mathbf{x}_i y_i + \frac{2}{n} \left(\sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T \right) \boldsymbol{\theta} + \frac{\lambda_n}{n} \partial \|\boldsymbol{\theta}\|_1, \quad (10.23)$$

where we have divided both sides by n . Taking the limit as $n \rightarrow \infty$, if $\lambda_n/n \rightarrow 0$, then we are left with the first two terms; this is exactly what we would have if the unregularized LS had been chosen as the cost function. Recall from Chapter 6 that in this case, the solution asymptotically converges¹ (under some general assumptions, which are assumed to hold true here) to the true parameter vector; that is, we have strong consistency.

10.4.2 THE ADAPTIVE NORM-WEIGHTED LASSO

There are two ways to get out of the previously stated conflict. One is to replace the ℓ_1 norm with a nonconvex function, which can lead to an estimator that satisfies the oracle properties simultaneously [55]. The other is to modify the ℓ_1 norm by replacing it with a *weighted* version. Recall that the weighted ℓ_1 norm was discussed in Section 10.3 as a means to assist the optimization procedure to unveil the sparse solution. Here the notion of weighted ℓ_1 norm comes as a necessity imposed by our willingness to satisfy the oracle properties. This gives rise to the *adaptive time-and-norm-weighted LASSO* (TNWL) cost estimate defined as

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^l} \left\{ \sum_{j=1}^n \beta^{n-j} (y_j - \mathbf{x}_j^T \boldsymbol{\theta})^2 + \lambda_n \sum_{i=1}^l w_{i,n} |\theta_i| \right\}, \quad (10.24)$$

where $\beta \leq 1$ is used as the forgetting factor to allow for tracking slow variations. The time varying weighting sequences is denoted as $w_{i,n}$. There are different options. In [127] and under a stationary environment with $\beta = 1$, it is shown that if

$$w_{i,n} = \frac{1}{|\theta_i^{\text{est}}|^\gamma},$$

where θ_i^{est} is the estimate of the i th component obtained by *any* \sqrt{n} -consistent estimator, such as the unregularized LS, then for specific choices of λ_n and γ the corresponding estimator satisfies the oracle

¹ Recall that this convergence is with probability 1.

properties simultaneously. The main reasoning behind the weighted ℓ_1 norm is that as time goes by, and the \sqrt{n} -consistent estimator provides better and better estimates, then the weights corresponding to indices outside the true support (zero values) are inflated and those corresponding to the true support converge to a finite value. This helps the algorithm, simultaneously to locate the support and obtain unbiased (asymptotically) estimates of the large coefficients.

Another choice for the weighting sequence is related to the *smoothly clipped absolute deviation* (SCAD) [55, 128]. This is defined as

$$w_{i,n} = \chi_{(0,\mu_n)}(|\theta_i^{\text{est}}|) + \frac{(\alpha\mu_n - |\theta_i^{\text{est}}|)_+}{(\alpha - 1)\mu_n} \chi_{(\mu_n,\infty)}(|\theta_i^{\text{est}}|),$$

where $\chi(\cdot)$ stands for the characteristic function, $\mu_n = \lambda_n/n$, and $\alpha > 2$. Basically, this corresponds to a quadratic spline function. It turns out [128] that if λ_n is chosen to grow faster than \sqrt{n} and slower than n , the adaptive LASSO with $\beta = 1$ satisfies both oracle conditions simultaneously.

A time adaptive scheme for solving the TNWL LASSO was presented in [3]. The cost function of the adaptive LASSO in (10.24) can be written as

$$J(\boldsymbol{\theta}) = \boldsymbol{\theta}^T R_n \boldsymbol{\theta} - \mathbf{r}_n^T \boldsymbol{\theta} + \lambda_n \|\boldsymbol{\theta}\|_{1,w_n},$$

where

$$R_n := \sum_{j=1}^n \beta^{n-j} \mathbf{x}_j \mathbf{x}_j^T, \quad \mathbf{r}_n := \sum_{j=1}^n \beta^{n-j} y_j \mathbf{x}_j,$$

and $\|\boldsymbol{\theta}\|_{1,w_n}$ is the weighted ℓ_1 norm. We know from Chapter 6, and it is straightforward to see, that

$$R_n = \beta R_{n-1} + \mathbf{x}_n \mathbf{x}_n^T, \quad \mathbf{r}_n = \beta \mathbf{r}_{n-1} + y_n \mathbf{x}_n.$$

The complexity for both of the previous updates, for matrices of a general structure, amounts to $\mathcal{O}(l^2)$ multiply/add operations. One alternative is to update R_n and \mathbf{r}_n and then solve a convex optimization task for each time instant, n , using any standard algorithm. However, this is not appropriate for real-time applications, due to its excessive computational cost. In [3], a time-recursive version of a coordinate descent algorithm has been developed. As we saw in Section 10.2.2, coordinate descent algorithms update one component at each iteration step. In [3], iteration steps are associated with time updates, as is always the case with the online algorithms. As each new training pair (y_n, \mathbf{x}_n) is received, a single component of the unknown vector is updated. Hence, at each time instant, a scalar optimization task has to be solved, and its solution is given in closed form, which results in a simple soft thresholding operation. One of the drawbacks of the coordinate techniques is that each coefficient is updated every l time instants, which, for large values of l , can slow down convergence. Variants of the basic scheme that cope with this drawback are also addressed in [3], referred to as online cyclic coordinate descent time weighted LASSO (OCCD-TWL). The complexity of the scheme is of the order of $\mathcal{O}(l^2)$. Computational savings are possible if the input sequence is a time series and fast schemes for the updates of R_n and the RLS can then be exploited. However, if an RLS-type algorithm is used in parallel, the convergence of the overall scheme may be slowed down, because the RLS-type algorithm has to converge first in order to provide reliable estimates for the weights, as pointed out before.

10.4.3 ADAPTIVE CoSaMP (AdCoSaMP) ALGORITHM

In [90], an adaptive version of the CoSaMP algorithm, which is summarized in [Algorithm 10.2](#), was proposed. Iteration steps, i , now coincide with time updates, n , and the LS solver in Step 3c of the general CSMP scheme is replaced by an LMS one.

Let us focus first on the quantity $X^T e^{(i-1)}$ in Step 3a of the CSMP scheme, which is used to compute the support at iteration i . In the online setting and at (iteration) time n , this quantity is now “rephrased” as

$$X^T e_{n-1} = \sum_{j=1}^{n-1} x_j e_j.$$

In order to make the algorithm flexible to adapt to variations of the environment as the time index, n , increases, the previous correlation sum is modified to

$$\mathbf{p}_n := \sum_{j=1}^{n-1} \beta^{n-1-j} x_j e_j = \beta \mathbf{p}_{n-1} + \mathbf{x}_{n-1} e_{n-1}.$$

The LS task, constrained on the active columns that correspond to the indices in the support S in Step 3c, is performed in an online rationale by involving the basic LMS recursions, that is,²

$$\tilde{e}_n := y_n - \mathbf{x}_{n|S}^T \tilde{\theta}_{|S}(n-1)$$

$$\tilde{\theta}_{|S}(n) := \tilde{\theta}_{|S}(n-1) + \mu \mathbf{x}_{n|S} \tilde{e}_n,$$

where $\tilde{\theta}_{|S}(\cdot)$ and $\mathbf{x}_{n|S}$ denote the respective subvectors corresponding to the indices in the support S . The resulting algorithm is summarized as follows.

Algorithm 10.4 (The AdCoSaMP scheme).

1. Select the value of $t = 2k$.
2. Initialize the algorithm: $\theta(1) = \mathbf{0}$, $\tilde{\theta}(1) = \mathbf{0}$, $\mathbf{p}_1 = \mathbf{0}$, $e_1 = y_1$.
3. Choose μ and β .
4. For $n = 2, 3, \dots$, execute the following steps:
 - (a) $\mathbf{p}_n = \beta \mathbf{p}_{n-1} + \mathbf{x}_{n-1} e_{n-1}$.
 - (b) Obtain the current support:

$$S = \text{supp}\{\theta(n-1)\} \cup \left\{ \begin{array}{l} \text{indices of the } t \text{ largest in} \\ \text{magnitude components of } \mathbf{p}_n \end{array} \right\}.$$

- (c) Perform the LMS update:

$$\tilde{e}_n = y_n - \mathbf{x}_{n|S}^T \tilde{\theta}_{|S}(n-1),$$

$$\tilde{\theta}_{|S}(n) = \tilde{\theta}_{|S}(n-1) + \mu \mathbf{x}_{n|S} \tilde{e}_n.$$

- (d) Obtain the set S_k of the indices of the k largest components of $\tilde{\theta}_{|S}(n)$.
- (e) Obtain $\theta(n)$ such that:

$$\theta_{|S_k}(n) = \tilde{\theta}_{|S_k}, \quad \text{and } \theta_{|S_k^c}(n) = \mathbf{0},$$

where S_k^c is the complement set of S_k .

- (f) Update the error: $e_n = y_n - \mathbf{x}_n^T \theta(n)$.

² The time index for the parameter vector is given in parentheses, due to the presence of the other subscripts.

In place of the standard LMS, its normalized version can alternatively be adopted. Note that Step 4e is directly related to the hard thresholding operation.

In [90], it is shown that if the sensing matrix, which is now time dependent and keeps increasing in size, satisfies a condition similar to RIP for each time instant, called *exponentially weighted isometry property* (ERIP), which depends on β , then the algorithm asymptotically satisfies an error bound, which is similar to the one that has been derived for CoSaMP in [93], plus an extra term that is due to the excess mean-square error (see [Chapter 5](#)), which is the price paid by replacing the LS solver by the LMS.

10.4.4 SPARSE ADAPTIVE PROJECTION SUBGRADIENT METHOD (SpAPSM)

The APSM family of algorithms was introduced in [Chapter 8](#), as one among the most popular techniques for online/adaptive learning. As pointed out there, a major advantage of this algorithmic family is that one can readily incorporate convex constraints. In [Chapter 8](#), APSM was used as an alternative to methods that build around the LS loss function, such as the LMS and the RLS. The rationale behind APSM is that because our data are assumed to be generated by a regression model, then the unknown vector could be estimated by finding a point in the intersection of a sequence of hyperslabs that are defined by the data points, that is, $S_n[\epsilon] := \{\boldsymbol{\theta} \in \mathbb{R}^l : |y_n - \mathbf{x}_n^\top \boldsymbol{\theta}| \leq \epsilon\}$. Also, it was pointed out that such a model was very natural when the noise is bounded. When dealing with sparse vectors, there is an additional constraint that we want our solution to satisfy, that is, $\|\boldsymbol{\theta}\|_1 \leq \rho$ (see also the LASSO formulation [\(9.7\)](#)). This task fits nicely in the APSM rationale and the basic recursion can be readily written, without much thought or derivation, as follows: for any arbitrarily chosen initial point $\boldsymbol{\theta}_0$, define $\forall n$,

$$\boldsymbol{\theta}_n = P_{B_{\ell_1}[\delta]} \left(\boldsymbol{\theta}_{n-1} + \mu_n \left(\sum_{i=n-q+1}^n \omega_i^{(n)} P_{S_i[\epsilon]}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right) \right), \quad (10.25)$$

where $q \geq 1$ is the number of hyperslabs that are considered each time, μ_n is an extrapolation parameter, and it is a user-defined variable. In order for convergence to be guaranteed, theory dictates that it must lie in the interval $(0, 2\mathcal{M}_n)$, where

$$\mathcal{M}_n := \begin{cases} \frac{\sum_{i=n-q+1}^n \omega_i^{(n)} \|P_{S_i[\epsilon]}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1}\|^2}{\left\| \sum_{i=n-q+1}^n \omega_i^{(n)} P_{S_i[\epsilon]}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right\|^2}, \\ \text{if } \left\| \sum_{i=n-q+1}^n \omega_i^{(n)} P_{S_i[\epsilon]}(\boldsymbol{\theta}_{n-1}) - \boldsymbol{\theta}_{n-1} \right\| \neq 0, \\ 1, \text{ otherwise,} \end{cases} \quad (10.26)$$

and $P_{B_{\ell_1}[\rho]}(\cdot)$ is the projection operator onto the ℓ_1 ball $B_{\ell_1}[\rho] := \{\boldsymbol{\theta} \in \mathbb{R}^l : \|\boldsymbol{\theta}\|_1 \leq \rho\}$, because the solution is constrained to live within this ball. Note that recursion [\(10.25\)](#) is analogous to the iterative soft thresholding shrinkage algorithm in the batch processing case [\(10.7\)](#). There, we saw that the only difference the sparsity imposes on an iteration, with respect to its unconstrained counterpart, is an extra soft thresholding. This is exactly the case here. The term in the parentheses is the iteration for the unconstrained task. Moreover, as has been shown in [\[46\]](#), projection on the ℓ_1 ball is equivalent to a soft thresholding operation. Following the general arguments given in [Chapter 8](#), the previous iteration converges arbitrarily close to a point in the intersection

$$B_{\ell_1}[\delta] \cap \bigcap_{n \geq n_0} S_n[\epsilon],$$

for some finite value of n_0 . In [76, 77], the weighted ℓ_1 ball (denoted here as $B_{\ell_1[w_n, \rho]}$) has been used to improve convergence as well as the tracking speed of the algorithm, when the environment is time varying. The weights were adopted in accordance with what was discussed in Section 10.3, that is,

$$w_{i,n} := \frac{1}{|\theta_{i,n-1}| + \epsilon_n}, \quad \forall i \in \{1, 2, \dots, l\},$$

where $(\epsilon_n)_{n \geq 0}$ is a sequence (can be also constant) of small numbers to avoid division by zero. The basic time iteration becomes as follows: for any arbitrarily chosen initial point θ_0 , define $\forall n$,

$$\theta_n = P_{B_{\ell_1}[w_n, \rho]} \left(\theta_{n-1} + \mu_n \left(\sum_{i=n-q+1}^n \omega_i^{(n)} P_{S_i[\epsilon]}(\theta_{n-1}) - \theta_{n-1} \right) \right), \quad (10.27)$$

where $\mu_n \in (0, 2\mathcal{M}_n)$ and \mathcal{M}_n is given in (10.26). Figure 10.10 illustrates the associated geometry of the basic iteration in \mathbb{R}^2 and for the case of $q = 2$. It comprises two parallel projections on the

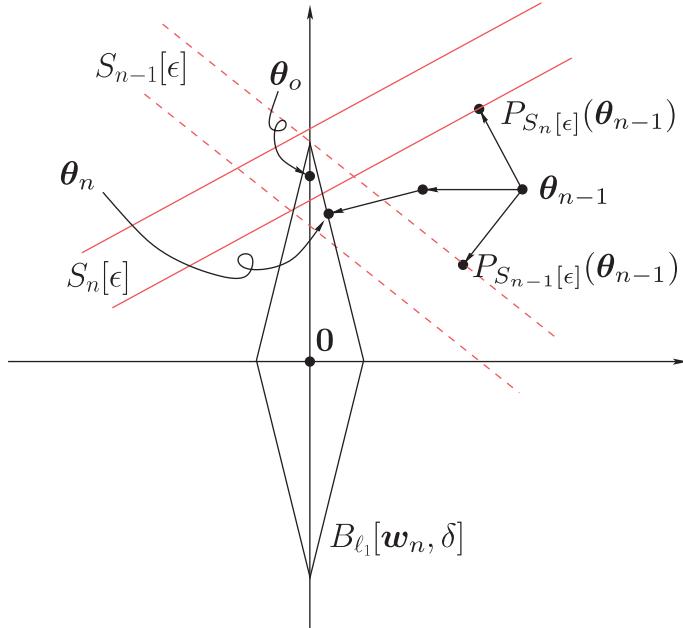


FIGURE 10.10

Geometric illustration of the update steps involved in the SpAPSM algorithm, for the case of $q = 2$. The update at time n is obtained by first convexly combining the projections onto the current and previously formed hyperslabs, $S_n[\epsilon]$, $S_{n-1}[\epsilon]$, and then projecting onto the weighted ℓ_1 ball. This brings the update closer to the target solution θ_o .

hyperslabs, followed by one projection onto the weighted ℓ_1 ball. In [76], it is shown (Problem 10.7) that a good bound for the weighted ℓ_1 norm is the sparsity level k of the target vector, which is assumed to be known and is a user-defined parameter. In [76], it is shown that asymptotically, and under some general assumptions, this algorithmic scheme converges arbitrarily close to the intersection of the hyperslabs with the weighted ℓ_1 balls, that is,

$$\bigcap_{n \geq n_0} \left(P_{B_{\ell_1}[\mathbf{w}_n, \rho]} \cap S_n[\epsilon] \right),$$

for some nonnegative integer n_0 . It has to be pointed out that in the case of weighted ℓ_1 norms, the constraint is *time varying* and the convergence analysis is not covered by the standard analysis used for APSM, and had to be extended to this more general case.

The complexity of the algorithm amounts to $\mathcal{O}(ql)$. The larger the q the faster the convergence rate, at the expense of higher complexity. In [77], in order to reduce the dependence of the complexity on q , the notion of the *subdimensional* projection is introduced, where projections onto the q hyperslabs could be restricted along the directions of the most significant coefficients of the currently available estimates. The dependence on q now becomes $\mathcal{O}(qk_n)$, where k_n is the sparsity level of the currently available estimate, which after a few steps of the algorithm gets much lower than l . The total complexity amounts to $\mathcal{O}(l) + \mathcal{O}(qk_n)$ per iteration step. This allows the use of large values of q , which (at only a small extra computational cost compared to $\mathcal{O}(l)$) drives the algorithm to a performance close to that of the adaptive weighted LASSO.

Projection onto the weighted ℓ_1 ball

Projecting onto an ℓ_1 ball is equivalent to a soft thresholding operation. Projection onto the weighted ℓ_1 norm results in a slight variation of the soft thresholding, with different threshold values per component. In the sequel, we give the iteration steps for the more general case of the weighted ℓ_1 ball. The proof is a bit technical and lengthy and it will not be given here. It was derived, for the first time, via purely geometric arguments, and without the use of the classical Lagrange multipliers, in [76]. Lagrange multipliers have been used instead in [46], for the case of the ℓ_1 ball. The efficient computation of the projection on the ℓ_1 ball was treated earlier, in a more general context, in [100].

Recall from the definition of a projection, discussed in Chapter 8, that given a point outside the ball, $\boldsymbol{\theta} \in \mathbb{R}^l \setminus B_{\ell_1}[\mathbf{w}, \rho]$, its projection onto the weighted ℓ_1 ball is the point $P_{B_{\ell_1}[\mathbf{w}, \rho]}(\boldsymbol{\theta}) \in B_{\ell_1}[\mathbf{w}, \rho] := \{\mathbf{z} \in \mathbb{R}^l : \sum_{i=1}^l w_i |z_i| \leq \rho\}$ that lies closest to $\boldsymbol{\theta}$ in the Euclidean sense. If $\boldsymbol{\theta}$ lies within the ball then it coincides with its projection. Given the weights and the value of ρ , the following iterations provide the projection.

Algorithm 10.5 (Projection onto the weighted ℓ_1 ball $B_{\ell_1}[\mathbf{w}, \rho]$).

1. Form the vector $[|\theta_1|/w_1, \dots, |\theta_l|/w_l]^T \in \mathbb{R}^l$.
2. Sort the previous vector in a nonascending order, so that $|\theta_{\tau(1)}|/w_{\tau(1)} \geq \dots \geq |\theta_{\tau(l)}|/w_{\tau(l)}$. The notation τ stands for the permutation, which is implicitly defined by the sorting operation. Keep in mind the inverse τ^{-1} , which is the index of the position of the element in the original vector.
3. $r_1 := l$.
4. Let $m = 1$. While $m \leq l$, do
 - (a) $m_* := m$.
 - (b) Find the maximum j_* among those $j \in \{1, 2, \dots, r_m\}$ such that $\frac{|\theta_{\tau(j)}|}{w_{\tau(j)}} > \frac{\sum_{i=1}^{r_m} w_{\tau(i)} |\theta_{\tau(i)}| - \rho}{\sum_{i=1}^{r_m} w_{\tau(i)}^2}$.

- (c) If $j_* = r_m$ then break the loop.
 - (d) Otherwise set $r_{m+1} := j_*$.
 - (e) Increase m by 1 and go back to Step 4a.
5. Form the vector $\hat{\mathbf{p}} \in \mathbb{R}^{r_{m_*}}$ whose j -th component, $j = 1, \dots, r_{m_*}$, is given by
- $$\hat{p}_j := |\theta_{\tau(j)}| - \frac{\sum_{i=1}^{r_{m_*}} w_{\tau(i)} |\theta_{\tau(i)}| - \rho}{\sum_{i=1}^{r_{m_*}} w_{\tau(i)}^2} w_{\tau(j)}.$$
6. Use the inverse mapping τ^{-1} to insert the element \hat{p}_j into the $\tau^{-1}(j)$ position of the l -dimensional vector \mathbf{p} , $\forall j \in \{1, 2, \dots, r_{m_*}\}$, and fill in the rest with zeros.
7. The desired projection is $P_{B_{\ell_1}[\mathbf{w}, \rho]}(\boldsymbol{\theta}) = [\text{sgn}(\theta_1)p_1, \dots, \text{sgn}(\theta_l)p_l]^T$.

Remarks 10.3.

- *Generalized Thresholding Rules:* Projections onto both ℓ_1 and weighted ℓ_1 balls impose convex sparsity inducing constraints via properly performed soft thresholding operations. More recent advances within the SpAPSM framework [78, 109] allow the substitution of $P_{B_{\ell_1}[\rho]}$ and $P_{B_{\ell_1}[\mathbf{w}, \rho]}$ with a *generalized thresholding*, built around the notions of SCAD, nonnegative garrote, and a number of thresholding functions corresponding to the *nonconvex*, ℓ_p , $p < 1$ penalties. Moreover, it is shown that such generalized thresholding (GT) operators are nonlinear mappings with *their fixed point set being a union of subspaces*, that is, the nonconvex object that lies at the heart of any sparsity-promoting technique. Such schemes are very useful for low values of q , where one can improve upon the performance obtained by the LMS-based AdCoSAMP, at comparable complexity levels.
- A comparative study of various online sparsity-promoting low complexity schemes, including the proportionate LMS, in the context of the echo cancelation task, is given in [80]. It turns out that the SpAPSM-based schemes outperform LMS-based sparsity-promoting algorithms.
- More algorithms and methods that involve sparsity-promoting regularization, in the context of more general convex loss functions, compared to the squared error, are discussed in [Chapter 8](#), where related references are provided.
- *Distributed Sparsity-Promoting Algorithms:* Besides the algorithms reported so far, a number of algorithms in the context of distributed learning have also appeared in the literature. As pointed out in [Chapter 5](#), algorithms complying to the consensus as well as the diffusion rationale have been proposed (see, e.g., [29, 39, 89, 102]). A review of such algorithms appears in [30].

Example 10.2 (Time-varying signal). In this example, the performance curves of the most popular online algorithms, mentioned before, are studied in the context of a time varying environment. A typical simulation setup, which is commonly adopted by the adaptive filtering community in order to study the tracking agility of an algorithm, is that of an unknown vector that undergoes an abrupt change after a number of observations. Here, we consider a signal, \mathbf{s} , with a sparse wavelet representation, that is, $\mathbf{s} = \Psi \boldsymbol{\theta}$, where Ψ is the corresponding transformation matrix. In particular, we set $l = 1024$ with 100 nonzero wavelet coefficients. After 1500 measurements (observations), 10 arbitrarily picked wavelet coefficients change their values to new ones, selected uniformly at random from the interval $[-1, 1]$. Note that this may affect the sparsity level of the signal, and we can now end with up to 110 nonzero coefficients. A total of $N = 3000$ sensing vectors are used, which result from the wavelet transform of the input vectors $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, 3000$, having elements drawn from $\mathcal{N}(0, 1)$. In this way,

the online algorithms do not estimate the signal itself, but its sparse wavelet representation, θ . The observations are corrupted by additive white Gaussian noise of variance $\sigma_n^2 = 0.1$. Regarding SpAPSM, the extrapolation parameter μ_n is set equal to $1.8 \times \mathcal{M}_n$, $\omega_i^{(n)}$ are all getting the same value $1/q$, the hyperslabs parameter ϵ was set equal to $1.3\sigma_n$, and $q = 390$. The parameters for all algorithms were selected in order to optimize their performance. Because the sparsity level of the signal may change (from $k = 100$ up to $k = 110$) and because in practice it is not possible to know in advance the exact value of k , we feed the algorithms with an overestimate, k , of the true sparsity value, and in particular we used $\hat{k} = 150$ (i.e., 50% overestimation up to the 1500th iteration).

The results are shown in Figure 10.11. Note the enhanced performance obtained via the SpAPSM algorithm. However, it has to be pointed out that the complexity of the AdCoSAMP is much lower compared to the other two algorithms, for the choice of $q = 390$ for the SpAPSM. The interesting observation is that SpAPSM achieves a better performance compared to OCCD-TWL, albeit at significantly lower complexity. If on the other hand complexity is of major concern, use of SpAPSM offers the flexibility to use generalized thresholding operators, which lead to improved performance for small values of q , at complexity comparable to that of LMS-based sparsity promoting algorithms [79, 80].

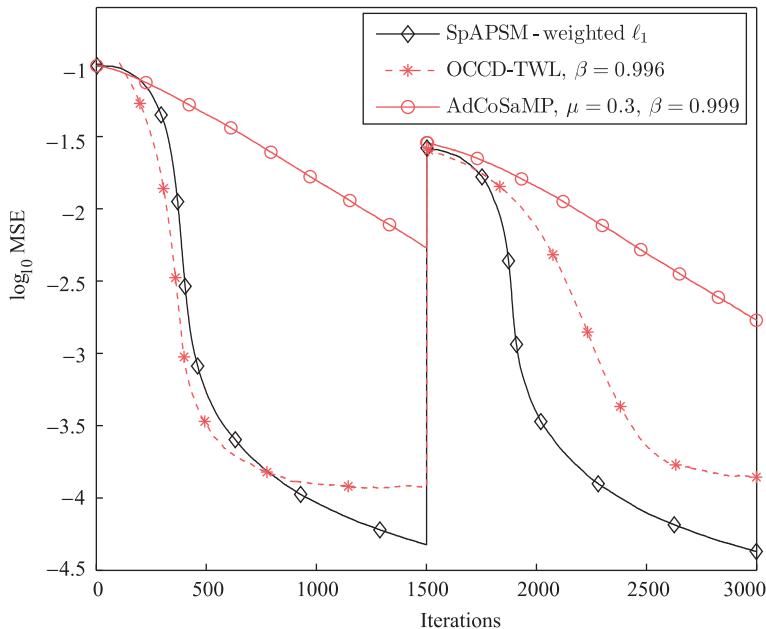


FIGURE 10.11

MSE learning curves for AdCoSAMP, SpAPSM, and OCCD-TWL for the simulation in Example 10.2. The vertical axis shows the \log_{10} of the mean-square, that is, $\log_{10} \frac{1}{2} \|s - \Psi\theta_n\|^2$, and the horizontal shows the time index. At time $n = 1500$, the system undergoes a sudden change.

10.5 LEARNING SPARSE ANALYSIS MODELS

All our discussion so far has been spent in the terrain of signals that are either sparse themselves or that can be sparsely represented in terms of the atoms of a dictionary in a synthesis model, as introduced in (9.16), that is,

$$\mathbf{s} = \sum_{i \in \mathcal{I}} \theta_i \boldsymbol{\psi}_i.$$

As a matter of fact, most of the research activity has been focused on the synthesis model. This may be partly due to the fact that the synthesis modeling path may provide a more intuitively appealing structure to describe the generation of the signal in terms of the elements (atoms) of a dictionary. Recall from Section 9.9 that the sparsity assumption was imposed on $\boldsymbol{\theta}$ in the synthesis model and the corresponding optimization task was formulated in (9.38) and (9.39) for the exact and noisy cases, respectively.

However, this is not the only way to approach the task of sparse modeling. Very early in this chapter, in Section 9.4, we referred to the analysis model

$$\mathbf{S} = \Phi^H \mathbf{s}$$

and pointed out that in a number of real-life applications, the resulting transform \mathbf{S} is sparse. To be fair, the most orthodox way to deal with the underlying model sparsity would be to consider $\|\Phi^H \mathbf{s}\|_0$. Thus, if one wants to estimate \mathbf{s} , a very natural way would be to cast the related optimization task as

$$\begin{aligned} \min_{\mathbf{s}} \quad & \|\Phi^H \mathbf{s}\|_0, \\ \text{s.t.} \quad & \mathbf{y} = \mathbf{Xs}, \text{ or } \|\mathbf{y} - \mathbf{Xs}\|_2^2 \leq \epsilon, \end{aligned} \tag{10.28}$$

depending on whether the measurements via a sensing matrix, \mathbf{X} , are exact or noisy. Strictly speaking, the total variation minimization approach, which was used in Example 10.1, falls under this analysis model formulation umbrella, because what is minimized is the ℓ_1 norm of the gradient transform of the image.

The optimization tasks in either of the two formulations given in (10.28) build around the assumption that the signal of interest has *sparse analysis representation*. The obvious question that is now raised is whether the optimization tasks in (10.28) and their counterparts in (9.38) or (9.39) are any different. One of the first efforts to shed light on this problem was in [51]. There, it was pointed out that the two tasks, though related, are in general different. Moreover, their comparative performance depends on the specific problem at hand. However, it is fair to say that this is a new field of research and more definite conclusions are currently being shaped. An easy answer can be obtained for the case where the involved dictionary corresponds to an orthonormal transformation matrix (e.g., DFT). In this case, we already know that the analysis and synthesis matrices are related as

$$\Phi = \Psi = \Psi^{-H},$$

which leads to an equivalence between the two previously stated formulations. Indeed, for such a transform we have

$$\underbrace{\mathbf{S} = \Phi^H \mathbf{s}}_{\text{Analysis}} \Leftrightarrow \underbrace{\mathbf{s} = \Phi \mathbf{S}}_{\text{Synthesis}}.$$

Using the last formula into the (10.28), the tasks in (9.38) or (9.39) are readily obtained by replacing θ by s . However, this reasoning cannot be extended to the case of overcomplete dictionaries; in these cases, the two optimization tasks may lead to different solutions.

The previous discussion concerning the comparative performance between the synthesis or analysis-based sparse representations is not only of “philosophical” value. It turns out that, often in practice, the nature of certain overcomplete dictionaries does not permit the use of the synthesis-based formulation. These are the cases where the columns of the overcomplete dictionary exhibit a high degree of dependence; that is, the coherence of the matrix, as defined in [Section 9.6.1](#), has large values. Typical examples of such overcomplete dictionaries are the Gabor frames, the curvelet frames, and the oversampled DFT. The use of such dictionaries lead to enhanced performance in a number of applications (e.g., [111, 112]). Take as an example the case of our familiar DFT transform. This transform provides a representation of our signal samples in terms of sampled exponential sinusoids, whose integral frequencies are multiples of $\frac{2\pi}{l}$, that is,

$$s := \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_{l-1} \end{bmatrix} = \sum_{i=0}^{l-1} S_i \psi_i, \quad (10.29)$$

where S_i are the DFT coefficients and ψ_i is the sampled sinusoid with frequency equal to $\frac{2\pi}{l}i$, that is,

$$\psi_i = \begin{bmatrix} 1 \\ \exp(-j\frac{2\pi}{l}i) \\ \vdots \\ \exp(-j\frac{2\pi}{l}i(l-1)) \end{bmatrix}. \quad (10.30)$$

However, this is not necessarily the most efficient representation. For example, it is highly unlikely that a signal comprises only integral frequencies and only such signals can result in a sparse representation using the DFT basis. Most probably, in general, there will be frequencies lying in between the frequency samples of the DFT basis that result in nonsparse representations. Using these extra frequencies, a much better representation of the frequency content of the signal can be obtained. However, in such a dictionary, the atoms are no longer linearly independent, and the coherence of the respective (dictionary) matrix increases.

Once a dictionary exhibits high coherence, then there is no way of finding a sensing matrix, X , so that $X\Psi$ obeys the RIP. Recall that at the heart of sparsity-aware learning lies the concept of stable embedding, that allows the recovery of a vector/signal after projecting it on a lower dimensional space; which is what all the available conditions (e.g., RIP), guarantee. However, no stable embedding is possible with highly coherent dictionaries. Take as an extreme example the case where the first and second atoms are identical. Then no sensing matrix X can achieve a signal recovery that distinguishes the vector $[1, 0, \dots, 0]^T$ from $[0, 1, 0, \dots, 0]^T$. Can then one conclude that for highly coherent overcomplete dictionaries, compressed sensing techniques are not possible? Fortunately, the answer to this is negative. After all, our goal in compressed sensing has always been the recovery of the signal $s = \Psi\theta$ and not the identification of the sparse vector θ in the synthesis model representation. The latter was just a means to an end. While the unique recovery of θ cannot be guaranteed for highly coherent dictionaries, this does

not necessarily cause any problems for the recovery of s , using a small set of measurement samples. The escape route will come by considering the analysis model formulation.

10.5.1 COMPRESSED SENSING FOR SPARSE SIGNAL REPRESENTATION IN COHERENT DICTIONARIES

Our goal in this subsection is to establish conditions that guarantee recovery of a signal vector, which accepts a sparse representation in a redundant and coherent dictionary, using a small number of signal-related measurements. Let the dictionary at hand be a tight frame, Ψ (Appendix 10.7). Then, our signal vector is written as

$$s = \Psi\theta, \quad (10.31)$$

where θ is assumed to be k -sparse. Recalling the properties of a tight frame, as they are summarized in Appendix 10.7, the coefficients in the expansion (10.31) can be written as $\langle \psi_i, s \rangle$, and the respective vector as

$$\theta = \Psi^T s,$$

because a tight frame is self-dual. Then, the analysis counterpart of the synthesis formulation in (9.39) can be cast as

$$\begin{aligned} \min_s \quad & \|\Psi^T s\|_1, \\ \text{s.t.} \quad & \|y - Xs\|_2^2 \leq \epsilon. \end{aligned} \quad (10.32)$$

The goal now is to investigate the accuracy of the recovered solution to this convex optimization task. It turns out that similar strong theorems are also valid for this problem, as with the case of the synthesis formulation, which was studied in Chapter 9.

Definition 10.1. Let Σ_k be the union of all subspaces spanned by all subsets of k columns of Ψ . A sensing matrix, X , obeys the restricted isometry property adapted to Ψ , (Ψ -RIP) with δ_k , if

$$(1 - \delta_k) \|s\|_2^2 \leq \|Xs\|_2^2 \leq (1 + \delta_k) \|s\|_2^2 : \quad \Psi - \text{RIP Condition}, \quad (10.33)$$

for all $s \in \Sigma_k$.

The union of subspaces, Σ_k , is the image under Ψ of all k -sparse vectors. This is the difference with the RIP definition given in Section 9.7.2. All the random matrices discussed earlier in this chapter can be shown to satisfy this form of RIP, with overwhelming probability, provided the number of observations, N , is at least of the order of $k \ln(l/k)$. We are now ready to establish the main theorem concerning our ℓ_1 minimization task.

Theorem 10.2. Let Ψ be an arbitrary tight frame and X a sensing matrix that satisfies the Ψ -RIP with $\delta_{2k} \leq 0.08$, for some positive k . Then the solution, s_* , of the minimization task in (10.32) satisfies the property

$$\|s - s_*\|_2 \leq C_0 k^{-\frac{1}{2}} \|\Psi^T s - (\Psi^T s)_k\|_1 + C_1 \sqrt{\epsilon}, \quad (10.34)$$

where C_0, C_1 are constants depending on δ_{2k} , and $(\Psi^T s)_k$ denotes the best k -sparse approximation of $\Psi^T s$, that results by setting all but the k largest in magnitude components of $\Psi^T s$ equal to zero.

The bound in (10.34) is the counterpart of that given in (9.36). In other words, the previous theorem states that if $\Psi^T s$ decays rapidly, then s can be reconstructed from just a few (compared to the signal length l) observations. The theorem was first given in [25] and it is the first time that such a theorem provides results for the sparse analysis model formulation in a general context.

10.5.2 COSPARSITY

In the sparse synthesis formulation, one searches for a solution in a union of subspaces that are formed by all possible combinations of k columns of the dictionary, Ψ . Our signal vector lies in one of these subspaces—the one that is spanned by the columns of Ψ whose indices lie in the support set (Section 10.2.1). In the sparse analysis approach, things get different. The kick-off point is the sparsity of the transform $S := \Phi^T s$, where Φ defines the transformation matrix or analysis operator. Because S is assumed to be sparse, there exists an index set \mathcal{I} such that $\forall i \in \mathcal{I}, S_i = 0$. In other words, $\forall i \in \mathcal{I}$, $\phi_i^T s := \langle \phi_i, s \rangle = 0$, where ϕ_i stands for the i th column of Φ . Hence, the subspace in which s lives is the orthogonal complement of the subspace formed by those columns of Φ that correspond to a zero in the transform vector S . Assume, now, that $\text{card}(\mathcal{I}) = C_o$. The signal, s , can be identified by searching on the *orthogonal complements* of the subspaces formed by all possible combinations of C_o columns of Φ , that is,

$$\langle \phi_i, s \rangle = 0, \quad \forall i \in \mathcal{I}.$$

The difference between the synthesis and analysis problems is illustrated in Figure 10.12. To facilitate the theoretical treatment of this new setting, the notion of *cosparsity* was introduced in [92].

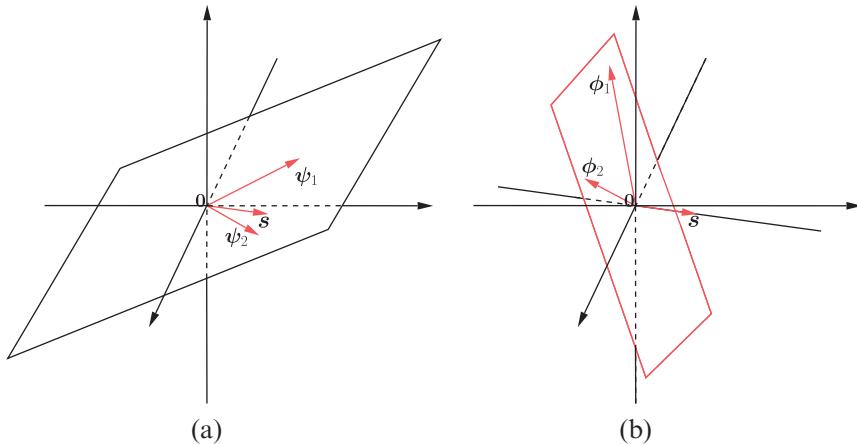


FIGURE 10.12

Searching for a sparse vector s . (a) In the synthesis model, the sparse vector lies in subspaces formed by combinations of k (in this case $k = 2$) columns of the dictionary Ψ . (b) In the analysis model, the sparse vector lies in the orthogonal complement of the subspace formed by C_o (in this case $C_o = 2$) columns of the transformation matrix Φ .

Definition 10.2. The cosparsity of a signal $s \in \mathbb{R}^l$ with respect to a $p \times l$ matrix Φ^T is defined as

$$C_o := p - \|\Phi^T s\|_0. \quad (10.35)$$

In words, the cosparsity is the number of zeros in the obtained transform vector $S = \Phi^T s$; in contrast, the sparsity measures the number of the nonzero elements of the respective sparse vector. If one assumes that Φ has “full spark,”³ that is, $l + 1$, then any l of the columns of Φ , and thus any l rows of Φ^T , are guaranteed to be independent. This indicates that for such matrices, the maximum value that cosparsity can take is equal to $C_o = l - 1$. Otherwise, the existence of l zeros will necessarily correspond to a zero signal vector. Higher cosparsity levels are possible by relaxing the full spark requirement.

Let now the cosparsity of our signal with respect to a matrix Φ^T be C_o . Then, in order to dig out the signal from the subspace in which it is hidden, one must form all possible combinations of C_o columns of Φ and search in their orthogonal complements. In the case that Φ is full rank, we have seen previously that $C_o < l$, and, hence, any set of C_o columns of Φ are linearly independent. In other words, the dimension of the span of those columns is C_o . As a result, the dimensionality of the orthogonal complement, into which we search for s , is $l - C_o$.

We have by now accumulated enough information to elaborate a bit more on the statement made before concerning the different nature of the synthesis and analysis tasks. Let us consider a synthesis task using an $l \times p$ dictionary, and let k be the sparsity level in the corresponding expansion of a signal in terms of this dictionary. The dimensionality of the subspaces in which the solution is sought is k (k is assumed to be less than the spark of the respective matrix). Let us keep the same dimensionality for the subspaces in which we are going to search for a solution in an analysis task. Hence, in this case $C_o = l - k$ (assuming a full spark matrix). Also, for the sake of comparison assume that the analysis matrix is $p \times l$. Solving the synthesis task, one has to search $\binom{p}{k}$ subspaces, while solving the analysis task one has to search $\binom{p}{C_o=l-k}$ subspaces. These are two different numbers; assuming that $k \ll l$ and also that $l < p/2$, which are natural assumptions for overcomplete dictionaries, then the latter of the two numbers is much larger than the former (use your computer to play with some typical values). In other words, there are many more analysis than synthesis low-dimensional subspaces for which to search. The large number of low-dimensional subspaces makes the algorithmic recovery of a solution from the analysis model a tougher task [92]. However, it might reveal a much stronger descriptive power of the analysis model compared to the synthesis one.

Another interesting aspect that highlights the difference between the two approaches is the following. Assume that the synthesis and analysis matrices are related as $\Phi = \Psi$, as was the case for tight frames. Under this assumption, $\Phi^T s$ provides a set of coefficients for the synthesis expansion in terms of the atoms of $\Phi = \Psi$. Moreover, if $\|\Phi^T s\|_0 = k$, then the $\Phi^T s$ is a possible k -sparse solution for the synthesis model. However, there is no guarantee that this is the sparsest one.

It is now time to investigate whether conditions that guarantee uniqueness of the solution for the sparse analysis formulation can be derived. The answer is in the affirmative, and it has been established in [92] for the case of exact measurements.

Lemma 10.1. *Let Φ be a transformation matrix of full spark. Then, for almost all $N \times l$ sensing matrices and for $N > 2(l - C_o)$, the equation*

$$y = Xs,$$

has at most one solution with cosparsity at least C_o .

³ Recall by Definition 9.2 that $\text{spark}(\Phi)$ is defined for an $l \times p$ matrix Φ with $p \geq l$ and of full rank.

The above lemma guarantees the uniqueness of the solution, if one exists, of the optimization

$$\begin{aligned} \min_s \quad & \|\Phi^T s\|_0 \\ \text{s.t.} \quad & y = Xs. \end{aligned} \tag{10.36}$$

However, solving the previous ℓ_0 minimization task is a difficult one, and we know that its synthesis counterpart has been shown to be NP-hard, in general. Its relaxed convex relative is the ℓ_1 minimization

$$\begin{aligned} \min_s \quad & \|\Phi^T s\|_1 \\ \text{s.t.} \quad & y = Xs. \end{aligned} \tag{10.37}$$

In [92], conditions are derived that guarantee the equivalence of the ℓ_0 and ℓ_1 tasks, in (10.36) and (10.37), respectively; this is done in a way similar to that for the sparse synthesis modeling. Also, in [92], a greedy algorithm inspired by the orthogonal matching pursuit, discussed in [Section 10.2.1](#), has been derived. A thorough study on greedy-like algorithms applicable to the cosparse model can be found in [62]. In [103] an iterative analysis thresholding is proposed and theoretically investigated. Other algorithms that solve the ℓ_1 optimization in the analysis modeling framework can be found in, for example, [21, 49, 108]. NESTA can also be used for the analysis formulation. Moreover, a critical aspect affecting the performance of algorithms obeying the cosparse analysis model is the choice of the analysis matrix Φ . It turns out that it is not always the best practice to use fixed and predefined matrices. As a promising alternative, problem-tailored analysis matrices can be learned using the available data (e.g., [106, 119]).

10.6 A CASE STUDY: TIME-FREQUENCY ANALYSIS

The goal of this section is to demonstrate how all the previously stated theoretical findings can be exploited in the context of a real application. Sparse modeling has been applied to almost everything. So picking up a typical application would not be easy. We preferred to focus on a less “publicized” application, that of analyzing echolocation signals emitted by bats. However, the analysis will take place within the framework of time-frequency representation, which is one of the research areas that significantly inspired the evolution of compressed sensing theory. Time-frequency analysis of signals has been a field of intense research for a number of decades, and it is one of the most powerful signal processing tools. Typical applications include speech processing, sonar sounding, communications, biological signals, and EEG processing, to name but a few (see, e.g., [13, 20, 57]).

Gabor transform and frames

It is not our intention to present the theory behind the Gabor transform. Our goal is to outline some basic related notions and use them as a vehicle for the less familiar reader to better understand how redundant dictionaries are used and to get better acquainted with their potential performance benefits.

The Gabor transform was introduced in the mid-1940s by Dennis Gabor (1900–1979), a Hungarian-British engineer. His most notable scientific achievement was the invention of holography, for which he won the Nobel Prize for Physics in 1971.

The discrete version of the Gabor transform can be seen as a special case of the short time Fourier transform (STFT) (e.g., [57, 87]). In the standard DFT transform, the full length of a time sequence, comprising l samples, is used all in “one go” in order to compute the corresponding frequency content. However, the latter can be time varying, so the DFT will provide an average information, which cannot be of much use. The Gabor transform (and the STFT in general) introduces time localization via the use of a window function, which slides along the signal segment in time, and at each time instant focuses on a different part of the signal. This is a way that allows one to follow the slow time variations, which take place in the frequency domain. The time localization in the context of the Gabor transform is achieved via a Gaussian window function, that is,

$$g(n) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{n^2}{2\sigma^2}\right). \quad (10.38)$$

Figure 10.13a shows the Gaussian window, $g(n - m)$, centered at time instant m . The choice of the window spreading factor, σ , will be discussed later on.

Let us now construct the atoms of the Gabor dictionary. Recall that in the case of the signal representation in terms of the DFT in (10.29), each frequency is represented only once, by the corresponding sampled sinusoid, (10.30). In the Gabor transform, each frequency appears l times; the corresponding sampled sinusoid is multiplied by the Gaussian window sequence, each time shifted by one sample. Thus, at the i th frequency bin, we have l atoms, $\mathbf{g}^{(m,i)}$, $m = 0, 1, \dots, l - 1$, with elements given by

$$g^{(m,i)}(n) = g(n - m)\psi_i(n), \quad n, m, i = 0, 1, \dots, l - 1, \quad (10.39)$$

where $\psi_i(n)$ is the n th element of the vector $\boldsymbol{\psi}_i$ in (10.30). This results to an overcomplete dictionary comprising l^2 atoms in the l -dimensional space. **Figure 10.13b** illustrates the effect of multiplying

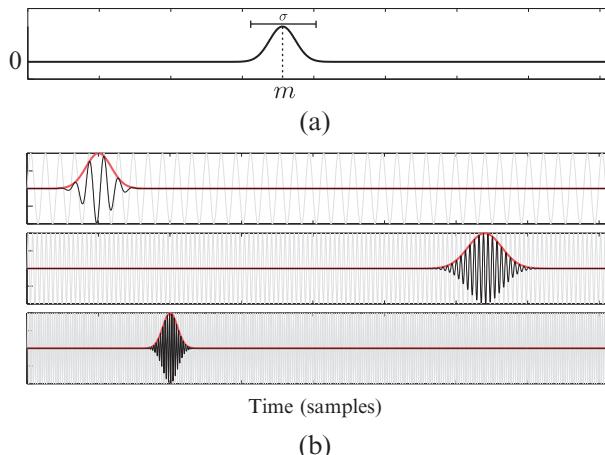
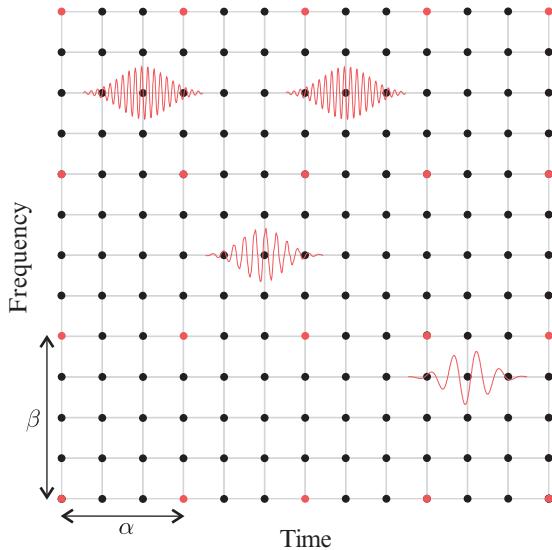


FIGURE 10.13

(a) The Gaussian window with spreading factor σ centered at time instant m . (b) Pulses obtained by windowing three different sinusoids with Gaussian windows of different spread and applied at different time instants.

**FIGURE 10.14**

Each atom of the Gabor dictionary corresponds to a node in the time-frequency grid. That is, it is a sampled windowed sinusoid whose frequency and location in time are given by the coordinates of the respective node. In practice, this grid may be subsampled by factors α and β for the two axes respectively, in order to reduce the number of the involved atoms.

different sinusoids with Gaussian pulses of different spread and at different time delays. Figure 10.14 is a graphical interpretation of the atoms involved in the Gabor dictionary. Each node, (m, i) , in this time-frequency plot, corresponds to an atom of frequency equal to $\frac{2\pi}{l}i$ and delay equal to m .

Note that the windowing of a signal of finite duration inevitably introduces boundary effects, especially when the delay m gets close to the time segment edges, 0 and $l - 1$. A solution that facilitates the theoretical analysis is to use a modulo l arithmetic to wrap around at the edge points (this is equivalent to extending the signal periodically); see, for example, [113].

Once the atoms have been defined, they can be stacked one next to the other to form the columns of the $l \times l^2$ Gabor dictionary, G . It can be shown that the Gabor dictionary is a tight frame, [125].

Time-frequency resolution

By definition of the Gabor dictionary, it is readily understood that the choice of the window spread, as measured by σ , must be a critical factor, since it controls the localization in time. As known from our Fourier transform basics, when the pulse becomes short, in order to increase the time resolution, its corresponding frequency content spreads out, and vice versa. From Heisenberg's principle, we know that we can never achieve high time and frequency resolution simultaneously; one is gained at the expense of the other. It is here where the Gaussian shape in the Gabor transform is justified. It can be shown that the Gaussian window gives the optimal trade-off between time and frequency resolution, [57, 87]. The time-frequency resolution trade-off is demonstrated in Figure 10.15, where three sinusoids are shown windowed with different pulse durations. The diagram shows the corresponding spread in

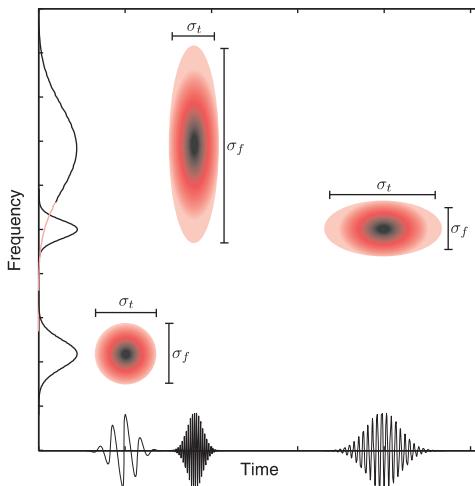


FIGURE 10.15

The shorter the width of the pulsed (windowed) sinusoid is in time, the wider the spread of its frequency content around the frequency of the sinusoid. The Gaussian-like curves along the frequency axis indicate the energy spread in frequency of the respective pulses. The values of σ_t and σ_f indicate the spread in time and frequency, respectively.

the time-frequency plot. The value of σ_t indicates the time spread and σ_f the spread of the respective frequency content around the basic frequency of each sinusoid.

Gabor frames

In practice, l^2 can take large values, and it is desirable to see whether one can reduce the number of the involved atoms without sacrificing the frame-related properties. This can be achieved by an appropriate subsampling, as illustrated in Figure 10.14. We only keep the atoms that correspond to the red nodes. That is, we subsample by keeping every α nodes in time and every β nodes in frequency in order to form the dictionary, that is,

$$G_{(\alpha,\beta)} = \{g^{(m\alpha,i\beta)}\}, \quad m = 0, 1, \dots, \frac{l}{\alpha} - 1, \quad i = 0, 1, \dots, \frac{l}{\beta} - 1,$$

where α and β are divisors of l . Then, it can be shown (e.g., [57]) that if $\alpha\beta < l$ the resulting dictionary retains its frame properties. Once $G_{(\alpha,\beta)}$ is obtained, the canonical dual frame is readily available via (10.46) (adjusted for complex data), from which the corresponding set of expansion coefficients, θ , results.

Time-frequency analysis of echolocation signals emitted by bats

Bats are using echolocation for navigation (flying around at night), for prey detection (small insects), and for prey approaching and catching; each bat adaptively changes the shape and frequency content of its calls in order to better serve the previous tasks. Echolocation is used in a similar way for sonars.

Bats emit calls as they fly, and “listen” to the returning echoes in order to build up a sonic map of their surroundings. In this way, bats can infer the distance and the size of obstacles as well as of other flying creatures/insects. Moreover, all bats emit special types of calls, called social calls, which are used for socializing, flirting, and so on. The fundamental characteristics of the echolocation calls, for example the frequency range and average time duration, differ from species to species because, thanks to evolution, bats have adapted their calls in order to become better suited to the environment in which a species operates.

Time-frequency analysis of echolocation calls provides information about the species (species identification) as well as the specific task and behavior of the bats in certain environments. Moreover, the bat-biosonar system is studied in order for humans to learn more about nature and get inspired for subsequent advances in applications such as sonar navigation systems, radars, medical ultrasonic devices, and more.

[Figure 10.16](#) shows a case of a recorded echolocation signal from bats. Zooming at two different parts of the signal, we can observe that the frequency is changing with time. In [Figure 10.17](#), the DFT of the signal is shown, but there is not much information that can be drawn from it except that the signal is compressible in the frequency domain; most of the activity takes place within a short range of frequencies.

Our echolocation signal was a recording of total length $T = 21.845$ ms, [75]. Samples were taken at the sampling frequency $f_s = 750$ KHz, which results in a total of $l = 16384$ samples. Although the signal itself is not sparse in the time domain, we will take advantage of the fact that it is sparse in a transformed domain. We will assume that the signal is sparse in its expansion in terms of the Gabor dictionary.

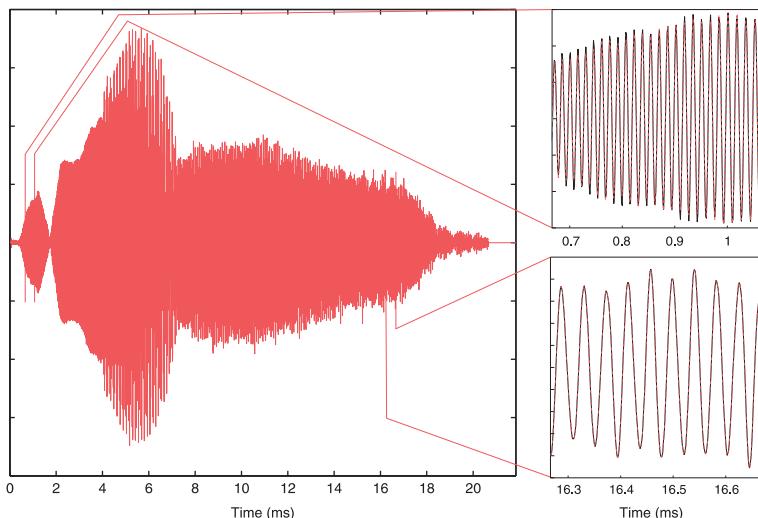


FIGURE 10.16

The recorded echolocation signal. The frequency of the signal is time varying, which is indicated by focusing on two different parts of the signal.

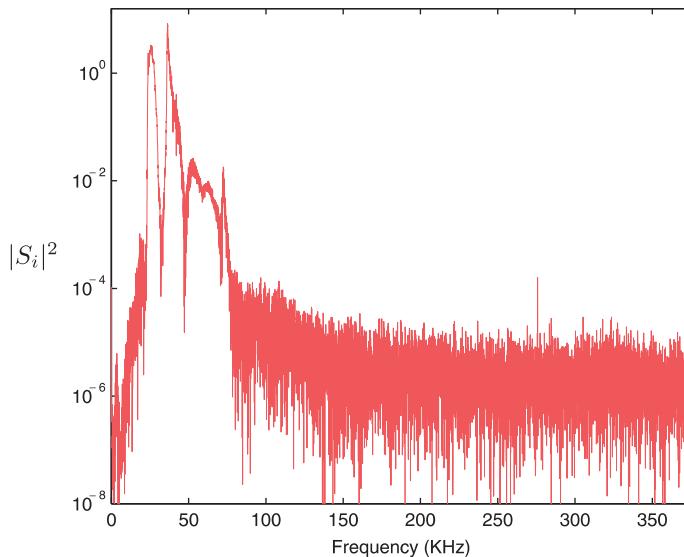


FIGURE 10.17

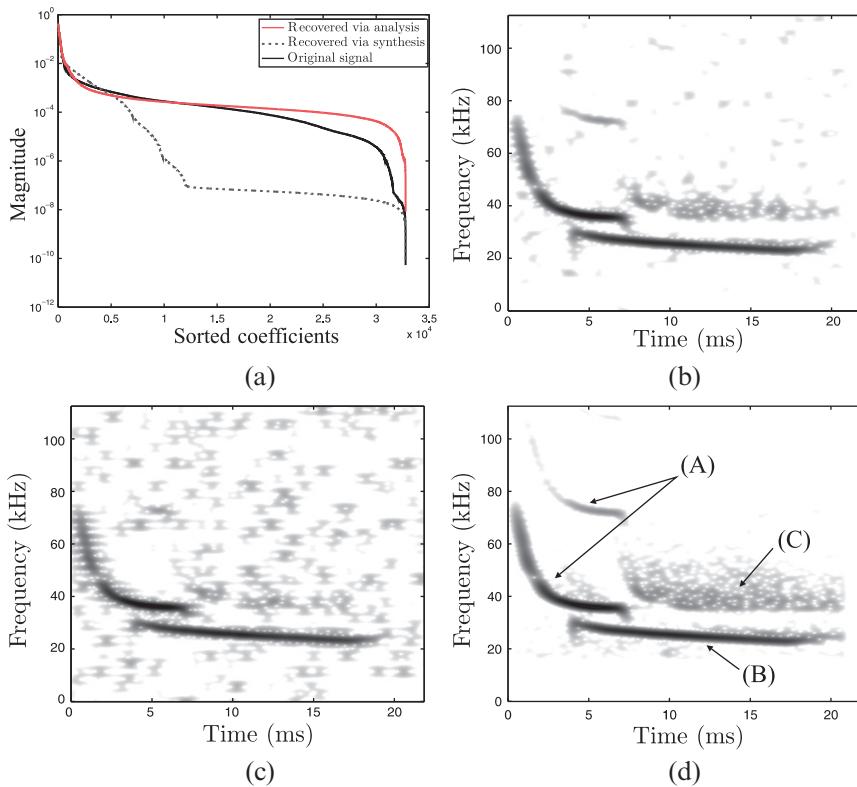
Plot of the energy of the DFT transform coefficients, S_i . Observe that most of the frequency activity takes place within a short frequency range.

Our goal in this example is to demonstrate that one does not really need all 16,384 samples to perform time-frequency analysis; all the processing can be carried out using a reduced number of observations, by exploiting the theory of compressed sensing. To form the measurements vector, y , the number of observations was chosen to be $N = 2048$. This amounts to a reduction of eight times with respect to the number of available samples. The observations vector was formed as

$$y = Xs,$$

where X is an $N \times l$ sensing matrix comprising ± 1 generated in a random way. This means that once we obtain y , we do not need to store the original samples anymore, leading to a savings in memory. Ideally, one could have obtained the reduced number of observations by sampling directly the analog signal at sub-Nyquist rates, as has already been discussed at the end of [Section 9.9](#). Another goal is to use both the analysis and synthesis models and demonstrate their difference.

Three different spectrograms were computed. Two of them, shown in [Figure 10.18b](#) and [c](#), correspond to the reconstructed signals obtained by the analysis ([10.37](#)) and the synthesis ([9.37](#)) formulations, respectively. In both cases, the NESTA algorithm was used and the $G_{(128,64)}$ frame was employed. Note that the latter dictionary is redundant by a factor of 2. The spectrograms are the result of plotting the time-frequency grid and coloring each node (t, i) according to the energy $|\theta|^2$ of the coefficient associated with the respective atom in the Gabor dictionary. The full Gabor transform was applied on the reconstructed signals to obtain the spectrograms, in order to get better coverage of the time-frequency grid. The scale is logarithmic and the darker areas correspond to larger values. The spectrogram of the original signal obtained via the full Gabor transform is shown

**FIGURE 10.18**

(a) Plot of the magnitude of the coefficients, sorted in decreasing order, in the expansion in terms of the $G_{(128,64)}$ Gabor frame. The results correspond to the analysis and synthesis model formulations. The third curve corresponds to the case of analyzing the original vector signal directly, by projecting it on the dual frame. (b) The spectrogram from the analysis and (c) the spectrogram from the synthesis formulations, respectively. (d) The spectrogram corresponding to $G_{(64,32)}$ frame using the analysis formulation. For all cases, the number of observations used was one eighth of the total number of signal samples. A, B, and C indicate different parts of the signal, as explained in the text.

in Figure 10.18d. It is evident that the analysis model resulted in a more clear spectrogram, which resembles the original one better. When the frame $G_{(64,32)}$ is employed, which is a highly redundant Gabor dictionary comprising $8l$ atoms, then the analysis model results in a recovered signal whose spectrogram is visually indistinguishable from the original one in Figure 10.18d.

Figure 10.18a is the plot of the magnitude of the corresponding Gabor transform coefficients, sorted in decreasing values. The synthesis model provides a sparser representation in the sense that the coefficients decrease much faster. The third curve is the one that results if we multiply the dual frame matrix $\tilde{G}_{(128,64)}$ directly with the vector of the original signal samples, and it is shown for comparison reasons.

To conclude, the curious reader may wonder what these curves in Figure 10.18d mean after all. The call denoted by (A) belongs to a Pipistrellus pipistrellus (!) and the call denoted by (B) is either a social call or belongs to a different species. The (C) is the return echo from the signal (A). The large spread in time of (C) indicates a highly reflective environment [75].

10.7 APPENDIX TO CHAPTER 10: SOME HINTS FROM THE THEORY OF FRAMES

In order to remain in the same framework as the one already adopted for this chapter and comply with the notation previously used, we will adhere to the real data case, although everything we will say is readily extended to the complex-valued case by replacing transposition with its Hermitian counterpart.

A frame in a vector space⁴ $V \subseteq \mathbb{R}^l$ is a generalization of the notion of a basis. Recall from our linear algebra basics (see also Section 8.15) that *basis* is a set of vectors $\psi_i, i \in \mathcal{I}$, with the following two properties: (a) $V = \text{span}\{\psi_i : i \in \mathcal{I}\}$, where the cardinality $\text{card}(\mathcal{I}) = l$; and (b) $\psi_i, i \in \mathcal{I}$, are linearly independent. If, in addition, $\langle \psi_i, \psi_j \rangle = \delta_{i,j}$ then the basis is known as orthonormal. If we now relax the second condition and allow $l < \text{card}(\mathcal{I})$, we introduce redundancy in the signal representations, which, as has already been mentioned, can offer a number of advantages in a wide range of applications. However, once redundancy is introduced, we lose uniqueness in the signal representation

$$s = \sum_{i \in \mathcal{I}} \theta_i \psi_i, \quad (10.40)$$

due to the dependency among the vectors ψ_i . The question that is now raised is whether there is a simple and systematic way to compute the coefficients θ_i in the previous expansion.

Definition 10.3. The set $\psi_i, i \in \mathcal{I}$, which spans a vector space, V , is called a *frame* if there exist positive real numbers, A and B , such that for every nonzero $s \in V$,

$$0 < A \|s\|_2^2 \leq \sum_{i \in \mathcal{I}} |\langle \psi_i, s \rangle|^2 \leq B \|s\|_2^2, \quad (10.41)$$

where A and B are known as the bounds of the frame.

Note that if $\psi_i, i \in \mathcal{I}$, comprise an orthonormal basis, then $A = B = 1$ and (10.41) is the celebrated Parseval's theorem. Thus, (10.41) can be considered a generalization of Parseval's theorem. Looking more carefully, we notice that this is a stability condition that closely resembles our familiar RIP condition in (9.29). Indeed, the upper bound guarantees that the expansion never diverges (this applies to infinite dimensional spaces) and the lower bound guarantees that no nonzero vector, $\|s\| \neq 0$, will never become zero after projecting it along the atoms of the frame. To look at it from a slightly different perspective, form the dictionary matrix

$$\Psi = [\psi_1, \psi_2, \dots, \psi_p],$$

where we used p to denote the cardinality of \mathcal{I} . Then, the lower bound in (10.41) guarantees that s can be reconstructed from its transform samples $\Psi^T s$; note that in such a case, if $s_1 \neq s_2$, then their respective transform values will be different.

⁴ We constrain our discussion in this section to finite dimensional Euclidean spaces. The theory of frames has been developed for general Hilbert spaces.

It can be shown that if condition (10.41) is valid, then there exists another set of vectors, $\tilde{\psi}_i, i \in \mathcal{I}$, known as the *dual frame*, with the following elegant property:

$$s = \sum_{i \in \mathcal{I}} \langle \tilde{\psi}_i, s \rangle \tilde{\psi}_i = \sum_{i \in \mathcal{I}} \langle \psi_i, s \rangle \tilde{\psi}_i, \quad \forall s \in V. \quad (10.42)$$

Once a dual frame is available, the coefficients in the expansion of a vector in terms of the atoms of a frame are easily obtained. If we form the matrix $\tilde{\Psi}$ of the dual frame vectors, then it easily checks out that since condition (10.42) is true for any s , it implies that

$$\tilde{\Psi} \Psi^T = \Psi \tilde{\Psi}^T = I, \quad (10.43)$$

where I is the $l \times l$ identity matrix. Note that all of us have used the property in (10.42), possibly in a disguised form, many times in our professional life. Indeed, consider the simple case of two linearly independent vectors in the two-dimensional space (in order to make things simple). Then, (10.40) becomes

$$s = \theta_1 \psi_1 + \theta_2 \psi_2 = \Psi \theta.$$

Solving for the unknown θ is nothing but the solution of a linear set of equations; note that the involved matrix Ψ is invertible. Let us rephrase a bit our familiar solution

$$\theta = \Psi^{-1} s := \tilde{\Psi} s := \begin{bmatrix} \tilde{\psi}_1^T \\ \tilde{\psi}_2^T \end{bmatrix} s, \quad (10.44)$$

where $\tilde{\psi}_i^T, i = 1, 2$, are the rows of the inverse matrix. Using now the previous notation, it is readily seen that

$$s = \langle \tilde{\psi}_1, s \rangle \psi_1 + \langle \tilde{\psi}_2, s \rangle \psi_2.$$

Moreover, note that in this special case of independent vectors, the respective definitions imply

$$\begin{bmatrix} \tilde{\psi}_1^T \\ \tilde{\psi}_2^T \end{bmatrix} [\psi_1, \psi_2] = I,$$

and the dual frame is not only unique but it also fulfills the *biorthogonality* condition, that is,

$$\langle \tilde{\psi}_i, \psi_j \rangle = \delta_{i,j}. \quad (10.45)$$

In the case of a general frame, the dual frames are neither biorthogonal nor uniquely defined. The latter can also be verified by the condition (10.43) that defines the respective matrices. Ψ^T is a rectangular tall matrix and its left inverse is not unique. There is, however, a uniquely defined dual frame, known as the *canonical dual frame*, given as

$$\tilde{\psi}_i := (\Psi \Psi^T)^{-1} \psi_i, \text{ or } \tilde{\Psi} := (\Psi \Psi^T)^{-1} \Psi. \quad (10.46)$$

Another family of frames of special type are the *tight frames*. For tight frames, the two bounds in (10.41) are equal, that is, $A = B$. Thus, once a tight frame is available, we can normalize each vector in the frame as

$$\psi_i \longmapsto \frac{1}{\sqrt{A}} \psi_i,$$

which then results to the *Parseval tight frame*; the condition (10.41) now becomes similar in appearance to our familiar Parseval's theorem for orthonormal bases

$$\sum_{i \in \mathcal{I}} |\langle \psi_i, s \rangle|^2 = \|s\|_2^2. \quad (10.47)$$

Moreover, it can be shown (Problem 10.9) that a Parseval tight frame coincides with its canonical dual frame (that is, it is self-dual) and we can write

$$s = \sum_{i \in \mathcal{I}} \langle \psi_i, s \rangle \psi_i,$$

or in matrix form

$$\tilde{\Psi} = \Psi, \quad (10.48)$$

which is similar with what we know for orthonormal bases; however in this case, orthogonality does not hold.

We will conclude this subsection with a simple example of a Parseval (tight) frame, known as the *Mercedes Benz (MB)*,

$$\Psi = \begin{bmatrix} 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \sqrt{\frac{2}{3}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} \end{bmatrix}.$$

One can easily check that all the properties of a Parseval tight frame are fulfilled.

If constructing a frame, especially in high-dimensional spaces, sounds a bit difficult, the following theorem (from Naimark, see, for example, [67]) offers a systematic method for such constructions.

Theorem 10.3. A set $\{\psi_i\}_{i \in \mathcal{I}}$ in a Hilbert space \mathbb{H}_s is a Parseval tight frame if and only if it can be obtained via an orthogonal projection, $P_{\mathbb{H}_s} : \mathbb{H} \rightarrow \mathbb{H}_s$, of an orthonormal basis $\{e_i\}_{i \in \mathcal{I}}$ in a larger Hilbert space \mathbb{H} , such that $\mathbb{H}_s \subset \mathbb{H}$.

To verify the theorem, check that the MB frame is obtained by orthogonally projecting the three-dimensional orthonormal basis

$$e_1 = \begin{bmatrix} 0 \\ -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}, \quad e_2 = \begin{bmatrix} \sqrt{\frac{2}{3}} \\ -\frac{1}{\sqrt{6}} \\ -\frac{1}{\sqrt{6}} \end{bmatrix}, \quad e_3 = \begin{bmatrix} \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{3}} \end{bmatrix},$$

using the projection matrix

$$P_{\mathbb{H}_s} := \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & -\frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & -\frac{1}{3} & \frac{2}{3} \end{bmatrix}.$$

Observe that the effect of the projection

$$P_{\mathbb{H}_s}[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3] = \Psi^T,$$

is to set \mathbf{e}_3 to the zero vector.

Frames were introduced by Duffin and Schaeffer in their study on nonharmonic Fourier series in 1952 [47] and they remained rather obscured until they were used in the context of wavelet theory (e.g., [36]). The interested reader can obtain the proofs of what has been said in this section from these references. An introductory review with a lot of engineering flavor can be found in [81], where the major references in the field are given.

PROBLEMS

- 10.1** Show that the step, in a greedy algorithm, that selects the column of the sensing matrix, in order to maximize the correlation between the column and the currently available error vector $\mathbf{e}^{(i-1)}$, is equivalent with selecting the column that reduces the ℓ_2 norm of the error vector.
Hint. All the parameters obtained in previous steps are fixed, and the optimization is with respect to the new column as well as the corresponding weighting coefficient in the estimate of the parameter vector.
- 10.2** Prove the proposition stating that if there is a sparse solution to the linear system $\mathbf{y} = X\boldsymbol{\theta}$ such that

$$k_0 = \|\boldsymbol{\theta}\|_0 < \frac{1}{2} \left(1 + \frac{1}{\mu(X)} \right),$$

where $\mu(X)$ is the mutual coherence of X , then the column selection procedure in a greedy algorithm will always select a column among the active columns of X , which correspond to the support of $\boldsymbol{\theta}$; that is, the columns that take part in the representation of \mathbf{y} in terms of the columns of X .

Hint. Assume that

$$\mathbf{y} = \sum_{i=1}^{k_0} \theta_i \mathbf{x}_i.$$

- 10.3** Give an explanation to justify why in step 4 of the CoSaMP algorithm the value of t is taken to be equal to $2k$.
10.4 Show that if

$$J(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) = \frac{1}{2} \|\mathbf{y} - X\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1 + \frac{1}{2} d(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}),$$

where

$$d(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) := c \|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|_2^2 - \|X\boldsymbol{\theta} - X\tilde{\boldsymbol{\theta}}\|_2^2.$$

then minimization results to

$$\hat{\boldsymbol{\theta}} = S_{\lambda/c} \left(\frac{1}{c} X^T (\mathbf{y} - X\tilde{\boldsymbol{\theta}}) + \tilde{\boldsymbol{\theta}} \right).$$

- 10.5** Prove the basic recursion of the parallel coordinate descent algorithm.

Hint. Assume that at the i th iteration, it is the turn of the j th component to be updated, so that the following is minimized:

$$J(\theta_j) = \frac{1}{2} \|\mathbf{y} - X\boldsymbol{\theta}^{(i-1)} + \theta_j^{(i-1)} \mathbf{x}_j - \theta_j \mathbf{x}_j\|_2^2 + \lambda |\theta_j|.$$

- 10.6** Derive the iterative scheme to minimize the weighted ℓ_1 ball, using a majorization-minimization procedure to minimize $\sum_{i=1}^l \ln(|\theta_i| + \epsilon)$, subject to the observations set, $\mathbf{y} = X\boldsymbol{\theta}$.

Hint. Use the linearization of the logarithmic function to bound it from above, because it is a concave function and its graph is located below its tangent.

- 10.7** Show that the weighted ℓ_1 ball used in SpAPSM is upper bounded by the ℓ_0 norm of the target vector.
10.8 Show that the canonical dual frame minimizes the total ℓ_2 norm of the dual frame, that is,

$$\sum_{i \in \mathcal{I}} \|\tilde{\psi}_i\|_2^2.$$

Hint. Use the result of [Problem 9.10](#).

- 10.9** Show that Parseval's tight frames are self-dual.
10.10 Prove that the bounds A , B of a frame coincide with the maximum and minimum eigenvalues of the matrix product $\Psi\Psi^T$.

MATLAB Exercises

- 10.11** Construct a multitone signal having samples

$$\theta_n = \sum_{j=1}^3 a_j \cos\left(\frac{\pi}{2N}(2m_j - 1)n\right), n = 0, \dots, l-1,$$

where $N = 30$, $l = 2^8$, $\mathbf{a} = [0.3, 1, 0.75]^T$, and $\mathbf{m} = [4, 10, 30]^T$. (a) Plot this signal in the time and in the frequency domain (use the “fft.m” Matlab function to compute the Fourier transform). (b) Build a sensing matrix 30×2^8 with entries drawn from a normal distribution, $\mathcal{N}(0, \frac{1}{\sqrt{N}})$, and recover $\boldsymbol{\theta}$ based on these observations by ℓ_1 minimization using, for example, “solvelasso.m” (see [Matlab exercise 9.21](#)). (c) Build a sensing matrix 30×2^8 , where each of its rows contains only a single nonzero component taking the value 1. Moreover, each column has at most one nonzero component. Observe that the multiplication of this sensing matrix with $\boldsymbol{\theta}$ just picks certain components of $\boldsymbol{\theta}$ (those that correspond to the position of the nonzero value of each row of the sampling matrix). Show by solving the corresponding ℓ_1 minimization task, as in question (b), that $\boldsymbol{\theta}$ can be recovered exactly using such a sparse sensing matrix (containing only 30 nonzero components!). Observe that the unknown $\boldsymbol{\theta}$ is sparse in the frequency domain and give an explanation why the recovering is successful with the specific sparse sensing matrix.

- 10.12** Implement the OMP algorithm (see [10.2.1](#)) as well as the CSMP (see [10.2.1](#)) with $t = 2k$. Assume a compressed sensing system using normal distributed sensing matrix. (a) Compare the two algorithms in the case where $\alpha = \frac{N}{l} = 0.2$ for $\beta = \frac{k}{N}$ taking values in the set $\{0.1, 0.2, 0.3, \dots, 1\}$ (choose yourself a signal and a sensing matrix in order to comply with

the recommendations above). (b) Repeat the same test when $\alpha = 0.8$. Observe that this experiment, if it is performed for many different α values, $0 \leq \alpha \leq 1$, can be used for the estimation of phase transition diagrams such as the one depicted in [Figure 10.4](#). (c) Repeat (a) and (b) with the obtained measurements now contaminated with noise corresponding to 20 dB SNR.

- 10.13** Reproduce the MRI reconstruction experiment of [Figure 10.9](#) by running the matlab script “MRIcs.m,” which is available from the website of the book.
- 10.14** Reproduce the bat echolocation Time-Frequency analysis experiment of [Figure 10.18](#) by running the Matlab script “BATcs.m,” which is available from the website of the book.

REFERENCES

- [1] M.G. Amin (Ed.), Compressive Sensing for Urban Radar, CRC Press, 2104.
- [2] M.R. Andersen, Sparse inference using approximate message passing, MSc Thesis, Technical University of Denmark, Department of Applied Mathematics and Computing, 2014.
- [3] D. Angelosante, J.A. Bazerque, G.B. Giannakis, Online adaptive estimation of sparse signals: where RLS meets the ℓ_1 -norm, IEEE Trans. Signal Proc. 58(7) (2010) 3436-3447.
- [4] M. Asif, J. Romberg, Dynamic Updating for ℓ_1 minimization, IEEE J. Selected Topics Signal Process. 4(2) (2010) 421-434.
- [5] M. Asif, J. Romberg, On the LASSO and Dantzig selector equivalence, in: Proceedings of the Conference on Information Sciences and Systems (CISS), Princeton, NJ, March 2010.
- [6] F. Bach, Optimization with sparsity-inducing penalties, Foundat. Trends Machine Learn. 4 (2012) 1-106.
- [7] F. Bach, R. Jenatton, J. Mairal, G. Obozinski, Structured sparsity through convex optimization, Stat. Sci. 27(4) (2012) 450-468.
- [8] S. Bakin, Adaptive regression and model selection in data mining problems, Ph.D. thesis, Australian National University, 1999.
- [9] R. Baraniuk, V. Cevher, M. Wakin, Low-dimensional models for dimensionality reduction and signal recovery: A geometric perspective, Proc. IEEE 98(6) (2010) 959-971.
- [10] R.G. Baraniuk, V. Cevher, M.F. Duarte, C. Hegde, Model-based compressive sensing, IEEE Trans. Informat. Theory 56(4) (2010) 1982-2001.
- [11] A. Beck, M. Teboulle, A fast iterative shrinkage algorithm for linear inverse problems, SIAM J. Imaging Sci. 2(1) (2009) 183-202.
- [12] S. Becker, J. Bobin, E.J. Candès, NESTA: A fast and accurate first-order method for sparse recovery, SIAM J. Imaging Sci. 4(1) (2011) 1-39.
- [13] A. Belouchrani, M.G. Amin, Blind source separation based on time-frequency signal representations, IEEE Trans. Signal Process. 46 (11) (1998) 2888-2897.
- [14] E. van den Berg, M.P. Friedlander, Probing the pareto frontier for the basis pursuit solutions, SIAM J. Sci. Comput. 31(2) (2008) 890-912.
- [15] P. Bickel, Y. Ritov, A. Tsybakov, Simultaneous analysis of LASSO and Dantzig selector, Ann. Stat. 37(4) (2009) 1705-1732.
- [16] A. Blum, Random projection, margins, kernels and feature selection, Lecture Notes on Computer Science (LNCS), 2006, pp. 52-68.
- [17] T. Blumensath, M.E. Davies, Iterative hard thresholding for compressed sensing, Appl. Comput. Harmonic Anal. 27(3) (2009) 265-274.
- [18] T. Blumensath, M.E. Davies, Sampling theorems for signals from the union of finite-dimensional linear subspaces, IEEE Trans. Informat. Theory 55(4) (2009) 1872-1882.

- [19] T. Blumensath, M.E. Davies, Normalized iterative hard thresholding: guaranteed stability and performance, *IEEE Selected Topics Signal Process.* 4(2) (2010) 298-309.
- [20] B. Boashash, *Time Frequency Analysis*, Elsevier, 2003.
- [21] J.F. Cai, S. Osher, Z. Shen, Split Bregman methods and frame based image restoration, *Multiscale Model. Simulat.* 8(2)(2009) 337-369.
- [22] E. Candès, J. Romberg, T. Tao, Robust uncertainty principles: Exact signal reconstruction from highly incomplete Fourier information, *IEEE Trans. Informat. Theory* 52(2) (2006) 489-509.
- [23] E.J. Candès, T. Tao, The Dantzig selector: Statistical estimation when p is much larger than n , *Ann. Stat.* 35(6) (2007) 2313-2351.
- [24] E.J. Candès, M.B. Wakin, S.P. Boyd, Enhancing sparsity by reweighted ℓ_1 minimization, *J. Fourier Anal. Appl.* 14(5) (2008) 877-905.
- [25] E.J. Candès, Y.C. Eldar, D. Needell, P. Randall, Compressed sensing with coherent and redundant dictionaries, *Appl. Comput. Harmonic Anal.* 31(1) (2011) 59-73.
- [26] V. Cevher, P. Indyk, C. Hegde, R.G. Baraniuk, Recovery of clustered sparse signals from compressive measurements, in: International Conference on Sampling Theory and Applications (SAMPTA), Marseille, France, 2009.
- [27] V. Cevher, P. Indyk, L. Carin, R.G. Baraniuk, Sparse signal recovery and acquisition with graphical models, *IEEE Signal Process. Mag.* 27(6) (2010) 92-103.
- [28] S. Chen, D.L. Donoho, M. Saunders, Atomic decomposition by basis pursuit, *SIAM J. Sci. Comput.* 20(1) (1998) 33-61.
- [29] S. Chouvardas, K. Slavakis, Y. Kopsinis, S. Theodoridis, A sparsity promoting adaptive algorithm for distributed learning, *IEEE Trans. Signal Process.* 60(10) (2012) 5412-5425.
- [30] S. Chouvardas, Y. Kopsinis, S. Theodoridis, Sparsity-aware distributed learning, in: A. Hero, J. Moura, T. Luo, S. Cui (Eds.), *Big Data over Networks*, Cambridge University Press, 2014.
- [31] P.L. Combettes, V.R. Wajs, Signal recovery by proximal forward-backward splitting, *SIAM J. Multiscale Model. Simulat.* 4(4) (2005) 1168-1200.
- [32] P.L. Combettes, J.-C. Pesquet, Proximal splitting methods in signal processing, in: *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, Springer-Verlag, 2011.
- [33] W. Dai, O. Milenkovic, Subspace pursuit for compressive sensing signal reconstruction, *IEEE Trans. Informat. Theory* 55(5) (2009) 2230-2249.
- [34] I. Daubechies, M. Defrise, C. De-Mol, An iterative thresholding algorithm for linear inverse problems with a sparsity constraint, *Commun. Pure Appl. Math.* 57(11) (2004) 1413-1457.
- [35] I. Daubechies, R. DeVore, M. Fornasier, C.S. Güntürk, Iteratively reweighted least squares minimization for sparse recovery, *Commun. Pure Appl. Math.* 63(1) (2010) 1-38.
- [36] I. Daubechies, A. Grossman, Y. Meyer, Painless nonorthogonal expansions, *J. Math. Phys.* 27 (1986) 1271-1283.
- [37] M.A. Davenport, M.B. Wakin, Analysis of orthogonal matching pursuit using the restricted isometry property, *IEEE Trans. Informat. Theory* 56(9) (2010) 4395-4401.
- [38] R.A. DeVore, V.N. Temlyakov, Some remarks on greedy algorithms, *Adv. Comput. Math.* 5 (1996) 173-187.
- [39] P. Di Lorenzo, A.H. Sayed, Sparse distributed learning based on diffusion adaptation, *IEEE Trans. Signal Process.* 61(6) (2013) 1419-1433.
- [40] D.L. Donoho, J. Tanner, Neighborliness of randomly-projected simplices in high dimensions, in: *Proceedings on National Academy of Sciences*, 2005, pp. 9446-9451.
- [41] D.A. Donoho, A. Maleki, A. Montanari, Message-passing algorithms for compressed sensing, *Proc. Natl Acad. Sci. USA* 106(45) (2009) 18914-18919.
- [42] D.L. Donoho, J. Tanner, Counting the faces of randomly projected hypercubes and orthants, with applications, *Discrete Comput. Geomet.* 43(3) (2010) 522-541.

- [43] D.L. Donoho, J. Tanner, Precise undersampling theorems, Proc. IEEE 98(6) (2010) 913-924.
- [44] D.L. Donoho, I.M. Johnstone, Ideal spatial adaptation by wavelet shrinkage, Biometrika 81(3) (1994) 425-455.
- [45] D. Donoho, I. Johnstone, G. Kerkyacharian, D. Picard, Wavelet shrinkage: asymptopia? J. R. Stat. Soc. B 57 (1995) 301-337.
- [46] J. Duchi, S.S. Shwartz, Y. Singer, T. Chandra, Efficient projections onto the ℓ_1 -ball for learning in high dimensions, in: Proceedings of the International Conference on Machine Learning (ICML), 2008, pp. 272-279.
- [47] R.J. Duffin, A.C. Schaeffer, A class of nonharmonic Fourier series, Trans. Amer. Math. Soc. 72 (1952) 341-366.
- [48] B. Efron, T. Hastie, I.M. Johnstone, R. Tibshirani, Least angle regression, Ann. Stat. 32 (2004) 407-499.
- [49] M. Elad, J.L. Starck, P. Querre, D.L. Donoho, Simultaneous cartoon and texture image inpainting using morphological component analysis (MCA), Appl. Comput. Harmonic Anal. 19 (2005) 340-358.
- [50] M. Elad, B. Matalon, M. Zibulevsky, Coordinate and subspace optimization methods for linear least squares with non-quadratic regularization, Appl. Comput. Harmonic Anal. 23 (2007) 346-367.
- [51] M. Elad, P. Milanfar, R. Rubinstein, Analysis versus synthesis in signal priors, Inverse Problems 23 (2007) 947-968.
- [52] M. Elad, Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing, Springer, 2010.
- [53] Y.C. Eldar, P. Kuppinger, H. Bolcskei, Block-sparse signals: Uncertainty relations and efficient recovery, IEEE Trans. Signal Process. 58(6) (2010) 3042-3054.
- [54] Y.C. Eldar, G. Kutyniok, Compressed Sensing: Theory and Applications, Cambridge University Press, 2012.
- [55] J. Fan, R. Li, Variable selection via nonconcave penalized likelihood and its oracle properties, J. Amer. Stat. Assoc. 96(456) (2001) 1348-1360.
- [56] M.A. Figueiredo, R.D. Nowak, An EM algorithm for wavelet-based image restoration, IEEE Trans. Image Process. 12(8) (2003) 906-916.
- [57] P. Flandrin, Time-Frequency/Time-scale Analysis, Academic Press, 1999.
- [58] S. Foucart, Hard thresholding pursuit: an algorithm for compressive sensing, SIAM J. Numer. Anal. 49(6) (2011) 2543-2563.
- [59] J. Friedman, T. Hastie, R. Tibshirani, A note on the group LASSO and a sparse group LASSO, arXiv:1001.0736v1[math.ST] (2010).
- [60] P.J. Garrigues, B. Olshausen, Learning horizontal connections in a sparse coding model of natural images, in: Advances in Neural Information Processing Systems (NIPS), 2008.
- [61] A.C. Gilbert, S. Muthukrishnan, M.J. Strauss, Improved time bounds for near-optimal sparse Fourier representation via sampling, in: Proceedings of SPIE (Wavelets XI), San Diego, CA, 2005.
- [62] R. Giryes, S. Nam, M. Elad, R. Gribonval, M. Davies, Greedy-like algorithms for the cosparse analysis model, Linear Algebra Appl. 441(0) (2014) 22-60.
- [63] T. Goldstein, S. Osher, The split Bregman algorithm for ℓ_1 regularized problems, SIAM J. Imaging Sci. 2(2) (2009) 323-343.
- [64] I.F. Gorodnitsky, B.D. Rao, Sparse signal reconstruction from limited data using FOCUSS: A re-weighted minimum norm algorithm, IEEE Trans. Signal Process. 45(3) (1997) 600-614.
- [65] L. Hageman, D. Young, Applied Iterative Methods. Academic Press, New York, 1981.
- [66] T. Hale, W. Yin, Y. Zhang, A fixed-point continuation method for ℓ_1 regularized minimization with applications to compressed sensing, Tech. Rep. TR07-07, Department of Computational and Applied Mathematics, Rice University, 2007.

- [67] D. Han, D.R. Larson, *Frames, Bases and Group Representations*, American Mathematical Society, Providence, RI, 2000.
- [68] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, second ed., Springer, 2008.
- [69] J.C. Hoch, A.S. Stern, D.L. Donoho, I.M. Johnstone, Maximum entropy reconstruction of complex (phase sensitive) spectra, *J. Magnet. Resonance* 86(2) (1990) 236-246.
- [70] P.A. Jansson, *Deconvolution: Applications in Spectroscopy*. Academic Press, New York, 1984.
- [71] R. Jenatton, J.-Y. Audibert, F. Bach, Structured variable selection with sparsity-inducing norms, *J. Machine Learn. Res.* 12 (2011) 2777-2824.
- [72] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, D. Gorinevsky, An interior-point method for large-scale ℓ_1 -regularized least squares, *IEEE J. Selected Topics Signal Process.* 1(4) (2007) 606-617.
- [73] N.G. Kingsbury, T.H. Reeves, Overcomplete image coding using iterative projection-based noise shaping, in: *Proceedings IEEE International Conference on Image Processing (ICIP)*, 2002, pp. 597-600.
- [74] K. Knight, W. Fu, Asymptotics for the LASSO-type estimators, *Ann. Stat.* 28(5) (2000) 1356-1378.
- [75] Y. Kopsinis, E. Aboutanios, D.E. Waters, S. McLaughlin, Time-frequency and advanced frequency estimation techniques for the investigation of bat echolocation calls, *J. Acoust. Soc. Amer.* 127(2) (2010) 1124-1134.
- [76] Y. Kopsinis, K. Slavakis, S. Theodoridis, Online sparse system identification and signal reconstruction using projections onto weighted ℓ_1 balls, *IEEE Trans. Signal Process.* 59(3) (2011) 936-952.
- [77] Y. Kopsinis, K. Slavakis, S. Theodoridis, S. McLaughlin, Reduced complexity online sparse signal reconstruction using projections onto weighted ℓ_1 balls, in: *Digital Signal Processing (DSP)*, 2011 17th International Conference on, July 2011, pp. 1-8.
- [78] Y. Kopsinis, K. Slavakis, S. Theodoridis, S. McLaughlin, Generalized thresholding sparsity-aware algorithm for low complexity online learning, in: *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Kyoto, Japan, March 2012, pp. 3277-3280.
- [79] Y. Kopsinis, K. Slavakis, S. Theodoridis, S. McLaughlin, Thresholding-based online algorithms of complexity comparable to sparse LMS methods, in: *Circuits and Systems (ISCAS)*, 2013 IEEE International Symposium on, May 2013, pp. 513-516.
- [80] Y. Kopsinis, S. Chouvardas, S. Theodoridis, Sparsity-aware learning in the context of echo cancelation: A set theoretic estimation approach, in: *Proceedings of the European Signal Processing Conference (EUSIPCO)*, Lisbon, Portugal, September 2014.
- [81] J. Kovacevic, A. Chebira, Life beyond bases: the advent of frames, *IEEE Signal Process. Mag.* 24(4) (2007) 86-104.
- [82] J. Langford, L. Li, T. Zhang, Sparse online learning via truncated gradient, *J. Machine Learn. Res.* 10 (2009) 777-801.
- [83] Y.M. Lu, M.N. Do, Sampling signals from a union of subspaces, *IEEE Signal Process. Mag.* 25(2) (2008) 41-47.
- [84] Z.Q. Luo, P. Tseng, On the convergence of the coordinate descent method for convex differentiable minimization, *J. Optim. Theory Appl.* 72(1) (1992) 7-35.
- [85] A. Maleki, D.L. Donoho, Optimally tuned iterative reconstruction algorithms for compressed sensing, *IEEE J. Selected Topics Signal Process.* 4(2) (2010) 330-341.
- [86] D.M. Malioutov, M. Cetin, A.S. Willsky, Homotopy continuation for sparse signal representation, in: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2005, pp. 733-736.
- [87] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, third ed., Academic Press, 2008.
- [88] S. Mallat, S. Zhang, Matching pursuit in a time-frequency dictionary, *IEEE Trans. Signal Process.* 41 (1993) 3397-3415.
- [89] G. Mateos, J. Bazerque, G. Giannakis, Distributed sparse linear regression, *IEEE Trans. Signal Process.* 58(10) (2010) 5262-5276.

- [90] G. Mileounis, B. Babadi, N. Kalouptsidis, V. Tarokh, An adaptive greedy algorithm with application to nonlinear communications, *IEEE Trans. Signal Process.* 58(6) (2010) 2998-3007.
- [91] A. Mousavi, A. Maleki, R.G. Baraniuk, Parameterless optimal approximate message passing, *arXiv:1311.0035v1[cs.IT]* 2013.
- [92] S. Nam, M. Davies, M. Elad, R. Gribonval, The cosparse analysis model and algorithms, *Appl. Comput. Harmonic Anal.* 34(1) (2013) 30-56.
- [93] D. Needell, J.A. Tropp, COSAMP: iterative signal recovery from incomplete and inaccurate samples, *Appl. Comput. Harmonic Anal.* 26(3) (2009) 301-321.
- [94] D. Needell, R. Ward, Stable image reconstruction using total variation minimization, *SIAM J. Imaging Sci.*, 6(2) (2013) 1035-1058.
- [95] Y. Nesterov, *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2004.
- [96] Y.E. Nesterov, A method for solving the convex programming problem with convergence rate $O(1/k^2)$, *Dokl. Akad. Nauk SSSR* 269 (1983) 543-547 (in Russian).
- [97] G. Obozinski, B. Taskar, M. Jordan, Multi-task feature selection, *Tech. Rep., Department of Statistics, University of California, Berkeley*, 2006.
- [98] G. Obozinski, B. Taskar, M.I. Jordan, Joint covariate selection and joint subspace selection for multiple classification problems, *Stat. Comput.* 20(2) (2010) 231-252.
- [99] M.R. Osborne, B. Presnell, B.A. Turlach, A new approach to variable selection in least squares problems, *IMA J. Numer. Anal.* 20 (2000) 389-403.
- [100] P.M. Pardalos, N. Kovoor, An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds, *Math. Program.* 46 (1990) 321-328.
- [101] F. Parvaresh, H. Vikalo, S. Misra, B. Hassibi, Recovering Sparse Signals Using Sparse Measurement Matrices in Compressed DNA Microarrays, *IEEE J. Selected Topics Signal Process.* 2(3) (2008) 275-285.
- [102] S. Patterson, Y.C. Eldar, I. Keidar, Distributed compressed sensing for static and time-varying networks, *arXiv:1308.6086[cs.IT]* 2014.
- [103] T. Peleg M. Elad, Performance guarantees of the thresholding algorithm for the cosparse analysis model, *IEEE Trans. Informat. Theory* 59(3) (2013) 1832-1845.
- [104] M.D. Plumley, Geometry and homotopy for ℓ_1 sparse representation, in: *Proceedings of the International Workshop on Signal Processing with Adaptive Sparse Structured Representations (SPARS)*, Rennes, France, 2005.
- [105] R.T. Rockafellar, Monotone operators and the proximal point algorithms, *SIAM J. Control Optim.* 14(5) (1976) 877-898.
- [106] R. Rubinstein, R. Peleg, M. Elad, Analysis KSVD: A dictionary-learning algorithm for the analysis sparse model, *IEEE Trans. Signal Process.* 61(3) (2013) 661-677.
- [107] L.I. Rudin, S. Osher, E. Fatemi, Nonlinear total variation based noise removal algorithms, *Physica D Nonlinear Phenomena* 60(1-4) (1992) 259-268.
- [108] I.W. Selesnick, M.A.T. Figueiredo, Signal restoration with overcomplete wavelet transforms: Comparison of analysis and synthesis priors, in: *Proceedings of SPIE*, 2009.
- [109] K. Slavakis, Y. Kopsinis, S. Theodoridis, S. McLaughlin, Generalized thresholding and online sparsity-aware learning in a union of subspaces, *IEEE Trans. Signal Process.* 61(15) (2013) 3760-3773.
- [110] P. Sprechmann, I. Ramirez, G. Sapiro, Y.C. Eldar, CHiLasso: a collaborative hierarchical sparse modeling framework, *IEEE Trans. Signal Process.* 59(9) (2011) 4183-4198.
- [111] J.L. Starck, E.J. Candès, D.L. Donoho, The curvelet transform for image denoising, *IEEE Trans. Image Process.* 11(6) (2002) 670-684.
- [112] J.L. Starck, J. Fadili, F. Murtagh, The undecimated wavelet decomposition and its reconstruction, *IEEE Trans. Signal Process.* 55(10) (2007) 297-309.

- [113] T. Strohmer, Numerical algorithms for discrete Gabor expansions, in: *Gabor Analysis and Algorithms: Theory and Applications*, Birkhauser, Boston, MA, 1998, pp. 267-294.
- [114] V.N. Temlyakov, Nonlinear methods of approximation, *Foundat. Comput. Math.* 3(1) (2003) 33-107.
- [115] J.A. Tropp, Greed is good, *IEEE Trans. Informat. Theory* 50 (2004) 2231-2242.
- [116] Y. Tsai, Sparse solution of underdetermined linear systems: algorithms and applications, Ph.D. thesis, Stanford University, 2007.
- [117] B.A. Turlach, W.N. Venables, S.J. Wright, Simultaneous variable selection, *Technometrics* 47(3) (2005) 349-363.
- [118] S. Wright, R. Nowak, M. Figueiredo, Sparse reconstruction by separable approximation, *IEEE Trans. Signal Process.* 57(7) (2009) 2479-2493.
- [119] M. Yaghoobi, S. Nam, R. Gribonval, M. Davies, Constrained overcomplete analysis operator learning for cosparse signal modelling, *IEEE Trans. Signal Process.* 61(9) (2013) 2341-2355.
- [120] J. Yang, Y. Zhang, W. Yin, A fast alternating direction method for TV $\ell_1 - \ell_2$ signal reconstruction from partial Fourier data, *IEEE Trans. Selected Topics Signal Process.* 4(2) (2010) 288-297.
- [121] W. Yin, S. Osher, D. Goldfarb, J. Darbon, Bregman iterative algorithms for ℓ_1 -minimization with applications to compressed sensing, *SIAM J. Imaging Sci.* 1(1) (2008) 143-168.
- [122] M. Yuan, Y. Lin, Model selection and estimation in regression with grouped variables, *J. R. Stat. Soc. Ser. B* 68(1) (2006) 49-67.
- [123] T. Zhang, Sparse Recovery with orthogonal matching pursuit under RIP, *IEEE Trans. Informat. Theory* 57(9) (2011) 6215-6221.
- [124] M. Zibulevsky, M. Elad, L1-L2 optimization in signal processing, *IEEE Signal Process. Mag.* 27(3) (2010) 76-88.
- [125] M. Zibulevsky, Y.Y. Zeevi, Frame analysis of the discrete Gabor scheme, *IEEE Trans. Signal Process.* 42(4) (1994) 942-945.
- [126] H. Zou, T. Hastie, Regularization and variable selection via the elastic net, *J. R. Stat. Soc. Ser. B* 67(2) (2005) 301-320.
- [127] H. Zou, The adaptive LASSO and its oracle properties, *J. Amer. Stat. Assoc.* 101 (2006) 1418-1429.
- [128] H. Zou, R. Li, One-step sparse estimates in nonconcave penalized likelihood models, *Ann. Stat.* 36(4) (2008) 1509-1533.

This page intentionally left blank

LEARNING IN REPRODUCING KERNEL HILBERT SPACES

11

CHAPTER OUTLINE

11.1	Introduction	516
11.2	Generalized Linear Models.....	516
11.3	Volterra, Wiener, and Hammerstein Models	517
11.4	Cover's Theorem: Capacity of a Space in Linear Dichotomies	520
11.5	Reproducing Kernel Hilbert Spaces	523
11.5.1	Some Properties and Theoretical Highlights	525
11.5.2	Examples of Kernel Functions	526
<i>Constructing Kernels</i>	529	
<i>String Kernels</i>	530	
11.6	Representer Theorem	531
11.6.1	Semiparametric Representer Theorem	533
11.6.2	Nonparametric Modeling: A Discussion	534
11.7	Kernel Ridge Regression	534
11.8	Support Vector Regression	536
11.8.1	The Linear ϵ -Insensitive Optimal Regression	537
<i>The Solution</i>	539	
<i>Solving the Optimization Task</i>	540	
11.9	Kernel Ridge Regression Revisited.....	543
11.10	Optimal Margin Classification: Support Vector Machines.....	544
11.10.1	Linearly Separable Classes: Maximum Margin Classifiers	546
<i>The Solution</i>	549	
<i>The Optimization Task</i>	550	
11.10.2	Nonseparable Classes	551
<i>The Solution</i>	552	
<i>The Optimization Task</i>	552	
11.10.3	Performance of SVMs and Applications	556
11.10.4	Choice of Hyperparameters	556
11.11	Computational Considerations	557
11.11.1	Multiclass Generalizations	558
11.12	Online Learning in RKHS	559
11.12.1	The Kernel LMS (KLMS)	559
11.12.2	The Naive Online R_{reg} Minimization Algorithm (NORMA)	562

<i>Classification: The Hinge Loss Function</i>	564
<i>Regression: The Linear ϵ-insensitive Loss Function</i>	565
<i>Error Bounds and Convergence Performance</i>	565
11.12.3 The Kernel APSM Algorithm	566
<i>Regression</i>	566
<i>Classification</i>	567
11.13 Multiple Kernel Learning	574
11.14 Nonparametric Sparsity-Aware Learning: Additive Models	575
11.15 A Case Study: Authorship Identification	577
Problems	581
<i>MATLAB Exercises</i>	582
References	585

11.1 INTRODUCTION

Our emphasis in this chapter will be on learning nonlinear models. The necessity of adopting nonlinear models has already been discussed in [Chapter 3](#), in the context of the classification as well as the regression tasks. For example, recall that given two jointly distributed random vectors $(\mathbf{y}, \mathbf{x}) \in \mathbb{R}^k \times \mathbb{R}^l$, then we know that the optimal estimate of \mathbf{y} given $\mathbf{x} = \mathbf{x}$, in the mean-square error sense (MSE), is the corresponding conditional mean, that is, $\mathbb{E}[\mathbf{y}|\mathbf{x}]$, which in general is a nonlinear function of \mathbf{x} .

There are different ways of dealing with nonlinear modeling tasks. Our emphasis in this chapter will be on a path through the so-called reproducing kernel Hilbert spaces (RKHS). The technique consists of mapping the input variables to a new space, such that the originally nonlinear task is transformed into a linear one. From a practical point of view, the beauty behind these spaces is that their rich structure allows us to perform inner product operations in a very efficient way, with complexity independent of the dimensionality of the respective RKHS. Moreover, note that the dimension of such spaces can even be infinite.

We start the chapter by reviewing some more “traditional” techniques concerning Volterra series expansions, and then move slowly to explore the RKHS. Cover’s theorem, the basic properties of RKHS and their defining kernels are discussed. Kernel ridge regression and the support vector machine framework is presented. Then we move to online learning algorithms in RKHS and, finally, some more advanced concepts related to sparsity and multikernel representations are discussed. A case study in the context of text mining is presented at the end of the chapter.

11.2 GENERALIZED LINEAR MODELS

Given $(\mathbf{y}, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^l$, a generalized linear estimator $\hat{\mathbf{y}}$ of \mathbf{y} has the form

$$\hat{\mathbf{y}} = f(\mathbf{x}) := \theta_0 + \sum_{k=1}^K \theta_k \phi_k(\mathbf{x}), \quad (11.1)$$

where $\phi_1(\cdot), \dots, \phi_K(\cdot)$ are *preselected* (nonlinear) functions. A popular family of functions is the polynomial one, for example,

$$\hat{y} = \theta_0 + \sum_{i=1}^l \theta_i x_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^l \theta_{im} x_i x_m + \sum_{i=1}^l \theta_{ii} x_i^2. \quad (11.2)$$

Assuming $l = 2$ ($\mathbf{x} = [x_1, x_2]^T$), then (11.2) can be brought into the form of (11.1) by setting $K = 5$ and $\phi_1(\mathbf{x}) = x_1$, $\phi_2(\mathbf{x}) = x_2$, $\phi_3(\mathbf{x}) = x_1 x_2$, $\phi_4(\mathbf{x}) = x_1^2$, $\phi_5(\mathbf{x}) = x_2^2$. The generalization of (11.2) to r th order polynomials is readily obtained and it will contain products of the form $x_1^{p_1} x_2^{p_2} \cdots x_l^{p_l}$, with $p_1 + p_2 + \cdots + p_l \leq r$. It turns out that the number of free parameters, K , for an r th order polynomial is equal to

$$K = \frac{(l+r)!}{r!l!}.$$

Just to get a feeling for $l = 10$ and $r = 3$, $K = 286$. The use of polynomial expansions is justified by the Weierstrass theorem, stating that every continuous function, defined on a compact (closed and bounded) subset $S \subset \mathbb{R}^l$, can be uniformly approximated as closely as desired, with an arbitrary small error, ϵ , by a polynomial function, for example, [97]. Of course, in order to achieve a good enough approximation, one may have to use a large value of r . Besides polynomial functions, other types of functions can also be used, such as splines and trigonometric functions.

A common characteristic of this type of models is that the basis functions in the expansion are *preselected* and they are fixed and independent of the data. The advantage of such a path is that the associated models are linear with respect to the unknown set of free parameters, and they can be estimated by following any one of the methods described for linear models, presented in Chapters 4–8. However, one has to pay a price for that. As it has been shown in [7], for an expansion involving K

fixed functions, the squared approximation error *cannot* be made smaller than order $(\frac{1}{K})^{\frac{2}{l}}$. In other words, for high-dimensional spaces and in order to get a small enough error, one has to use large values of K . This is another face of the curse of dimensionality problem. In contrast, one can get rid of the dependence of the approximation error on the input space dimensionality, l , if the expansion involves data-dependent functions, which are optimized with respect to the specific data set. This is, for example, the case for a class of neural networks, to be discussed in Chapter 18. In this case, the price one pays is that the dependence on the free parameters is now nonlinear, making the optimization with regard to the unknown parameters a harder task.

11.3 VOLTERRA, WIENER, AND HAMMERSTEIN MODELS

We now turn our focus to the case of modeling nonlinear systems, where the involved input-output entities are time series/discrete time signals denoted as (u_n, d_n) , respectively. The counterpart of polynomial modeling in (11.2) is now known as the Volterra series expansion.

These types of models will not be pursued any more in this book, and they are briefly discussed here in order to put the nonlinear modeling task in a more general context as well as for historical reasons. *Thus, this section can be bypassed in a first reading.*

**FIGURE 11.1**

The nonlinear filter is excited by u_n and provides in its output d_n .

Volterra was an Italian mathematician (1860-1940) with major contributions also in physics and biology. One of his landmark theories is the development of Volterra series, which was used to solve integral and integro-differential equations. He was one of the Italian professors who refused to take an oath of loyalty to the fascist regime of Mussolini and he was obliged to resign from his university post.

Figure 11.1 shows an unknown nonlinear system/filter with the respective input-output signals. The output of a discrete time Volterra model can be written as

$$d_n = \sum_{k=1}^r \sum_{i_1=0}^M \sum_{i_2=0}^M \cdots \sum_{i_k=0}^M w_k(i_1, i_2, \dots, i_k) \prod_{j=1}^k u_{n-i_j}, \quad (11.3)$$

where $w_k(\cdot, \dots, \cdot)$ denotes the k th order *Volterra kernel*; in general, r can be infinite. For example, for $r = 2$ and $M = 1$, the input-output relation involves the linear combination of the terms

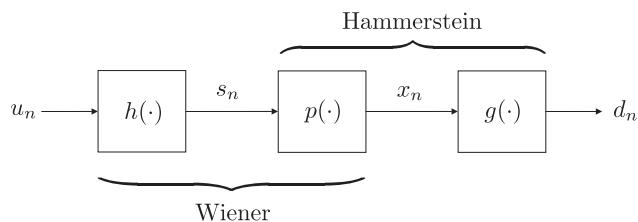
$$u_n, u_{n-1}, u_n^2, u_{n-1}^2, u_n u_{n-1}.$$

Special cases of the Volterra expansion are the *Wiener, Hammerstein, and Wiener-Hammerstein models*. These models are shown in Figure 11.2. The systems $h(\cdot)$ and $g(\cdot)$ are linear systems with memory, that is,

$$s_n = \sum_{i=0}^{M_1} h_n u_{n-i},$$

and

$$d_n = \sum_{i=0}^{M_2} g_n x_{n-i}.$$

**FIGURE 11.2**

The Wiener model comprises a linear filter followed by a memoryless polynomial nonlinearity. The Hammerstein model consists of a memoryless nonlinearity followed by a linear filter. The Wiener-Hammerstein model is the combination of the two.

The central box corresponds to a memoryless nonlinear system, which can be approximated by a polynomial of degree r . Hence,

$$x_n = \sum_{k=1}^r c_k(s_n)^k.$$

In other words, a Wiener model is a linear time invariant system (LTI) followed by the memoryless nonlinearity and the Hammerstein model is the combination of a memoryless nonlinearity followed by an LTI system. The Wiener-Hammerstein model is the combination of the two. Note that each one of these models is nonlinear with respect to the involved free parameters. In contrast, the equivalent Volterra model is linear with regard to the involved parameters; however, the number of the resulting free parameters is significantly increased with the order of the polynomial and the filter memory taps (M_1 and M_2). The interesting feature is that the equivalent to a Hammerstein model Volterra expansion consists only of the diagonal elements of the associated Volterra kernels. In other words, the output is expressed in terms of $u_n, u_{n-1}, u_{n-2}, \dots$ and their powers; there are no cross-product terms [59].

Remarks 11.1.

- The Volterra series expansion was first introduced as a generalization of the Taylor series expansion. Following [102], assume a memoryless nonlinear system. Then, its input-output relationship is given by

$$d(t) = f(u(t)),$$

and adopting the Taylor expansion, for a particular time $t \in (-\infty, +\infty)$, we can write

$$d(t) = \sum_{n=0}^{+\infty} c_n(u(t))^n, \quad (11.4)$$

assuming that the series converges. The Volterra series is the extension of (11.4) to systems with memory and we can write

$$\begin{aligned} d(t) = w_0 &+ \int_{-\infty}^{+\infty} w_1(\tau_1)u(t - \tau_1)d\tau_1 \\ &+ \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} w_2(\tau_1, \tau_2)u(t - \tau_1)u(t - \tau_2)d\tau_1 d\tau_2 \\ &+ \dots \end{aligned} \quad (11.5)$$

In other words, the Volterra series is a power series with memory. The problem of convergence of the Volterra series is similar to that of the Taylor series. In analogy to the Weierstrass approximation theorem, it turns out that the output of a nonlinear system can be approximated arbitrarily close using a sufficient number of terms in the Volterra series expansion¹ [42].

A major difficulty in the Volterra series is the computation of the Volterra kernels. Wiener was the first one to realize the potential of the Volterra series for nonlinear system modeling. In order to compute the involved Volterra kernels, he used the method of orthogonal functionals.

The method resembles the method of using a set of orthogonal polynomials, when one tries to

¹ The proof involves the theory of continuous functionals. A functional is a mapping of a function to the real axis. Observe that each integral is a functional, for a particular t and kernel.

approximate a function via a polynomial expansion, [131]. More on the Volterra modeling and related models can be obtained in, for example, [56, 71, 103]. Volterra models have extensively been used in a number of applications, including communications (e.g., [11]), biomedical engineering (e.g., [73]), and automatic control (e.g., [31]).

11.4 COVER'S THEOREM: CAPACITY OF A SPACE IN LINEAR DICHOTOMIES

We have already justified the method of expanding an unknown nonlinear function in terms of a fixed set of nonlinear ones, by mobilizing arguments from the approximation theory. Although this framework fits perfectly to the regression task, where the output takes values in an interval in \mathbb{R} , such arguments are not well-suited for the classification. In the latter case, the output value is of a discrete nature. For example, in a binary classification task, $y \in \{1, -1\}$, and as long as the sign of the predicted value, \hat{y} , is correct, we do not care how close y and \hat{y} are. In this section, we will present an elegant and powerful theorem that justifies the expansion of a classifier f in the form of (11.1). It suffices to look at (11.1) from a different angle.

Let us consider N points, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N \in \mathbb{R}^l$. We can say that these points are *in general position*, if there is no subset of $l + 1$ of them lying on a $(l - 1)$ -dimensional hyperplane. For example, in the two-dimensional space, any three of these points are not permitted to lie on a straight line.

Theorem 11.1 (Cover's theorem). *The number of groupings, denoted as $\mathcal{O}(N, l)$, that can be formed by $(l - 1)$ -dimensional hyperplanes to separate the N points in two classes, exploiting all possible combinations, is given by ([30], Problem 11.1),*

$$\mathcal{O}(N, l) = 2 \sum_{i=0}^l \binom{N-1}{i},$$

where

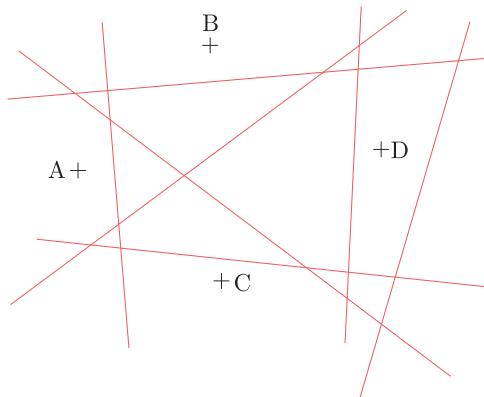
$$\binom{N-1}{i} = \frac{(N-1)!}{(N-1-i)!i!}.$$

Each one of these groupings in two classes is also known as a (linear) *dichotomy*. Figure 11.3 illustrates the theorem for the case of $N = 4$ points in the two-dimensional space. Observe that the possible groupings are [(ABCD)], [A,(BCD)], [B,(ACD)], [C,(ABD)], [D, (ABC)], [(AB), (CD)], and [(AC),(BD)]. Each grouping is counted twice, as it can belong to either ω_1 or ω_2 class. Hence, the total number of groupings is 14, which is equal to $\mathcal{O}(4, 2)$. Note that the number of all possible combinations of N points in two groups is 2^N , which is 16 in our case. The grouping that is not counted in $\mathcal{O}(4, 2)$, as it cannot be linearly separated, is [(BC),(AD)]. Note that if $N \leq l + 1$ then $\mathcal{O}(N, l) = 2^N$. That is, all possible combinations in groups of two are linearly separable; verify it for the case of $N = 3$ in the two-dimensional space.

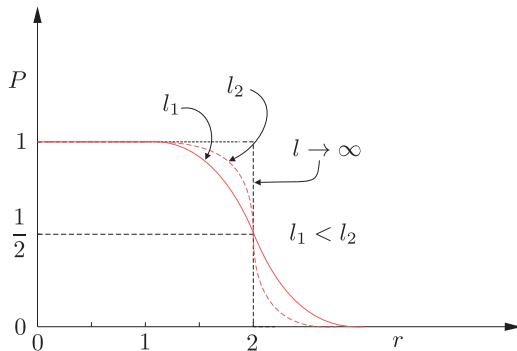
Based on the previous theorem, given N points in the l -dimensional space, the probability of grouping these points in two *linearly separable* classes is

$$P_N^l = \frac{\mathcal{O}(N, l)}{2^N} = \begin{cases} \frac{1}{2^{N-1}} \sum_{i=0}^l \binom{N-1}{i}, & N > l + 1, \\ 1, & N \leq l + 1. \end{cases}$$

(11.6)

**FIGURE 11.3**

The possible number of linearly separable groupings of two, for four points in the two-dimensional space is $\mathcal{O}(4, 2) = 14 = 2 \times 7$.

**FIGURE 11.4**

For $N > 2(l + 1)$ the probability of linear separability becomes small. For large values of l , and provided $N < 2(l + 1)$, the probability of any grouping of the data into two classes to be linearly separable tends to unity. Also, if $N \leq (l + 1)$, all possible groupings in two classes are linearly separable.

To visualize this finding, let us write $N = r(l + 1)$, and express the probability P_N^l in terms of r , for a fixed value of l . The resulting graph is shown in Figure 11.4. Observe that there are two distinct regions. One to the left of the point $r = 2$ and one to the right. At the point $r = 2$, that is, $N = 2(l + 1)$, the probability is always $\frac{1}{2}$, because $\mathcal{O}(2l + 2, l) = 2^{2l+1}$ (Problem 11.2). Note that the larger the value of l is the sharper the transition from one region to the other becomes. Thus, for large dimensional spaces and as long as $N < 2(l + 1)$, the probability of any grouping of the points in two classes to be *linearly separable* tends to unity.

The way the previous theorem is exploited in practice is the following: Given N feature vectors $x_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, a mapping

$$\phi : \mathbb{R}^l \ni x_n \mapsto \phi(x_n) \in \mathbb{R}^K, K \gg l,$$

is performed. Then according to the theorem, the higher the value of K is the higher the probability becomes for the images of the mapping, $\phi(\mathbf{x}_n) \in \mathbb{R}^K$, $n = 1, 2, \dots, N$, to be linearly separable in the space \mathbb{R}^K . Note that the expansion of a nonlinear classifier (that predicts the label in a binary classification task) is equivalent with using a linear one on the images of the original points after the mapping. Indeed,

$$f(\mathbf{x}) = \sum_{k=1}^K \theta_k \phi_k(\mathbf{x}) + \theta_0 = \boldsymbol{\theta}^T \begin{bmatrix} \phi(\mathbf{x}) \\ 1 \end{bmatrix}, \quad (11.7)$$

with

$$\phi(\mathbf{x}) := [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_K(\mathbf{x})]^T.$$

Provided that K is large enough, our task is *linearly separable* in the new space, \mathbb{R}^K with high probability, which justifies the use of linear classifier, $\boldsymbol{\theta}$, in (11.7). The procedure is illustrated in [Figure 11.5](#). The points in the two-dimensional space are not linearly separable. However, after the mapping in the three-dimensional space,

$$[x_1, x_2]^T \longmapsto \phi(\mathbf{x}) = [x_1, x_2, f(x_1, x_2)]^T, \quad f(x_1, x_2) = 4 \exp(-(x_1^2 + x_2^2)/3) + 5,$$

the points in the two classes become linearly separable. Note, however, that after the mapping, the points lie on the surface of a paraboloid. This surface is fully described in terms of two free variables. Loosely speaking, we can think of the two-dimensional plane, on which the data lie originally, to be folded/transformed to form the surface of the paraboloid. This is basically the idea behind the more

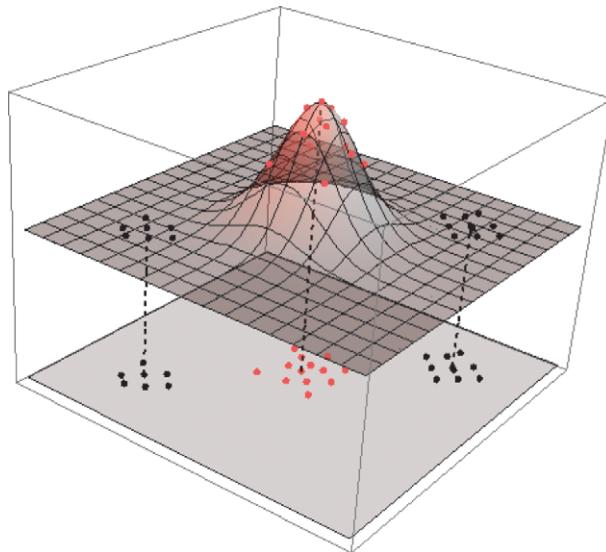


FIGURE 11.5

The points (red in one class and black for the other), that are not linearly separable in the original two-dimensional plane, become linearly separable after the nonlinear mapping in the three-dimensional space; one can draw a plane that separates the “black” from the “red” points.

general problem. After the mapping from the original l -dimensional space to the new K -dimensional one, the images of the points $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$, lie on a l -dimensional surface (*manifold*) in \mathbb{R}^K [17]. We cannot fool nature. Because l variables were originally chosen to describe each pattern (dimensionality, number of free parameters) the same number of free parameters will be required to describe the same objects after the mapping in \mathbb{R}^K . In other words, after the mapping, we embed an l -dimensional manifold in a K -dimensional space, in such a way, that the data in the two classes become linearly separable.

We have by now fully justified the need for mapping the task from the original low-dimensional space to a higher dimensional one, via a set of nonlinear functions. However, life is not easy to work in high-dimensional spaces. A large number of parameters are needed; this in turn poses computational complexity problems, and raises issues related to the generalization and overfitting performance of the designed predictors. In the sequel, we will address the former of the two problems by making a “careful” mapping to a higher dimensional space of a specific structure. The latter problem will be addressed via regularization, as it has already been discussed in various parts in previous chapters.

11.5 REPRODUCING KERNEL HILBERT SPACES

Consider a linear space \mathbb{H} , of real-valued functions defined on a set² $\mathcal{X} \subseteq \mathbb{R}^l$. Furthermore, suppose that \mathbb{H} is a Hilbert space; that is, it is equipped with an inner product operation, $\langle \cdot, \cdot \rangle_{\mathbb{H}}$, that defines a corresponding norm $\|\cdot\|_{\mathbb{H}}$ and \mathbb{H} is complete with respect to this norm.³ From now on, and for notational simplicity, we drop out the subscript \mathbb{H} from the inner product and norm notations and we are going to use them only if it is necessary to avoid confusion.

Definition 11.1. A Hilbert space \mathbb{H} is called *reproducing kernel Hilbert space* (RKHS), if there exists a function

$$\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R},$$

with the following properties:

- For every $\mathbf{x} \in \mathcal{X}$, $\kappa(\cdot, \mathbf{x})$ belongs to \mathbb{H} .
- $\kappa(\cdot, \cdot)$ has the so-called *reproducing property*, that is,

$$f(\mathbf{x}) = \langle f, \kappa(\cdot, \mathbf{x}) \rangle, \forall f \in \mathbb{H}, \forall \mathbf{x} \in \mathcal{X} : \quad \text{Reproducing Property.} \quad (11.8)$$

A direct consequence of the reproducing property, if we set $f(\cdot) = \kappa(\cdot, \mathbf{y})$, $\mathbf{y} \in \mathcal{X}$, is that

$$\langle \kappa(\cdot, \mathbf{y}), \kappa(\cdot, \mathbf{x}) \rangle = \kappa(\mathbf{x}, \mathbf{y}) = \kappa(\mathbf{y}, \mathbf{x}). \quad (11.9)$$

Definition 11.2. Let \mathbb{H} be an RKHS, associated with a kernel function $\kappa(\cdot, \cdot)$, and \mathcal{X} a set of elements. Then, the mapping

$$\mathcal{X} \ni \mathbf{x} \mapsto \phi(\mathbf{x}) := \kappa(\cdot, \mathbf{x}) \in \mathbb{H} : \quad \text{Feature Map,}$$

is known as *feature map* and the space, \mathbb{H} , the *feature space*.

² Generalization to more general sets is also possible.

³ For the unfamiliar reader, a Hilbert space is the generalization of Euclidean space allowing for infinite dimensions. More rigorous definitions and related properties are given in Section 8.15.

In other words, if \mathcal{X} is the set of our observation vectors, the feature mapping maps each vector to the RKHS \mathbb{H} . Note that, in general, \mathbb{H} can be of infinite dimension and its elements can be functions. That is, each training point is mapped to a function. In special cases, where \mathbb{H} becomes a (finite dimensional) Euclidean space, \mathbb{R}^K , the image is a vector $\phi(\mathbf{x}) \in \mathbb{R}^K$. From now on, the general infinite dimensional case will be treated and the images will be denoted as functions, $\phi(\cdot)$. Let us now see what we have gained by choosing to perform the feature mapping from the original space to a high-dimensional RKHS one. Let $\mathbf{x}, \mathbf{y} \in \mathcal{X} \subseteq \mathbb{R}^l$. Then, the inner product of the respective mapping images is written as

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \langle \kappa(\cdot, \mathbf{x}), \kappa(\cdot, \mathbf{y}) \rangle,$$

or

$$\boxed{\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \kappa(\mathbf{x}, \mathbf{y}) : \text{ Kernel Trick.}}$$

In other words, employing this type of mapping to our problem, we can perform inner product operations in \mathbb{H} in a very efficient way; that is, via a function evaluation performed in the original low-dimensional space! This property is also known as the *kernel trick*, and it facilitates significantly the computations. As will become apparent soon, the way this property is exploited in practice involves the following steps:

1. Map (implicitly) the input training data to an RKHS

$$\mathbf{x}_n \longmapsto \phi(\mathbf{x}_n) \in \mathbb{H}, \quad n = 1, 2, \dots, N.$$

2. Solve a *linear* estimation task in \mathbb{H} , involving the images $\phi(\mathbf{x}_n)$, $n = 1, 2, \dots, N$.
3. Cast the algorithm that solves for the unknown parameters in terms of inner product operations, in the form

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle, \quad i, j = 1, 2, \dots, N.$$

4. Replace each inner product by a kernel evaluation, that is,

$$\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \kappa(\mathbf{x}_i, \mathbf{x}_j).$$

It is apparent that one does not need to perform any explicit mapping of the data. All is needed is to perform the kernel operations at the final step. Note that, the specific form of $\kappa(\cdot, \cdot)$ does not concern the analysis. Once the algorithm for the prediction, \hat{y} , has been derived, one can use different choices for $\kappa(\cdot, \cdot)$. As we will see, different choices for $\kappa(\cdot, \cdot)$ correspond to different types of nonlinearity. **Figure 11.6** illustrates the rationale behind the procedure. In practice, the four steps listed above are equivalent to (a) work in the original (low-dimensional Euclidean space) and expressing all operations in terms of inner products and (b) at the final step substitute the inner products with kernel evaluations.

Example 11.1. Consider the case of the two-dimensional space and the mapping

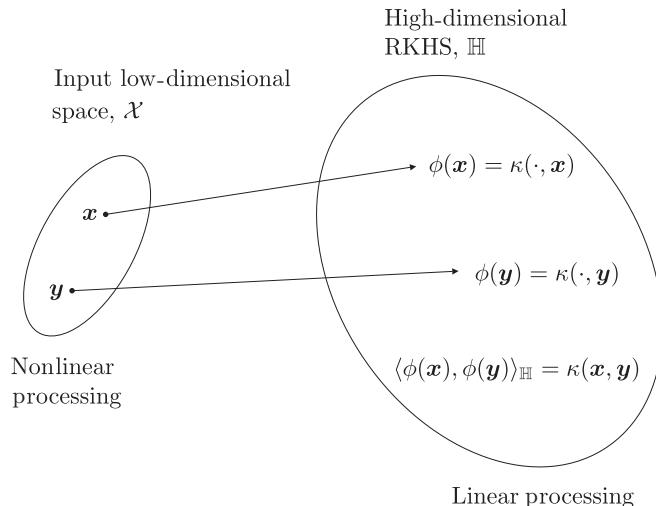
$$\mathbb{R}^2 \ni \mathbf{x} \longmapsto \phi(\mathbf{x}) = [x_1^2, \sqrt{2}x_1x_2, x_2^2] \in \mathbb{R}^3.$$

Then, given two vectors $\mathbf{x} = [x_1, x_2]^T$ and $\mathbf{y} = [y_1, y_2]^T$, it is straightforward to see that

$$\phi^T(\mathbf{x})\phi(\mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2.$$

That is, the inner product in the new space is given in terms of a function of the variables in the original space,

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^2.$$

**FIGURE 11.6**

The nonlinear task in the original low-dimensional space is mapped to a linear one in the high-dimensional RKHS \mathbb{H} . Using feature mapping, inner product operations are efficiently performed via kernel evaluations in the original low-dimensional spaces.

11.5.1 SOME PROPERTIES AND THEORETICAL HIGHLIGHTS

The reader who has no “mathematical anxieties” can bypass this subsection during a first reading.

Let \mathcal{X} be a set of points. Typically \mathcal{X} is a compact (closed and bounded) subset of \mathbb{R}^l . Let a function

$$\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}.$$

Definition 11.3. The function κ is called a *positive definite kernel*, if

$$\sum_{n=1}^N \sum_{m=1}^N a_n a_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \geq 0 : \quad \text{Positive Definite Kernel,} \quad (11.10)$$

for any real numbers, a_n, a_m , any points $\mathbf{x}_n, \mathbf{x}_m \in \mathcal{X}$ and any $N \in \mathbb{N}$.

Note that (11.10) can be written in an equivalent form. Define the so-called *kernel matrix*, \mathcal{K} , of order, N ,

$$\mathcal{K} := \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \quad (11.11)$$

Then, (11.10) is written as

$$\mathbf{a}^T \mathcal{K} \mathbf{a} \geq 0, \quad (11.12)$$

where

$$\mathbf{a} = [a_1, \dots, a_N]^T.$$

Because (11.10) is true for any $\mathbf{a} \in \mathbb{R}^N$, then (11.12) suggests that for a kernel to be positive definite, it suffices the corresponding kernel matrix to be positive semidefinite.⁴

Lemma 11.1. *The reproducing kernel, associated with an RKHS \mathbb{H} , is a positive definite kernel.*

The proof of the lemma is given in [Problem 11.3](#). Note that the opposite is also true. It can be shown, [83, 107], that if $\kappa : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a positive definite kernel, there exists an RKHS \mathbb{H} of functions on \mathcal{X} , such that $\kappa(\cdot, \cdot)$ is a reproducing kernel of \mathbb{H} . This establishes the equivalence between reproducing and positive definite kernels. Historically, the theory of positive definite kernels was developed first in the context of integral equations by Mercer [77], and the connection to RKHS was developed later on, see, for example, [2].

Lemma 11.2. *Let \mathbb{H} be an RKHS on the set \mathcal{X} with reproducing kernel $\kappa(\cdot, \cdot)$. Then the linear span of the function $\kappa(\cdot, \mathbf{x})$, $\mathbf{x} \in \mathcal{X}$ is dense in \mathbb{H} , that is,*

$$\mathbb{H} = \overline{\text{span}\{\kappa(\cdot, \mathbf{x}), \mathbf{x} \in \mathcal{X}\}}. \quad (11.13)$$

The proof of the lemma is given in [Problem 11.4](#). The overbar denotes the closure of a set. In other words, \mathbb{H} can be constructed by *all* possible linear combinations of the kernel function computed in \mathcal{X} , as well as the *limit points* of sequences of such combinations. Simply stated, \mathbb{H} can be *fully generated from the knowledge* of $\kappa(\cdot, \cdot)$.

The interested reader can obtain more theoretical results concerning RKH spaces from, for example, [64, 84, 89, 104, 107, 109].

11.5.2 EXAMPLES OF KERNEL FUNCTIONS

In this subsection, we present some typical examples of kernel functions, which are commonly used in various applications.

- The *Gaussian* kernel is among the most popular ones and it is given by our familiar form

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right),$$

with $\sigma > 0$ being a parameter. [Figure 11.7a](#) shows the Gaussian kernel as a function of $x, y \in \mathcal{X} = \mathbb{R}$ and $\sigma = 0.5$. [Figure 11.7b](#) shows $\phi(0) = \kappa(\cdot, 0)$ for various values of σ .

The dimension of the RKHS generated by the Gaussian kernel is *infinite*. A proof that the Gaussian kernel satisfies the required properties can be obtained, for example, from [109].

- The *homogeneous polynomial* kernel has the form

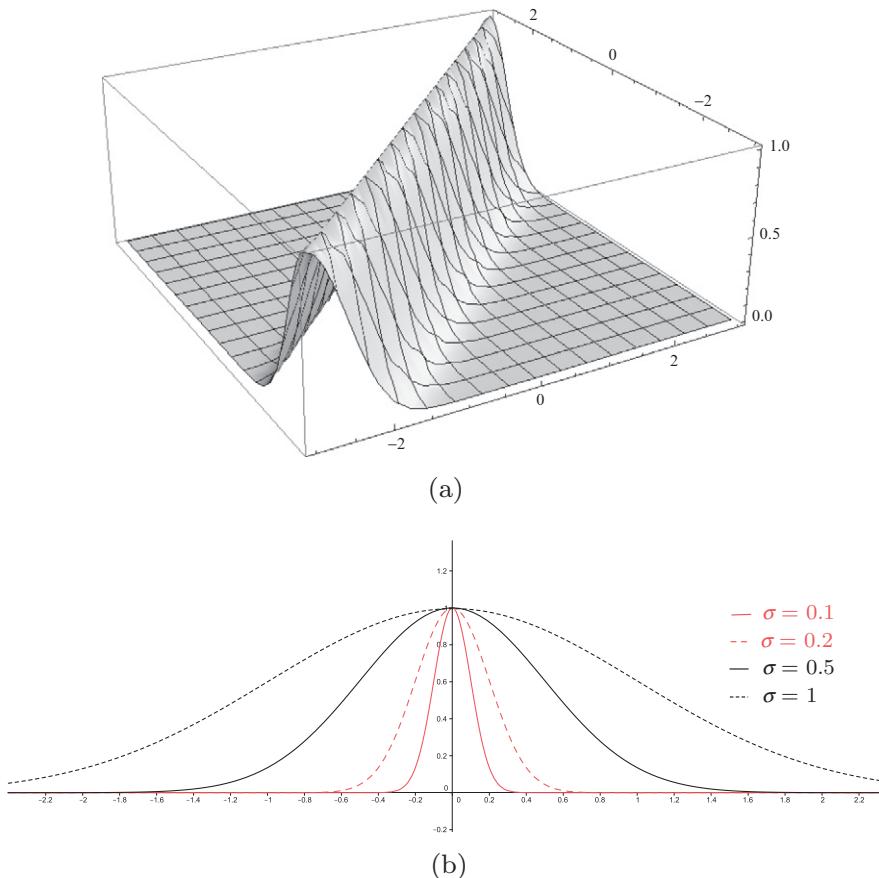
$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^r,$$

where r is a parameter.

- The *inhomogeneous polynomial* kernel is given by

$$\kappa(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^r,$$

⁴ It may be slightly confusing that the definition of a positive definite kernel requires a positive semidefinite kernel matrix. However, this is what has been the accepted definition.

**FIGURE 11.7**

(a) The Gaussian kernel for $\mathcal{X} = \mathbb{R}$, $\sigma = 0.5$. (b) The element $\phi(0) = \kappa(\cdot, 0)$ for different values of σ .

where $c \geq 0$ and r parameters. The graph of the kernel is given in Figure 11.8a. In Figure 11.8b the elements $\phi(\cdot, x_0)$ are shown for different values of x_0 . The dimensionality of the RKHS associated with polynomial kernels is finite.

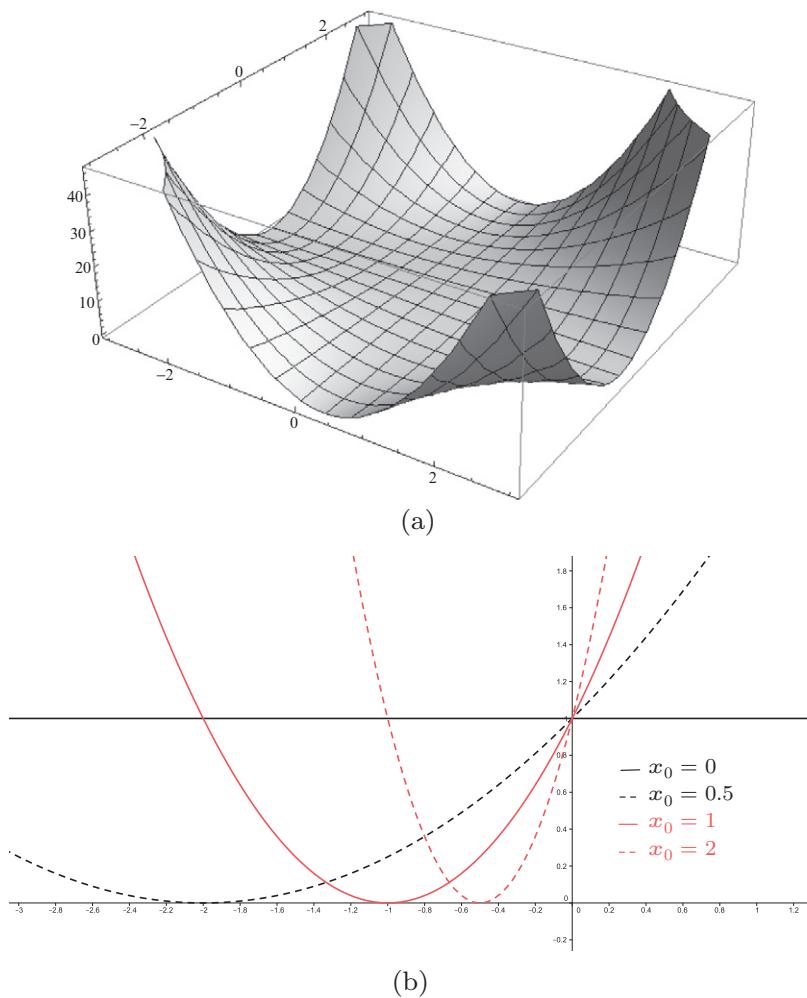
- The *Laplacian* kernel is given by

$$\kappa(x, y) = \exp(-t\|x - y\|),$$

where $t > 0$ is a parameter. The dimensionality of the RKHS associated with the Laplacian kernel is infinite.

- The *spline* kernels are defined as

$$\kappa(x, y) = B_{2p+1}(\|x - y\|^2),$$

**FIGURE 11.8**

(a) The inhomogeneous polynomial kernel for $\mathcal{X} = \mathbb{R}$, $r = 2$. (b) The element $\phi(x) = \kappa(\cdot, x_0)$, for different values of x_0 .

where the B_n spline is defined via the $n + 1$ convolutions of the unit interval $[-\frac{1}{2}, \frac{1}{2}]$, that is,

$$B_n(\cdot) := \bigotimes_{i=1}^{n+1} \chi_{[-\frac{1}{2}, \frac{1}{2}]}(\cdot)$$

and $\chi_{[-\frac{1}{2}, \frac{1}{2}]}(\cdot)$ is the characteristic function on the respective interval.⁵

⁵ It is equal to one if the variable belongs to the interval and zero otherwise.

- The *sampling function* or *sinc* kernel is of particular interest from a signal processing point of view. This kernel function is defined as

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}.$$

Recall that we have met this function in [Chapter 9](#) while discussing sub-Nyquist sampling.

Let us now consider the set of all squared integrable functions, which are *band limited*, that is,

$$\mathcal{F}_B = \left\{ f : \int_{-\infty}^{+\infty} |f(x)|^2 dx < +\infty, \text{ and } |F(\omega)| = 0, |\omega| > \pi \right\},$$

where $F(\omega)$ is the respective Fourier transform

$$F(\omega) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} f(x) e^{-j\omega x} dx.$$

It turns out that \mathcal{F}_B is an RKHS whose reproducing kernel is the sinc function (e.g., [\[50\]](#)), that is,

$$\kappa(x, y) = \text{sinc}(x - y).$$

This takes us back to the classical sampling theorem through the RKHS route. Without going into details, a by-product of this view is the Shannon's sampling theorem; any band limited function can be written as⁶

$$f(x) = \sum_n f(n) \text{sinc}(x - n). \quad (11.14)$$

Constructing kernels

Besides the previous examples, one can construct more kernels by applying the following properties ([Problem 11.6](#), [\[107\]](#)):

- If

$$\begin{aligned} \kappa_1(x, y) : \mathcal{X} \times \mathcal{X} &\mapsto \mathbb{R}, \\ \kappa_2(x, y) : \mathcal{X} \times \mathcal{X} &\mapsto \mathbb{R}, \end{aligned}$$

are kernels, then

$$\kappa(x, y) = \kappa_1(x, y) + \kappa_2(x, y),$$

and

$$\kappa(x, y) = \alpha \kappa_1(x, y), \quad \alpha > 0,$$

and

$$\kappa(x, y) = \kappa_1(x, y) \kappa_2(x, y),$$

are also kernels.

- Let

$$f : \mathcal{X} \mapsto \mathbb{R}.$$

⁶ The key point behind the proof is that in \mathcal{F}_B , the kernel $\kappa(x, y)$ can be decomposed in terms of a set of orthogonal functions, that is, $\text{sinc}(x - n)$, $n = 0, \pm 1, \pm 2, \dots$

Then

$$\kappa(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})f(\mathbf{y})$$

is a kernel.

- Let a function

$$g : \mathcal{X} \mapsto \mathbb{R}^l,$$

and a kernel function

$$\kappa_1(\cdot, \cdot) : \mathbb{R}^l \times \mathbb{R}^l \mapsto \mathbb{R}.$$

Then

$$\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(g(\mathbf{x}), g(\mathbf{y}))$$

is also a kernel.

- Let A be a positive definite $l \times l$ matrix. Then

$$\kappa(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T A \mathbf{y}$$

is a kernel.

- If

$$\kappa_1(\mathbf{x}, \mathbf{y}) : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R},$$

then

$$\kappa(\mathbf{x}, \mathbf{y}) = \exp(\kappa_1(\mathbf{x}, \mathbf{y}))$$

is also a kernel, and if $p(\cdot)$ is a polynomial with nonnegative coefficients,

$$\kappa(\mathbf{x}, \mathbf{y}) = p(\kappa_1(\mathbf{x}, \mathbf{y}))$$

is also a kernel.

The interested reader will find more information concerning kernels and their construction in, for example, [51, 107, 109].

String kernels

So far, our discussion has been focused on input data that were vectors in a Euclidean space. However, as we have already pointed out, the input data need not necessarily be vectors, and they can be elements of more general sets.

Let us denote by \mathcal{S} an alphabet set; that is, a set with a finite number of elements, which we call *symbols*. For example, this can be the set of all capital letters in the Latin alphabet. Bypassing the path of formal definitions, a *string* is a finite sequence, of any length, of symbols from \mathcal{S} . For example, two cases of strings are

$$T_1 = \text{"MYNAMEISSERGOS"}, \quad T_2 = \text{"HERNAMEISDESPOINA"}.$$

In a number of applications, such as in text mining, spam filtering, text summarization, and bioinformatics, it is important to quantify how “similar” two strings are. However, kernels, by their definition, are similarity measures; they are constructed so as to express inner products in the high-dimensional

feature space. An inner product is a similarity measure. Two vectors are most similar if they point to the same direction. Starting from this observation, there has been a lot of activity on defining kernels that measure similarity between strings. Without going into details, let us give such an example.

Let us denote by \mathcal{S}^* the set of all possible strings that can be constructed using symbols from \mathcal{S} . Also, a string, s , is said to be a *substring* of x if $x = bsa$, where a and b are other strings (possibly empty) from the symbols of \mathcal{S} . Given two strings $x, y \in \mathcal{S}^*$, define

$$\kappa(x, y) := \sum_{s \in \mathcal{S}^*} w_s \phi_s(x) \phi_s(y), \quad (11.15)$$

where, $w_s \geq 0$, and $\phi_s(x)$ is the number of times substring s appears in x . It turns out that this is indeed a kernel, in the sense that it complies with (11.10); such kernels constructed from strings are known as *string kernels*.

Obviously, a number of different variants of this kernel are available. The so-called *k-spectrum* kernel, considers common substrings only of length k . For example, for the two strings given before, the value of the 6-spectrum string kernel in (11.15) is equal to one (one common substring of length 6 is identified and appears once in each one of the two strings: “NAMEIS”). More on this topic, interested reader can obtain more on this topic from, for example, [107]. We will use the notion of the string kernel, in the case study in [Section 11.15](#).

11.6 REPRESENTER THEOREM

The theorem to be stated in this section is of major importance from a practical point of view. It allows us to perform *empirical* loss function optimization, based on a finite set of training points, in a very efficient way even if the function to be estimated belongs to a very high (even infinite) dimensional space, \mathbb{H} .

Theorem 11.2. *Let*

$$\Omega : [0, +\infty) \mapsto \mathbb{R}$$

be an arbitrary strictly monotonic increasing function. Let also

$$\mathcal{L} : \mathbb{R}^2 \mapsto \mathbb{R} \cup \{\infty\}$$

be an arbitrary loss function. Then each minimizer, $f \in \mathbb{H}$, of the regularized minimization task,

$$\min_{f \in \mathbb{H}} J(f) := \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) + \lambda \Omega(\|f\|^2) \quad (11.16)$$

admits a representation of the form,⁷

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n),$$

(11.17)

where $\theta_n \in \mathbb{R}$, $n = 1, 2, \dots, N$.

⁷ The property holds also for regularization of the form $\Omega(\|f\|)$, since the quadratic function is strictly monotonic on $[0, \infty)$, and the proof follows a similar line.

Proof. The linear span, $A := \text{span}\{\kappa(\cdot, \mathbf{x}_1), \dots, \kappa(\cdot, \mathbf{x}_N)\}$, forms a closed subspace. Then, each $f \in \mathbb{H}$ can be decomposed into two parts (see, (8.20)), that is,

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n) + f_{\perp},$$

where f_{\perp} is the part of f that is orthogonal to A . From the reproducing property, we obtain

$$\begin{aligned} f(\mathbf{x}_m) &= \langle f, \kappa(\cdot, \mathbf{x}_m) \rangle = \left\langle \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n), \kappa(\cdot, \mathbf{x}_m) \right\rangle \\ &= \sum_{n=1}^N \theta_n \kappa(\mathbf{x}_m, \mathbf{x}_n), \end{aligned}$$

where we used the fact that $\langle f_{\perp}, \kappa(\cdot, \mathbf{x}_n) \rangle = 0$, $n = 1, 2, \dots, N$. In other words, the expansion in (11.17) guarantees that at the training points, the value of f does not depend on f_{\perp} . Hence, the first term in (11.16), corresponding to the empirical loss, does *not* depend on f_{\perp} . Moreover, for all f_{\perp} we have

$$\begin{aligned} \Omega(\|f\|^2) &= \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2 + \|f_{\perp}\|^2\right) \\ &\geq \Omega\left(\left\|\sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n)\right\|^2\right). \end{aligned}$$

Thus, for *any* choice of θ_n , $n = 1, 2, \dots, N$, the cost function in (11.16) is minimized for $f_{\perp} = 0$. Thus, the claim is proved. \square

The theorem was first shown in [60]. In [1], the conditions under which the theorem exists were investigated and related sufficient and necessary conditions were derived. The importance of this theorem is that in order to optimize (11.16) with respect to f , one uses the expansion in (11.17) and minimization is carried out with respect to the *finite* set of parameters, θ_n , $n = 1, 2, \dots, N$.

Note that when working in high (even infinite) dimensional spaces, the presence of a regularizer can hardly be avoided; otherwise, the obtained solution will suffer from overfitting, as only a finite number of data samples are used for training. The effect of regularization on the generalization performance and stability of the associated solution has been studied in a number of classical papers, for example, [16, 37, 80, 92].

Usually, a bias term is often added and it is assumed that the minimizing function admits the following representation,

$$\tilde{f} = f + b, \tag{11.18}$$

$$f(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n). \tag{11.19}$$

In practice, the use of a bias term (which does not enter in the regularization) turns out to improve performance. First, it enlarges the class of functions in which the solution is searched and potentially leads to better performance. Moreover, due to the penalization imposed by the regularizing term,

$\Omega(\|f\|^2)$, the minimizer pushes the values, which the function takes at the training points, to smaller values. The existence of b tries to “absorb” some of this action; see, for example, [109].

Remarks 11.2.

- We will use the expansion in (11.17) in a number of cases. However, it is interesting to apply this expansion to the RKHS of the band limited functions and see what comes out. Assume that the available samples from a function f are $f(n)$, $n = 1, 2, \dots, N$ (assuming the case of normalized sampling period $x_s = 1$). Then according to the representer theorem, we can write the following approximation.

$$f(x) \approx \sum_{n=1}^N \theta_n \operatorname{sinc}(x - n). \quad (11.20)$$

Taking into account the orthonormality of the $\operatorname{sinc}(\cdot - n)$ functions, we get $\theta_n = f(n)$, $n = 1, 2, \dots, N$. However, note that in contrast to (11.14), which is exact, (11.20) is only an approximation. On the other hand, (11.20) can be used even if the obtained samples are contaminated by noise.

11.6.1 SEMIPARAMETRIC REPRESENTER THEOREM

The use of the bias term is also theoretically justified by the generalization of the representer theorem [104]. The essence of this theorem is to expand the solution into two parts. One that lies in an RKHS, \mathbb{H} , and another one that is given as a linear combination of a set of preselected functions.

Theorem 11.3. *Let us assume that in addition to the assumptions adopted in Theorem 11.2, we are given the set of real-valued functions*

$$\psi_m : \mathcal{X} \mapsto \mathbb{R}, \quad m = 1, 2, \dots, M,$$

with the property that the $N \times M$ matrix with elements $\psi_m(\mathbf{x}_n)$, $n = 1, 2, \dots, N$, $m = 1, 2, \dots, M$, has rank M . Then, any

$$\tilde{f} = f + h, \quad f \in \mathbb{H}, \quad h \in \operatorname{span}\{\psi_m, m = 1, 2, \dots, M\},$$

solving the minimization task

$$\min_{\tilde{f}} J(\tilde{f}) := \sum_{n=1}^N \mathcal{L}(y_n, \tilde{f}(\mathbf{x}_n)) + \Omega(\|f\|^2), \quad (11.21)$$

admits the following representation:

$$\tilde{f}(\cdot) = \sum_{n=1}^N \theta_n \kappa(\cdot, \mathbf{x}_n) + \sum_{m=1}^M b_m \psi_m(\cdot).$$

(11.22)

Obviously, the use of a bias term is a special case of the expansion above. An example of successful application of this theorem was demonstrated in [13] in the context of image de-noising. A set of nonlinear functions in place of ψ_m were used to account for the edges (nonsmooth jumps) in an image. The part of f lying in the RKHS accounted for the smooth parts in the image.

11.6.2 NONPARAMETRIC MODELING: A DISCUSSION

Note that searching a model function in an RKHS space is a typical task of *nonparametric* modeling. In contrast to the parametric modeling in Eq. (11.1), where the unknown function is *parameterized* in terms of a set of basis functions, the minimization in (11.16) or (11.21) is performed with regard to functions that are *constrained to belong in a specific space*. In the more general case, minimization could be performed with regard to any (continuous) function, for example,

$$\min_f \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) + \lambda \phi(f),$$

where $\mathcal{L}(\cdot, \cdot)$ can be any loss function and ϕ an appropriately chosen regularizing functional. Note, however, that in this case, the presence of the regularization is crucial. If there is no regularization, then any function that *interpolates* the data is a solution; such techniques have also been used in interpolation theory, for example, [79, 93]. The regularization term, $\phi(f)$, helps to smooth out the function to be recovered. To this end, functions of derivatives have been employed. For example, if the minimization cost is chosen as

$$\sum_{n=1}^N (y_n - f(x_n))^2 + \lambda \int (f''(x))^2 dx,$$

then the solution is a cubic spline; that is, a piecewise cubic function with knots the points x_n , $n = 1, 2, \dots, N$ and it is continuously differentiable to the second order. The choice of λ controls the degree of smoothness of the approximating function; the larger its value the smoother the minimizer becomes.

If on the other hand, f is constrained to lie in an RKHS and the minimizing task is as in (11.16), then the resulting function is of the form given in (11.17), where a kernel function is placed at each input training point. It must be pointed out that the parametric form that now results was not in our original intentions. It came out as a by-product of the theory. However, it should be stressed that, in contrast to the parametric methods, now the number of parameters to be estimated is not fixed but it depends on the number of the training points. Recall that this is an important difference and it was carefully pointed out when parametric methods were introduced and defined in Chapter 3.

11.7 KERNEL RIDGE REGRESSION

Ridge regression was introduced in Chapter 3 and it has also been treated in more detail in Chapter 6. Here, we will state the task in a general RKHS. The path to be followed is the typical one used to extend techniques, which have been developed for linear models, to the more general RKH spaces.

We assume that the generation mechanism of the data, represented by the training set $(y_n, \mathbf{x}_n) \in \mathbb{R} \times \mathbb{R}^l$, is modeled via a nonlinear regression task

$$y_n = g(\mathbf{x}_n) + \eta_n, \quad n = 1, 2, \dots, N. \quad (11.23)$$

Let us denote by f the estimate of the unknown g . Sometimes, f is called the *hypothesis* and the space \mathbb{H} in which f is searched is known as the *hypothesis space*. We will further assume that f lies in an RKHS, associated with a kernel

$$\kappa : \mathbb{R}^l \times \mathbb{R}^l \mapsto \mathbb{R}.$$

Motivated by the representer theorem, we adopt the following expansion

$$f(\mathbf{x}) = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}, \mathbf{x}_n).$$

According to the kernel ridge regression approach, the unknown coefficients are estimated by the following task

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \arg \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}), \\ J(\boldsymbol{\theta}) &:= \sum_{n=1}^N \left(y_n - \sum_{m=1}^N \theta_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \right)^2 + C \langle f, f \rangle, \end{aligned} \quad (11.24)$$

where C is the regularization parameter.⁸ Equation (11.24) can be rewritten as (Problem 11.7)

$$J(\boldsymbol{\theta}) = (\mathbf{y} - \mathcal{K}\boldsymbol{\theta})^T (\mathbf{y} - \mathcal{K}\boldsymbol{\theta}) + C\boldsymbol{\theta}^T \mathcal{K}^T \boldsymbol{\theta}, \quad (11.25)$$

where

$$\mathbf{y} = [y_1, \dots, y_N]^T, \quad \boldsymbol{\theta} = [\theta_1, \dots, \theta_N]^T,$$

and \mathcal{K} is the kernel matrix defined in (11.11); the latter is fully determined by the kernel function and the training points. Following our familiar-by-now arguments, minimization of $J(\boldsymbol{\theta})$ with regard to $\boldsymbol{\theta}$ leads to

$$(\mathcal{K}^T \mathcal{K} + C\mathcal{K}^T) \hat{\boldsymbol{\theta}} = \mathcal{K}^T \mathbf{y}$$

or

$$(\mathcal{K} + CI)\hat{\boldsymbol{\theta}} = \mathbf{y} : \quad \text{Kernel Ridge Regression,} \quad (11.26)$$

where $\mathcal{K}^T = \mathcal{K}$ has been assumed to be invertible.⁹ Once $\hat{\boldsymbol{\theta}}$ has been obtained, given an unknown vector, $\mathbf{x} \in \mathbb{R}^l$, the corresponding prediction value of the dependent variable is given by

$$\hat{y} = \sum_{n=1}^N \hat{\theta}_n \kappa(\mathbf{x}, \mathbf{x}_n) = \hat{\boldsymbol{\theta}}^T \kappa(\mathbf{x}),$$

where

$$\kappa(\mathbf{x}) = [\kappa(\mathbf{x}, \mathbf{x}_1), \dots, \kappa(\mathbf{x}, \mathbf{x}_N)]^T.$$

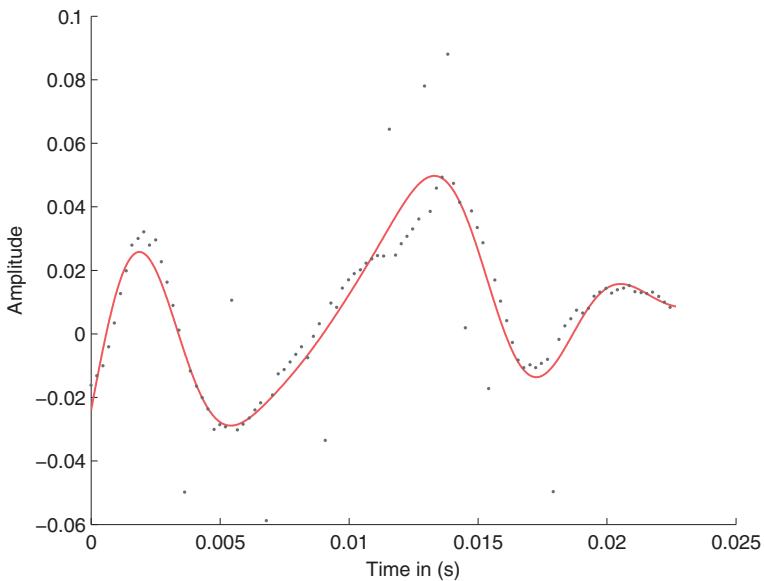
Employing (11.26), we obtain

$$\hat{y}(\mathbf{x}) = \mathbf{y}^T (\mathcal{K} + CI)^{-1} \kappa(\mathbf{x}). \quad (11.27)$$

Example 11.2. In this example, the prediction power of the kernel ridge regression in the presence of noise and outliers will be tested. The original data were samples from a music recording of *Blade Runner* by Vangelis Papathanasiou. A white Gaussian noise was then added at a 15 dB level and a

⁸ For the needs of this chapter, we denote the regularization constant as C , not to be confused with the Lagrange multipliers, to be introduced soon.

⁹ This is true, for example, for the Gaussian kernel, [104].

**FIGURE 11.9**

Plot of the data used for training together with the fitted (prediction) curve obtained via the kernel ridge regression, for Example 11.2. The Gaussian kernel was used.

number of outliers were intentionally randomly introduced and “hit” some of the values (10%, of them). The kernel ridge regression method was used, employing the Gaussian kernel with $\sigma = 0.004$. We allowed for a bias term to be present (see Problem 11.8). The prediction (fitted) curve, $\hat{y}(x)$, for various values of x , is shown in Figure 11.9 together with the (noisy) data used for training.

11.8 SUPPORT VECTOR REGRESSION

The least-squares cost function is not always the best criterion for optimization, in spite of its merits. In the case of the presence of a non-Gaussian noise with long tails and, hence, with an increased number of noise *outliers*, the square dependence of the LS criterion gets biased toward values associated with the presence of outliers. Recall from Chapter 3 that the method of least-squares is equivalent to the maximum likelihood estimation under the assumption of white Gaussian noise. Moreover, under this assumption, the LS estimator achieves the Cramer-Rao bound and it becomes a minimum variance estimator. However, under other noise scenarios, one has to look for alternative criteria.

The task of optimization in the presence of outliers was studied by Huber [53], whose goal was to obtain a strategy for choosing the loss function that “matches” best to the noise model. He proved that, under the assumption that the noise has a symmetric pdf, the optimal minimax strategy for regression is obtained via the following loss function,

$$\mathcal{L}(y, f(x)) = |y - f(x)|,$$

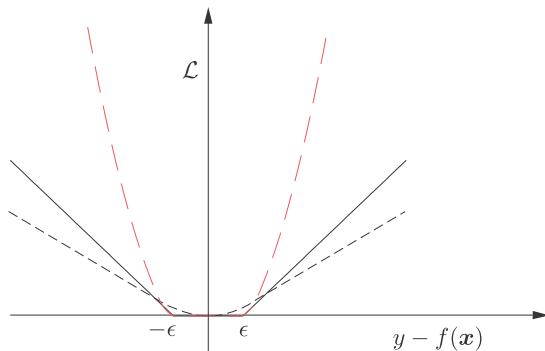


FIGURE 11.10

The Huber loss function (dotted-gray), the linear ϵ -insensitive (full-gray), and the quadratic ϵ -insensitive (red) loss functions, for $\epsilon = 0.7$.

which is known as the *least modulus* method. Note from [Section 5.8](#) that the stochastic gradient online version for this loss function leads to the sign-error LMS. Huber also showed that if the noise comprises two components, one corresponding to a Gaussian and another to an arbitrary pdf (which remains symmetric), then the best in the minimax sense loss function is given by

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} \epsilon|y - f(\mathbf{x})| - \frac{\epsilon^2}{2}, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ \frac{1}{2}|y - f(\mathbf{x})|^2, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases}$$

for some parameter ϵ . This is known as the Huber loss function and it is shown in [Figure 11.10](#). A loss function that can approximate the Huber one and, as we will see, turns out to have some nice computational properties, is the so-called *linear ϵ -insensitive* loss function, defined as (see also [Chapter 8](#))

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} |y - f(\mathbf{x})| - \epsilon, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ 0, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases} \quad (11.28)$$

and it is shown in [Figure 11.10](#). Note that for $\epsilon = 0$, it coincides with the least absolute loss function, and it is close to the Huber loss for small values of $\epsilon < 1$. Another version is the *quadratic ϵ -insensitive* defined as

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} |y - f(\mathbf{x})|^2 - \epsilon, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ 0, & \text{if } |y - f(\mathbf{x})| \leq \epsilon, \end{cases} \quad (11.29)$$

which coincides with the LS loss for $\epsilon = 0$. The corresponding graph is given in [Figure 11.10](#). Observe that the two previously discussed ϵ -insensitive loss functions retain their convex nature; however, they are no more differentiable at all points.

11.8.1 THE LINEAR ϵ -INSENSITIVE OPTIMAL REGRESSION

Let us now adopt (11.28) as the loss function to quantify model misfit. We will treat the regression task in (11.23), employing a linear model for f , that is

$$f(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x} + \theta_0.$$

Once we obtain the solution expressed in inner product operations, the more general solution for the case where f lies in an RKHS will be obtained via the kernel trick; that is, inner products will be replaced by kernel evaluations.

Let us now introduce two sets of *auxiliary* variables. If

$$y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \geq \epsilon,$$

define $\tilde{\xi}_n \geq 0$, such as

$$y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n.$$

Note that ideally, we would like to select $\boldsymbol{\theta}, \theta_0$, so that $\tilde{\xi}_n = 0$, because this would make the contribution of the respective term in the loss function equal to zero. Also, if

$$y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \leq -\epsilon,$$

define $\xi_n \geq 0$, such as

$$\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n.$$

Once more, we would like to select our unknown set of parameters so that ξ_n is zero.

We are now ready to formulate the minimizing task around the corresponding empirical cost, regularized by the norm of $\boldsymbol{\theta}$, which is cast in terms of the auxiliary variables as¹⁰

$$\text{minimize} \quad J(\boldsymbol{\theta}, \theta_0, \boldsymbol{\xi}, \tilde{\boldsymbol{\xi}}) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \left(\sum_{n=1}^N \xi_n + \sum_{n=1}^N \tilde{\xi}_n \right), \quad (11.30)$$

$$\text{subject to} \quad y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 \leq \epsilon + \tilde{\xi}_n, \quad n = 1, 2, \dots, N, \quad (11.31)$$

$$\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n \leq \epsilon + \xi_n, \quad n = 1, 2, \dots, N, \quad (11.32)$$

$$\tilde{\xi}_n \geq 0, \quad \xi_n \geq 0, \quad n = 1, 2, \dots, N. \quad (11.33)$$

Before we proceed further some explanations are in order.

- The auxiliary variables, $\tilde{\xi}_n$ and ξ_n , $n = 1, 2, \dots, N$, which measure the excess error with regard to ϵ , are known as *slack variables*. Note that according to the ϵ -insensitive rationale, any contribution to the cost function of an error with absolute value less than or equal to ϵ is zero. The previous optimization task attempts to estimate $\boldsymbol{\theta}, \theta_0$ so that the number of error values larger than ϵ and smaller than $-\epsilon$ is minimized. Thus, the optimization task in (11.30)–(11.33) is equivalent with minimizing the empirical loss function

$$\frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \mathcal{L}(y_n, \boldsymbol{\theta}^T \mathbf{x}_n + \theta_0),$$

where the loss function is the linear ϵ -insensitive one. Note that, any other method for minimizing (nondifferentiable) convex functions could be used, for example, Chapter 8. However, the constrained optimization involving the slack variables has a historical value and it was the path that paved the way in employing the kernel trick, as we will see soon.

¹⁰ It is common in the literature to formulate the regularized cost via the parameter C multiplying the loss term and not $\|\boldsymbol{\theta}\|^2$. In any case, they are both equivalent.

- As said before, the task could be cast directly as a linear one in an RKHS. In such a case, the path to follow is to assume a mapping

$$\mathbf{x} \mapsto \phi(\mathbf{x}) = \kappa(\cdot, \mathbf{x}),$$

for some kernel, and then approximate the nonlinear function, $g(\mathbf{x})$, in the regression task as a linear one in the respective RKHS, that is,

$$g(\mathbf{x}) \approx f(\mathbf{x}) = \langle \theta, \phi(\mathbf{x}) \rangle + \theta_0,$$

where θ is now treated as a function in the RKHS. However, in this case, in order to solve the minimizing task we should consider differentiation with regard to functions. Although the rules are similar to those of differentiation with respect to variables, because we have not given such definitions we have avoided following this path (for the time being).

The solution

The solution of the optimization task is obtained by introducing Lagrange multipliers and forming the corresponding Lagrangian (see below for the detailed derivation). Having obtained the Lagrange multipliers, the solution turns out to be given in a simple and rather elegant form,

$$\hat{\theta} = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \mathbf{x}_n,$$

where $\tilde{\lambda}_n, \lambda_n, n = 1, 2, \dots, N$, are the Lagrange multiplies associated with each one of the constraints. It turns out that the Lagrange multipliers are nonzero *only* for those points, \mathbf{x}_n , that correspond to error values *either equal or larger* than ϵ . These are known as *support vectors*. Points that score error values *less* than ϵ correspond to zero Lagrange multipliers and do not participate in the formation of the solution. The bias term can be obtained by anyone from the set of equations

$$y_n - \theta^T \mathbf{x}_n - \theta_0 = \epsilon, \quad (11.34)$$

$$\theta^T \mathbf{x}_n + \theta_0 - y_n = \epsilon, \quad (11.35)$$

where n above runs over the points that are associated with $\tilde{\lambda}_n > 0$ ($\lambda_n > 0$) and $\tilde{\xi}_n = 0$ ($\xi_n = 0$) (note that these points form a subset of the support vectors). In practice, $\hat{\theta}_0$ is obtained as the average from all the previous equations.

For the more general setting of the task in an RKHS, we can write

$$\hat{\theta}(\cdot) = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \kappa(\cdot, \mathbf{x}_n).$$

Once $\hat{\theta}$, $\hat{\theta}_0$ have been obtained, we are ready to perform prediction. Given a value \mathbf{x} , we first perform the (implicit) mapping using the feature map

$$\mathbf{x} \mapsto \kappa(\cdot, \mathbf{x}),$$

and we get

$$\hat{y}(\mathbf{x}) = \langle \hat{\theta}, \kappa(\cdot, \mathbf{x}) \rangle + \hat{\theta}_0$$

or

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} (\tilde{\lambda}_n - \lambda_n) \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0 : \quad \text{SVR Prediction,} \quad (11.36)$$

where $N_s \leq N$, is the number of nonzero Lagrange multipliers. Observe that (11.36) is an expansion in terms of nonlinear (kernel) functions. Moreover, as only a fraction of the points is involved (N_s), the use of the ϵ -insensitive loss function achieves a form of *sparsification* on the general expansion dictated by the representer theorem in (11.17) or (11.18).

Solving the optimization task

The reader who is not interested in proofs can bypass this part in a first reading.

The task in (11.30)–(11.33) is a *convex programming* minimization, with a set of linear inequality constraints. As it is discussed in [Appendix C](#), a minimizer has to satisfy the following *Karush-Kuhn-Tucker* conditions,

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0}, \quad \frac{\partial L}{\partial \theta_0} = 0, \quad \frac{\partial L}{\partial \tilde{\xi}_n} = 0, \quad \frac{\partial L}{\partial \xi_n} = 0, \quad (11.37)$$

$$\tilde{\lambda}_n(y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 - \epsilon - \tilde{\xi}_n) = 0, \quad n = 1, 2, \dots, N, \quad (11.38)$$

$$\lambda_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n - \epsilon - \xi_n) = 0, \quad n = 1, 2, \dots, N, \quad (11.39)$$

$$\tilde{\mu}_n \tilde{\xi}_n = 0, \quad \mu_n \xi_n = 0, \quad n = 1, 2, \dots, N, \quad (11.40)$$

$$\tilde{\lambda}_n \geq 0, \quad \lambda_n \geq 0, \quad \tilde{\mu}_n \geq 0, \quad \mu_n \geq 0, \quad n = 1, 2, \dots, N, \quad (11.41)$$

where L is the respective Lagrangian

$$\begin{aligned} L(\boldsymbol{\theta}, \theta_0, \tilde{\xi}, \xi, \lambda, \mu) &= \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \left(\sum_{n=1}^N \xi_n + \sum_{n=1}^N \tilde{\xi}_n \right) \\ &\quad + \sum_{n=1}^N \tilde{\lambda}_n (y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \theta_0 - \epsilon - \tilde{\xi}_n) \\ &\quad + \sum_{n=1}^N \lambda_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n - \epsilon - \xi_n) \\ &\quad - \sum_{n=1}^N \tilde{\mu}_n \tilde{\xi}_n - \sum_{n=1}^N \mu_n \xi_n, \end{aligned} \quad (11.42)$$

where $\tilde{\lambda}_n, \lambda_n, \tilde{\mu}_n, \mu_n$ are the corresponding Lagrange multipliers. A close observation of (11.38) and (11.39) reveals that (Why?)

$$\tilde{\xi}_n \xi_n = 0, \quad \tilde{\lambda}_n \lambda_n = 0, \quad n = 1, 2, \dots, N. \quad (11.43)$$

Taking the derivatives of the Lagrangian in (11.37) and equating to zero results in

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0} \longrightarrow \hat{\boldsymbol{\theta}} = \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) \mathbf{x}_n, \quad (11.44)$$

$$\frac{\partial L}{\partial \theta_0} = 0 \rightarrow \sum_{n=1}^N \tilde{\lambda}_n = \sum_{n=1}^N \lambda_n, \quad (11.45)$$

$$\frac{\partial L}{\partial \tilde{\xi}_n} = 0 \rightarrow C - \tilde{\lambda}_n - \tilde{\mu}_n = 0, \quad (11.46)$$

$$\frac{\partial L}{\partial \xi_n} = 0 \rightarrow C - \lambda_n - \mu_n = 0. \quad (11.47)$$

Note that all one needs in order to obtain $\hat{\theta}$ are the values of the Lagrange multipliers. As discussed in [Appendix C](#), these can be obtained by writing the problem in its dual representation form, that is,

$$\begin{aligned} \text{maximize with respect to } \lambda, \tilde{\lambda} & \quad \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) y_n - \epsilon (\tilde{\lambda}_n + \lambda_n), \\ & \quad -\frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N (\tilde{\lambda}_n - \lambda_n)(\tilde{\lambda}_m - \lambda_m) \mathbf{x}_n^T \mathbf{x}_m, \end{aligned} \quad (11.48)$$

$$\text{subject to} \quad 0 \leq \tilde{\lambda}_n \leq C, 0 \leq \lambda_n \leq C, n = 1, 2, \dots, N, \quad (11.49)$$

$$\sum_{n=1}^N \tilde{\lambda}_n = \sum_{n=1}^N \lambda_n. \quad (11.50)$$

Concerning the maximization task in [\(11.48\)](#)–[\(11.50\)](#), the following comments are in order.

- [\(11.48\)](#) results by plugging into the Lagrangian the estimate obtained in [\(11.44\)](#) and following the steps as required by the dual representation form, ([Problem 11.10](#)).
- [\(11.49\)](#) results from [\(11.46\)](#) and [\(11.47\)](#) taking into account that $\mu_n \geq 0, \tilde{\mu}_n \geq 0$.
- The beauty of the dual representation form is that it involves the observation vectors in the form of inner product operations. Thus, when the task is solved in an RKHS, [\(11.48\)](#) becomes

$$\begin{aligned} \text{maximize with respect to } \lambda, \tilde{\lambda} & \quad \sum_{n=1}^N (\tilde{\lambda}_n - \lambda_n) y_n - \epsilon (\tilde{\lambda}_n + \lambda_n) \\ & \quad -\frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N (\tilde{\lambda}_n - \lambda_n)(\tilde{\lambda}_m - \lambda_m) \kappa(\mathbf{x}_n, \mathbf{x}_m). \end{aligned}$$

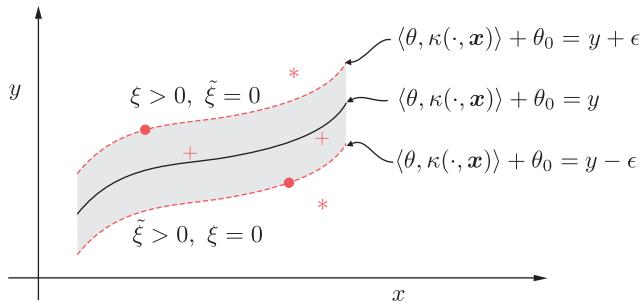
- The KKT conditions convey important information. The Lagrange multipliers, $\tilde{\lambda}_n, \lambda_n$, for points that score error less than ϵ , that is,

$$|\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 - y_n| < \epsilon,$$

are zero. This is a direct consequence of [\(11.38\)](#) and [\(11.39\)](#) and the fact that $\tilde{\xi}_n, \xi_n \geq 0$. Thus, the Lagrange multipliers are *nonzero* only for points which score error either equal to ϵ ($\tilde{\xi}_n, \xi_n = 0$) or larger values ($\tilde{\xi}_n, \xi_n > 0$). In other words, only the points with nonzero Lagrange multipliers (support vectors) enter in [\(11.44\)](#) which leads to a *sparsification* of the expansion in [\(11.44\)](#).

- Due to [\(11.43\)](#), either $\tilde{\xi}_n$ or ξ_n can be nonzero, but not both of them. This also applies to the corresponding Lagrange multipliers.
- Note that if $\tilde{\xi}_n > 0$ (or $\xi_n > 0$) then from [\(11.40\)](#), [\(11.46\)](#), and [\(11.47\)](#) we obtain that

$$\tilde{\lambda}_n = C \text{ or } \lambda_n = C.$$

**FIGURE 11.11**

The tube around the nonlinear regression curve. Points outside the tube (denoted by stars) have either $\tilde{\xi} > 0$ and $\xi = 0$ or $\xi > 0$ and $\tilde{\xi} = 0$. The rest of the points have $\tilde{\xi} = \xi = 0$. Points that are inside the tube correspond to zero Lagrange multipliers.

That is, the respective Lagrange multipliers get their maximum value. In other words, they have a “big say” in the expansion in (11.44). When $\tilde{\xi}_n$ and/or ξ_n are zero, then

$$0 \leq \tilde{\lambda}_n \leq C, \quad 0 \leq \lambda_n \leq C.$$

- Recall what we have said before concerning the estimation of θ_0 . Select any point corresponding to $0 < \tilde{\lambda}_n < C$, $(0 < \lambda_n < C)$, which we know correspond to $\tilde{\xi}_n = 0$ ($\xi_n = 0$). Then $\hat{\theta}_0$ is computed from (11.38) and (11.39). In practice, one selects all such points and computes θ_0 as the respective mean.
- Figure 11.11 illustrates $\hat{y}(x)$ for a choice of $\kappa(\cdot, \cdot)$. Observe that the value of ϵ forms a “tube” around the respective graph. Points lying outside the tube correspond to values of the slack variables larger than zero.

Remarks 11.3.

- Besides the linear ϵ -insensitive loss, similar analysis is valid for the quadratic ϵ -insensitive and Huber loss functions, for example, [27, 127]. It turns out that using the Huber loss function results in a larger number of support vectors. Note that a large number of support vectors increases complexity, as more kernel evaluations are involved.
- *Sparsity and ϵ -insensitive loss function:* Note that (11.36) is exactly the same form as (11.18). However, in the former case, the expansion is a sparse one using $N_s < N$, and in practice often $N_s \ll N$. The obvious question that is now raised is whether there is a “hidden” connection between the ϵ -insensitive loss function and the sparsity-promoting methods, discussed in Chapter 9. Interestingly enough, the answer is in the affirmative [46]. Assuming the unknown function, g , in (11.23) to reside in an RKHS, and exploiting the representer theorem, it is approximated by an expansion in an RKHS and the unknown parameters are estimated by minimizing

$$L(\theta) = \frac{1}{2} \|y(\cdot) - \sum_{n=1}^N \theta_n \kappa(\cdot, x_n)\|_{\mathbb{H}}^2 + \epsilon \sum_{n=1}^N |\theta_n|.$$

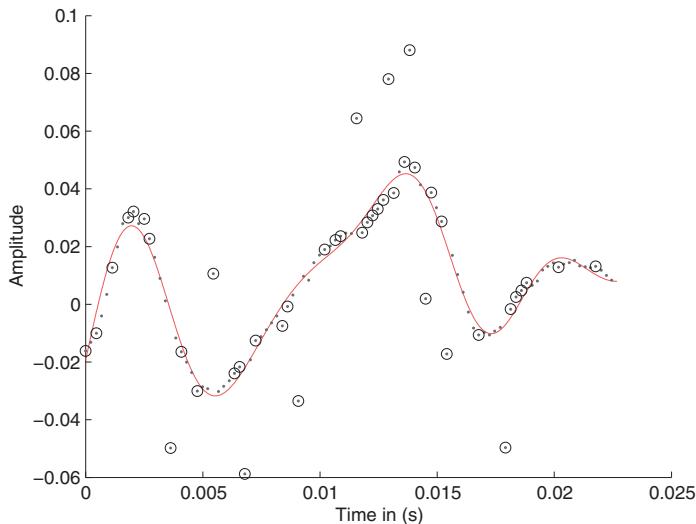


FIGURE 11.12

The resulting prediction curve for the same data points as those used for [Example 11.2](#). The improved performance compared to the kernel ridge regression used for [Figure 11.9](#) is readily observed. The encircled points are the support vectors resulting from the optimization, using the ϵ -insensitive loss function.

This is similar to what we did for the kernel ridge regression with the notable exception that the ℓ_1 norm of the parameters is involved for regularization. The norm $\|\cdot\|_{\mathbb{H}}$ denotes the norm associated with the RKHS. Elaborating on the norm, it can be shown that for the noiseless case, the minimization task becomes identical with the SVR one.

Example 11.3. Consider the same time series used for the nonlinear prediction task in [Example 11.2](#). This time, the SVR method was used optimized around the linear ϵ -insensitive loss function, with $\epsilon = 0.003$. The same Gaussian kernel with $\sigma = 0.004$ was employed as in the kernel ridge regression (KRR) case. [Figure 11.12](#) shows the resulting prediction curve, $\hat{y}(x)$ as a function of x given in [\(11.36\)](#). The encircled points are the support vectors. Even without the use of any quantitative measure, the resulting curve fits the data samples much better compared to the kernel ridge regression, exhibiting the enhanced robustness of the SVR method relative to the KRR, in the presence of outliers.

Remarks 11.4.

- A more recent trend to deal with outliers is via their explicit modeling. The noise is split into two components, the inlier and the outlier. The outlier part has to be spare; otherwise, it would not be called outlier. Then, sparsity-related arguments are mobilized to solve an optimization task that estimates both the parameters as well as the outliers; see, for example, [\[15, 72, 81, 86, 87\]](#).

11.9 KERNEL RIDGE REGRESSION REVISITED

The kernel ridge regression was introduced in [Section 11.7](#). Here, it will be restated via its dual representation form. The ridge regression in its primal representation can be cast as

$$\begin{aligned} \text{minimize with respect to } \boldsymbol{\theta}, \boldsymbol{\xi} & \quad J(\boldsymbol{\theta}, \boldsymbol{\xi}) = \sum_{n=1}^N \xi_n^2 + C\|\boldsymbol{\theta}\|^2, \\ \text{subject to} & \quad y_n - \boldsymbol{\theta}^T \mathbf{x}_n = \xi_n, \quad n = 1, 2, \dots, N, \end{aligned} \quad (11.51)$$

which leads to the following Lagrangian:

$$L(\boldsymbol{\theta}, \boldsymbol{\xi}, \boldsymbol{\lambda}) = \sum_{n=1}^N \xi_n^2 + C\|\boldsymbol{\theta}\|^2 + \sum_{n=1}^N \lambda_n(y_n - \boldsymbol{\theta}^T \mathbf{x}_n - \xi_n), \quad n = 1, 2, \dots, N. \quad (11.52)$$

Differentiating with respect to $\boldsymbol{\theta}$ and ξ_n , $n = 1, 2, \dots, N$, and equating to zero, we obtain

$$\boldsymbol{\theta} = \frac{1}{2C} \sum_{n=1}^N \lambda_n \mathbf{x}_n \quad (11.53)$$

and

$$\xi_n = \frac{\lambda_n}{2}, \quad n = 1, 2, \dots, N. \quad (11.54)$$

To obtain the Lagrange multipliers, (11.53) and (11.54) are substituted in (11.52) which results in the dual formulation of the problem, that is,

$$\begin{aligned} \text{maximize with respect to } \boldsymbol{\lambda} & \quad \sum_{n=1}^N \lambda_n y_n - \frac{1}{4C} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m \kappa(\mathbf{x}_n, \mathbf{x}_m) \\ & \quad - \frac{1}{4} \sum_{n=1}^N \lambda_n^2, \end{aligned} \quad (11.55)$$

where we have replaced $\mathbf{x}_n^T \mathbf{x}_m$ with the kernel operation according to the kernel trick. It is a matter of straightforward algebra to obtain ([99], Problem 11.9)

$$\boldsymbol{\lambda} = 2C(\mathcal{K} + CI)^{-1} \mathbf{y}, \quad (11.56)$$

which combined with (11.53) and involving the kernel trick we obtain the prediction rule for the kernel ridge regression, that is,

$$\hat{\mathbf{y}}(\mathbf{x}) = \mathbf{y}^T (\mathcal{K} + CI)^{-1} \boldsymbol{\kappa}(\mathbf{x}), \quad (11.57)$$

which is the same as (11.27); however, via this path one needs not to assume invertibility of \mathcal{K} . An efficient scheme for solving the kernel ridge regression has been developed in [119, 120].

11.10 OPTIMAL MARGIN CLASSIFICATION: SUPPORT VECTOR MACHINES

The optimal classifier, in the sense of minimizing the misclassification error, is the Bayesian classifier as discussed in Chapter 7. The method, being a member of the generative learning family, requires the knowledge of the underlying statistics. If this is not known, an alternative path is to resort to discriminative learning techniques and adopt a discriminant function, f that realizes the corresponding classifier and try to optimize it so as to minimize the respective empirical loss, that is,

$$J(f) = \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)),$$

where

$$y_n = \begin{cases} +1, & \text{if } \mathbf{x}_n \in \omega_1, \\ -1, & \text{if } \mathbf{x}_n \in \omega_2. \end{cases}$$

For a binary classification task, the first loss function that comes to mind is

$$\mathcal{L}(y, f(\mathbf{x})) = \begin{cases} 1, & \text{if } yf(\mathbf{x}) \leq 0, \\ 0, & \text{otherwise,} \end{cases} \quad (11.58)$$

which is also known as the $(0, 1)$ -loss function. However, this is a discontinuous function and its optimization is a hard task. To this end, a number of alternative loss functions have been adopted in an effort to approximate the $(0, 1)$ -loss function. Recall that the LS loss can also be employed but, as already pointed out in [Chapters 3 and 7](#) this is not well-suited for classification tasks and bears little resemblance with the $(0, 1)$ -loss function. In this section, we turn our attention to the so called *hinge* loss function defined as ([Chapter 8](#))

$$\mathcal{L}_\rho(y, f(\mathbf{x})) = \max \{0, \rho - yf(\mathbf{x})\}. \quad (11.59)$$

In other words, if the sign of the product between the true label (y) and that predicted by the discriminant function value ($f(\mathbf{x})$) is positive and larger than a threshold/margin (user-defined) value $\rho \geq 0$, the loss is zero. If not, the loss exhibits a linear increase. We say that a margin error is committed if $yf(\mathbf{x})$ cannot achieve a value of at least ρ . The hinge loss function is shown in [Figure 11.13](#), together with $(0, 1)$ and squared error loss functions.

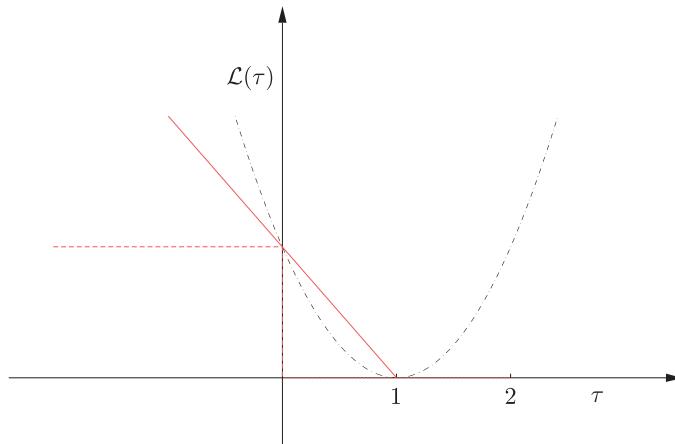


FIGURE 11.13

The $(0, 1)$ -loss (dotted red), the hinge loss (red), and the squared error (dotted black) functions tuned to pass through the $(0, 1)$ point for comparison. For the hinge loss, $\rho = 1$ $\tau = yf(\mathbf{x})$ for the hinge and $(0, 1)$ loss functions and $\tau = y - f(\mathbf{x})$ for the squared error one.

We will constrain ourselves to linear discriminant functions, residing in some RKHS, of the form

$$f(\mathbf{x}) = \theta_0 + \langle \boldsymbol{\theta}, \phi(\mathbf{x}) \rangle,$$

where

$$\phi(\mathbf{x}) = \kappa(\cdot, \mathbf{x})$$

is the feature map. However, for the same reasons discussed in [Section 11.8.1](#), we will cast the task as a linear one in the input space, \mathbb{R}^J , and at the final stage the kernel information will be “implanted” using the kernel trick.

The goal of designing a linear classifier now becomes equivalent with minimizing the cost

$$J(\boldsymbol{\theta}, \theta_0) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \mathcal{L}_\rho(y_n, \boldsymbol{\theta}^T \mathbf{x}_n + \theta_0). \quad (11.60)$$

Alternatively, employing slack variables, and following a similar reasoning as in [Section 11.8.1](#), minimizing (11.60) becomes equivalent to

$$\text{minimize with respect to } \boldsymbol{\theta}, \theta_0, \xi \quad J(\boldsymbol{\theta}, \xi) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \xi_n, \quad (11.61)$$

$$\text{subject to} \quad y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq \rho - \xi_n, \quad (11.62)$$

$$\xi_n \geq 0, \quad n = 1, 2, \dots, N. \quad (11.63)$$

From now on, we will adopt the value $\rho = 1$, without harming generality. Indeed, a margin error is committed if $y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \leq 1$, corresponding to $\xi_n > 0$. On the other hand, if $\xi_n = 0$, then $y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1$. Thus, the goal of the optimization task is to drive as many of the ξ_n 's to zero as possible. The optimization task in (11.61)–(11.63) has an interesting and important geometric interpretation.

11.10.1 LINEARLY SEPARABLE CLASSES: MAXIMUM MARGIN CLASSIFIERS

Assuming linearly separable classes, there is an infinity of linear classifiers that solve the classification task exactly, without committing errors on the training set (see [Figure 11.14a](#)). It is easy to see, and it will become apparent very soon that from this infinity of hyperplanes that solve the task, we can always identify a subset such as

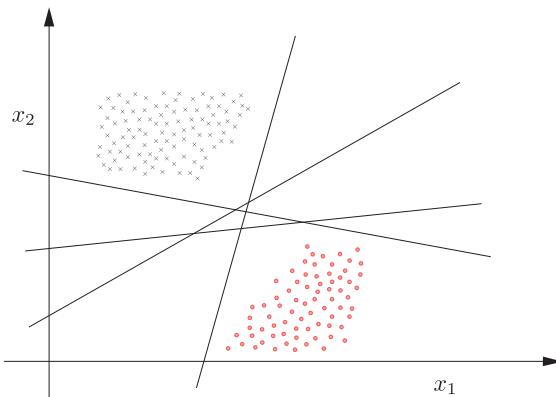
$$y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1, \quad n = 1, 2, \dots, N,$$

which guarantees that $\xi_n = 0, n = 1, 2, \dots, N$, in (11.61)–(11.63). Hence, for linearly separable classes, the previous optimization task is equivalent to

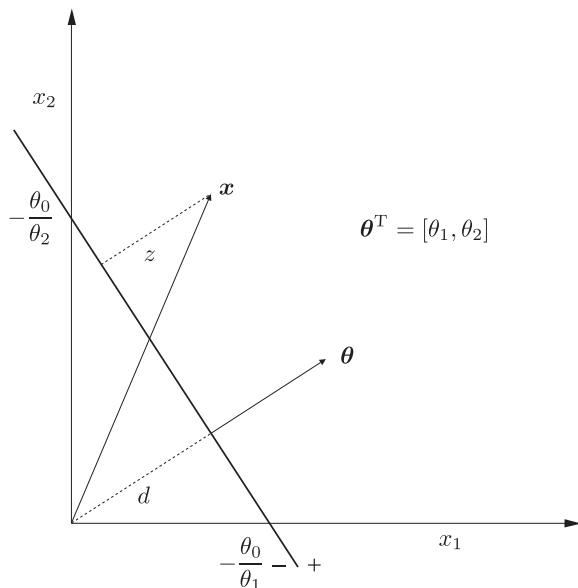
$$\text{minimize with respect to } \boldsymbol{\theta} \quad \frac{1}{2} \|\boldsymbol{\theta}\|^2 \quad (11.64)$$

$$\text{subject to} \quad y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) \geq 1, \quad n = 1, 2, \dots, N. \quad (11.65)$$

In other words, from this infinity of linear classifiers, which can solve the task and classify correctly all training patterns, our optimization task selects the one that has minimum norm. As will be explained next, the norm $\|\boldsymbol{\theta}\|$ is directly related to the margin formed by the respective classifier.

**FIGURE 11.14**

There is an infinite number of linear classifiers that can classify correctly all the patterns in a linearly separable class task.

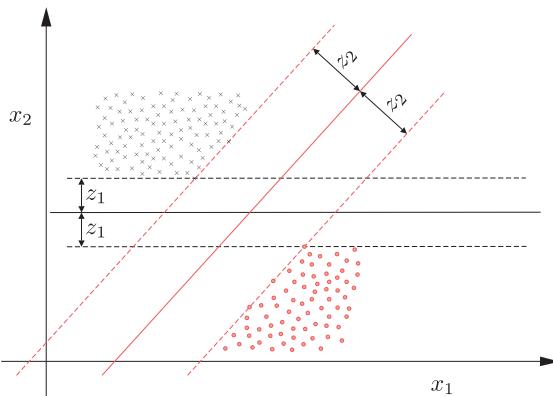
**FIGURE 11.15**

The direction of the hyperplane, $\theta^T x + \theta_0 = 0$, is determined by θ and its position in space by θ_0 .

Each hyperplane in space is described by the equation

$$f(x) = \theta^T x + \theta_0 = 0. \quad (11.66)$$

From classical geometry (see also [Problem 5.12](#)), we know that its direction in space is controlled by θ (which is perpendicular to the hyperplane) and its position is controlled by θ_0 , see [Figure 11.15](#).

**FIGURE 11.16**

For each direction, θ , “red” and “gray,” the (linear) hyperplane classifier, $\theta^T \mathbf{x} + \theta_0 = 0$, (full lines) is placed in between the two classes and normalized so that the nearest points from each class have a distance equal to one. The dotted lines, $\theta^T \mathbf{x} + \theta_0 = \pm 1$, which pass through the nearest points, are parallel to the respective classifier, and define the margin. The width of the margin is determined by the direction of the corresponding classifier in space and it is equal to $\frac{2}{\|\theta\|}$.

From the set of all hyperplanes that solve the task exactly and have certain direction (i.e., they share a common θ), we select θ_0 so as to place the hyperplane in between the two classes, such that its distance from the nearest points from each one of the two classes is the same. Figure 11.16, shows the linear classifiers (hyperplanes) in two different directions (full lines in gray and red). Both of them have been placed so as to have the same distance from the nearest points in both classes. Moreover note that, the distance z_1 associated with the “gray” classifier is smaller than the z_2 associated with the “red” one.

From basic geometry, we know that the distance of a point \mathbf{x} from a hyperplane, see Figure 11.15, is given by

$$z = \frac{|\theta^T \mathbf{x} + \theta_0|}{\|\theta\|},$$

which is obviously zero if the point lies on the hyperplane. Moreover, we can always scale by a constant factor, say a , both θ and θ_0 without affecting the geometry of the hyperplane, as described by Eq. (11.66). After an appropriate scaling, we can always make the distance of the nearest points from the two classes to the hyperplane equal to $z = \frac{1}{\|\theta\|}$; equivalently, the scaling guarantees that $f(\mathbf{x}) = \pm 1$ if \mathbf{x} is a nearest to the hyperplane point and depending on whether the point belongs to ω_1 (+1) or ω_2 (-1). The two hyperplanes, defined by $f(\mathbf{x}) = \pm 1$, are shown in Figure 11.16 as dotted lines, for both the “gray” and the “red” directions. The pair of these hyperplanes defines the corresponding margin, for each direction, whose width is equal to $\frac{2}{\|\theta\|}$.

Thus, any classifier, that is constructed as explained before and which solves the task, satisfies the following two properties:

- It has a *margin* of width equal to $\frac{1}{\|\theta\|} + \frac{1}{\|\theta\|}$

•

$$\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 \geq +1, \quad \mathbf{x}_n \in \omega_1,$$

$$\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0 \leq -1, \quad \mathbf{x}_n \in \omega_2.$$

Hence, the optimization task in (11.64)–(11.65) computes the linear classifier, which *maximizes the margin* subject to the constraints.

The margin interpretation of the regularizing term $\|\boldsymbol{\theta}\|^2$ ties nicely the task of designing classifiers, which maximize the margin, with the statistical learning theory and the pioneering work of Vapnik-Chernovenkis, which establishes elegant performance bounds on the generalization properties of such classifiers: see, for example, [27, 125, 128, 129].

The solution

Following similar steps as for the support vector regression case, the solution is given as a linear combination of a subset of the training samples, that is,

$$\hat{\boldsymbol{\theta}} = \sum_{n=1}^{N_s} \lambda_n y_n \mathbf{x}_n, \quad (11.67)$$

where N_s are the nonzero Lagrange multipliers. It turns out that only the Lagrange multipliers associated with the nearest-to-the-classifier points, that is, those points satisfying the constraints with equality ($y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) = 1$), are nonzero. These are known as the *support vectors*. The Lagrange multipliers corresponding to the points farther away ($y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) > 1$) are zero.

For the more general RKHS case, we have that

$$\hat{\boldsymbol{\theta}}(\cdot) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\cdot, \mathbf{x}_n),$$

which leads to the following prediction rule. Given an unknown \mathbf{x} , its class label is predicted according to the sign of

$$\hat{y}(\mathbf{x}) = \sum_{n=1}^{N_s} \lambda_n y_n \kappa(\mathbf{x}, \mathbf{x}_n) + \hat{\theta}_0 : \quad \text{Support Vector Machine Prediction,} \quad (11.68)$$

where $\hat{\theta}_0$ is obtained by selecting all constraints with $\lambda_n \neq 0$, corresponding to

$$y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \hat{\theta}_0) - 1 = 0, \quad n = 1, 2, \dots, N_s,$$

which for the RKHS case becomes

$$y_n \left(\sum_{m=1}^{N_s} \lambda_m y_m \kappa(\mathbf{x}_m, \mathbf{x}_n) + \hat{\theta}_0 \right) - 1 = 0, \quad n = 1, 2, \dots, N_s.$$

and $\hat{\theta}_0$ is computed as the average of the values obtained from each one of these constraints. Although the solution is unique, the corresponding Lagrange multipliers may not be unique; see, for example, [125].

Finally, it must be stressed that the number of support vectors is related to the generalization performance of the classifier. The *smaller the number of support vectors the better the generalization is expected to be*, [27, 125].

The optimization task

This part can also be bypassed in a first reading.

The task in (11.64)–(11.65) is a quadratic programming task and can be solved following similar steps to those adopted for the SVR task.

The associated Lagrangian is given by

$$L(\boldsymbol{\theta}, \theta_0, \lambda) = \frac{1}{2} \|\boldsymbol{\theta}\|^2 - \sum_{n=1}^N \lambda_n (y_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) - 1), \quad (11.69)$$

and the KKT conditions (Appendix C) become

$$\frac{\partial}{\partial \boldsymbol{\theta}} L(\boldsymbol{\theta}, \theta_0, \lambda) = \mathbf{0} \longrightarrow \hat{\boldsymbol{\theta}} = \sum_{n=1}^N \lambda_n y_n \mathbf{x}_n, \quad (11.70)$$

$$\frac{\partial}{\partial \theta_0} L(\boldsymbol{\theta}, \theta_0, \lambda) = 0 \longrightarrow \sum_{n=1}^N \lambda_n y_n = 0, \quad (11.71)$$

$$\lambda_n (y_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) - 1) = 0, \quad n = 1, 2, \dots, N, \quad (11.72)$$

$$\lambda_n \geq 0, \quad n = 1, 2, \dots, N. \quad (11.73)$$

The Lagrange multipliers are obtained via the dual representation form after plugging (11.70) into the Lagrangian (Problem 11.11) that is,

$$\text{maximize with respect to } \lambda \quad \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m, \quad (11.74)$$

$$\text{subject to} \quad \lambda_n \geq 0, \quad (11.75)$$

$$\sum_{n=1}^N \lambda_n y_n = 0. \quad (11.76)$$

For the case where the original task has been mapped to an RKHS, the cost function becomes

$$\sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \kappa(\mathbf{x}_n, \mathbf{x}_m). \quad (11.77)$$

- According to (11.72), if $\lambda_n \neq 0$, then necessarily

$$y_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) = 1.$$

That is, the respective points are the closest points, from each class, to the classifier (distance $\frac{1}{\|\boldsymbol{\theta}\|}$). They lie on either of the two hyperplanes forming the border of the margin. These points are the support vectors and the respective constraints are known as the *active constraints*. The rest of the points, associated with

$$y_n (\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) > 1, \quad (11.78)$$

which lie outside the margin, correspond to $\lambda_n = 0$ (*inactive constraints*).

- The cost function in (11.64) is strictly convex and, hence, the solution of the optimization task is unique (Appendix C).

11.10.2 NONSEPARABLE CLASSES

We now turn our attention to the more realistic case of overlapping classes and the corresponding geometric representation of the task in (11.61)–(11.63). In this case, there is no (linear) classifier that can classify correctly all the points, and some errors are bound to occur. Figure 11.17 shows the respective geometry for a linear classifier. There are three types of points.

- Points that lie on the border or outside the margin and in the correct side of the classifier, that is,

$$y_n f(\mathbf{x}_n) \geq 1.$$

These points commit no (margin) error, that is,

$$\xi_n = 0.$$

- Points which lie on the correct side of the classifier, but lie inside the margin (circled points), that is,

$$0 < y_n f(\mathbf{x}_n) < 1.$$

These points commit a margin error, and

$$0 < \xi_n < 1.$$

- Points that lie on the wrong side of the classifier (points in squares), that is,

$$y_n f(\mathbf{x}_n) \leq 0.$$

These points commit an error and

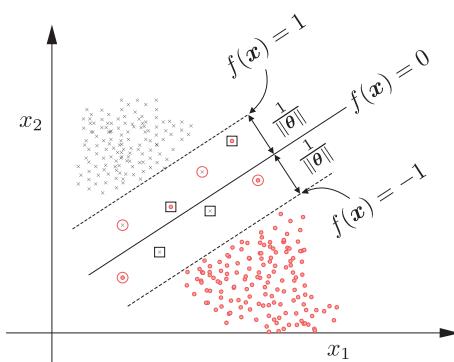


FIGURE 11.17

When classes are overlapping, there are three types of points: (a) points that lie outside or on the borders of the margin and are classified correctly ($\xi_n = 0$); (b) points inside the margin and classified correctly ($0 < \xi_n < 1$) denoted by circles; and (c) misclassified points denoted by a square ($\xi_n \geq 1$).

$$1 \leq \xi_n.$$

Our desire would be to estimate a hyperplane classifier, so as to *maximize the margin and at the same time to keep the number of errors (including margin errors) as small as possible*. This goal could be expressed via the optimization task in (11.61)–(11.62), if in place of ξ_n we had the indicator function, $I(\xi_n)$, where

$$I(\xi) = \begin{cases} 1, & \text{if } \xi > 0, \\ 0, & \text{if } \xi = 0. \end{cases}$$

However, in such a case the task becomes a combinatorial one. So, we relax the task and use ξ_n in place of the indicator function, leading to (11.61)–(11.62). Note that optimization is achieved in a trade-off rationale; the user-defined parameter, C , controls the influence of each of the two contributions to the minimization task. If C is large, the resulting margin (the distance between the two hyperplanes defined by $f(\mathbf{x}) = \pm 1$) will be small, in order to commit a smaller number of margin errors. If C is small, the opposite is true. As we will see from the simulation examples, the choice of C is very critical.

The solution

Once more, the solution is given as a linear combination of a subset of the training points,

$$\hat{\boldsymbol{\theta}} = \sum_{n=1}^{N_s} \lambda_n y_n \mathbf{x}_n, \quad (11.79)$$

where λ_n , $n = 1, 2, \dots, N_s$, are the nonzero Lagrange multipliers associated with the support vectors. In this case, support vectors are all points that lie either (a) on the pair of the hyperplanes that define the margin or (b) inside the margin or (c) outside the margin but on the wrong side of the classifier. That is, correctly classified points that lie outside the margin do not contribute to the solution, because the corresponding Lagrange multipliers are zero. Hence, the class prediction rule is the same as in (11.68), where $\hat{\theta}_0$ is computed from the constraints corresponding to $\lambda_n \neq 0$ and $\xi_n = 0$; these correspond to the points that lie on the hyperplanes defining the margin and on the correct side of the classifier.

The optimization task

As before, this part can be bypassed in a first reading.

The Lagrangian associated with (11.61)–(11.63) is given by

$$\begin{aligned} L(\boldsymbol{\theta}, \theta_0, \boldsymbol{\xi}, \boldsymbol{\lambda}) = & \frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{n=1}^N \xi_n - \sum_{n=1}^N \mu_n \xi_n \\ & - \sum_{n=1}^N \lambda_n (y_n (\boldsymbol{\theta}^\top \mathbf{x}_n + \theta_0) - 1 + \xi_n), \end{aligned}$$

leading to the following KKT conditions,

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \mathbf{0} \longrightarrow \hat{\boldsymbol{\theta}} = \sum_{n=1}^N \lambda_n y_n \mathbf{x}_n, \quad (11.80)$$

$$\frac{\partial L}{\partial \theta_0} = 0 \longrightarrow \sum_{n=1}^N \lambda_n y_n = 0, \quad (11.81)$$

$$\frac{\partial L}{\partial \xi_n} = 0 \longrightarrow C - \mu_n - \lambda_n = 0, \quad (11.82)$$

$$\lambda_n(y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) - 1 + \xi_n) = 0, \quad n = 1, 2, \dots, N, \quad (11.83)$$

$$\mu_n \xi_n = 0, \quad n = 1, 2, \dots, N, \quad (11.84)$$

$$\mu_n \geq 0, \quad \lambda_n \geq 0, \quad n = 1, 2, \dots, N, \quad (11.85)$$

and in our by-now-familiar procedure, the dual problem is cast as

$$\text{maximize with respect to } \boldsymbol{\lambda} \quad \sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \mathbf{x}_n^T \mathbf{x}_m \quad (11.86)$$

$$\text{subject to} \quad 0 \leq \lambda_n \leq C, \quad n = 1, 2, \dots, N, \quad (11.87)$$

$$\sum_{n=1}^N \lambda_n y_n = 0. \quad (11.88)$$

When working in an RKHS, the cost function becomes

$$\sum_{n=1}^N \lambda_n - \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N \lambda_n \lambda_m y_n y_m \kappa(\mathbf{x}_n, \mathbf{x}_m).$$

Observe that the only difference compared to its linearly class-separable counterpart in (11.74)–(11.76) is the existence of C in the inequality constraints for λ_n . The following comments are in order:

- From (11.83), we conclude that for all the points outside the margin, and on the correct side of the classifier, which correspond to $\xi_n = 0$, we have

$$y_n(\boldsymbol{\theta}^T \mathbf{x}_n + \theta_0) > 1,$$

hence, $\lambda_n = 0$. That is, these points do not participate in the formation of the solution in (11.80).

- $\lambda_n \neq 0$ only for the points that live either on the border hyperplanes or inside the margin or outside the margin but on the wrong side of the classifier. These comprise the support vectors.
- For points lying inside the margin or outside but on the wrong side, $\xi_n > 0$; hence, from (11.84), $\mu_n = 0$ and from (11.82) we get,

$$\lambda_n = C.$$

- Support vectors, which lie on the margin border hyperplanes, satisfy $\xi_n = 0$ and therefore μ_n can be nonzero, which leads to

$$0 \leq \lambda_n \leq C.$$

Remarks 11.5.

- *v-SVM*: An alternative formulation for the SVM classification has been given in [105], where the margin is defined by the pair of hyperplanes,

$$\boldsymbol{\theta}^T \mathbf{x} + \theta_0 = \pm \rho,$$

and $\rho \geq 0$ is left as a free variable, giving rise to the ν -SVM; ν controls the importance of ρ in the associated cost function. It has been shown, [21], that the ν -SVM and the formulation discussed above, which is sometimes referred to as the C -SVM, lead to the same solution for appropriate choices of C and ν . However, the advantage of ν -SVM lies in the fact that ν can be directly related to bounds concerning the number of support vectors and the corresponding error rate, see also [125].

- *Reduced Convex Hull Interpretation:* In [57], it has been shown that, for linearly separable classes, the SVM formulation is equivalent with finding the nearest points between the convex hulls, formed by the data in the two classes. This result was generalized for overlapping classes in [32]; it is shown that in this case, the ν -SVM task is equivalent to searching for the nearest points between the *reduced convex hulls* (RCH), associated with the training data. Searching for the RCH is a computationally hard task of combinatorial nature. The problem was efficiently solved in [74–76, 123], who came up with efficient iterative schemes to solve the SVM task, via nearest point searching algorithms. More on these issues can be obtained from [66, 124, 125].
- *ℓ_1 -Regularized Versions:* The regularization term, which has been used in the optimization tasks discussed so far, has been based on the ℓ_2 norm. A lot of research effort has been focused on using ℓ_1 norm regularization for tasks treating the linear case. To this end, a number of different loss functions have been used in addition to the least-squares, the hinge loss, and the ϵ -insensitive versions, as for example the logistic loss. The solution of such tasks comes under the general framework discussed in Chapter 8. As a matter of fact, some of these methods have been discussed there. A related concise review is provided in [132].
- *Multitask Learning:* In multitask learning, two or more related tasks, for example, classifiers, are jointly optimized. Such problems are of interest in, for example, econometrics, and bioinformatics. In [38], it is shown that the problem of estimating many task functions with regularization can be cast as a single task learning problem if a family of appropriately defined multitask kernel functions is used.

Example 11.4. In this example, the performance of the SVM is tested in the context of a two-class two-dimensional classification task. The data set comprises $N = 150$ points uniformly distributed in the region $[-5, 5] \times [-5, 5]$. For each point, $\mathbf{x}_n = [x_{n,1}, x_{n,2}]^T$, $n = 1, 2, \dots, N$, we compute

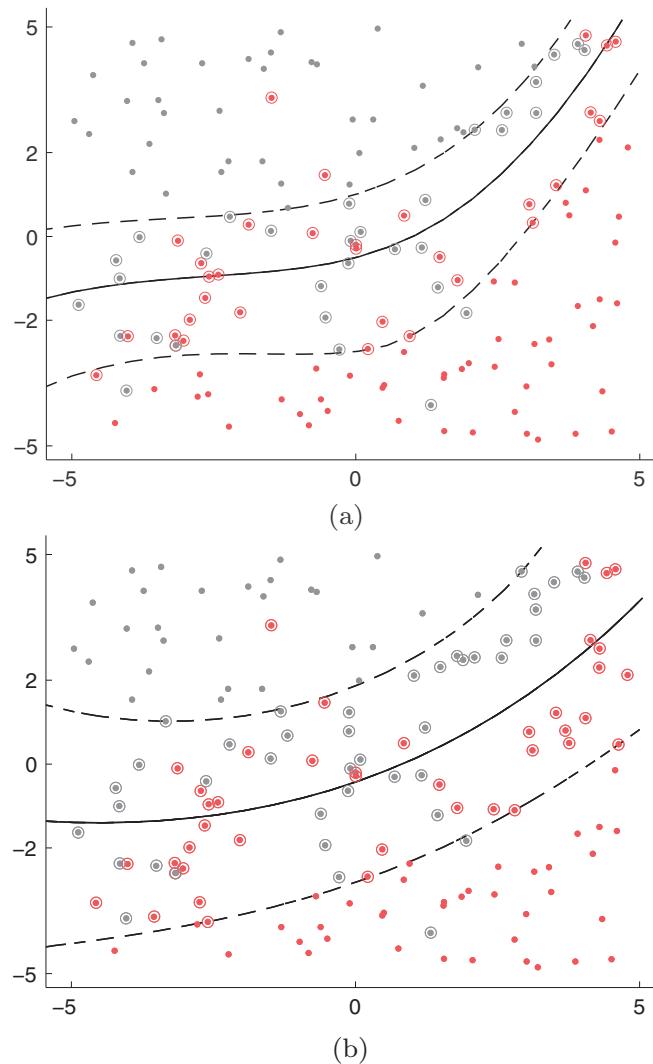
$$y_n = 0.5x_{n,1}^3 + 0.5x_{n,1}^2 + 0.5x_{n,1} + 1 + \eta,$$

where η stands for zero-mean Gaussian noise of variance $\sigma_\eta^2 = 4$. The point is assigned to either of the two classes, depending on which side of the graph of the function

$$f(x) = 0.5x^3 + 0.5x^2 + 0.5x + 1,$$

in the two-dimensional space, y_n lies. That is, if $y_n > f(x_{n,1})$, the point is assigned to class ω_1 ; otherwise, it is assigned to class ω_2 .

The Gaussian kernel was used with $\sigma = 10$, as this resulted in the best performance. Figure 11.18a shows the obtained classifier for $C = 20$ and Figure 11.18b for $C = 1$. Observe how the obtained classifier, and hence the performance, depends heavily on the choice of C . In the former case, the number of the support vectors was equal to 64 and for the latter equal to 84.

**FIGURE 11.18**

(a) The training data points for the two classes (red and gray respectively) of Example 11.4. The full line is the graph of the obtained SVM classifier and the dotted lines indicate the margin, for $C = 20$. (b) The result for $C = 1$. For both cases, the Gaussian kernel with $\sigma = 20$ was used.

11.10.3 PERFORMANCE OF SVMs AND APPLICATIONS

A notable characteristic of the support vector machines is that the complexity is independent of the dimensionality of the respective RKHS. The need of having a large number of parameters is bypassed and this has an influence on the generalization performance; SVMs exhibit very good generalization performance in practice. Theoretically, such a claim is substantiated by their maximum margin interpretation, in the framework of the elegant *structural risk minimization theory*, [27, 125, 128].

An extensive comparative study concerning the performance of SVMs against 16 other popular classifiers has been reported in [78]. The results verify that the SVM ranks at the very top among these classifiers, although there are cases for which other methods score better performance. Another comparative performance study is reported in [23].

It is hard to find a discipline related to machine learning/pattern recognition where the support vector machines and the concept of working in kernel spaces has not been applied. Early applications included data mining, spam categorization, object recognition, medical diagnosis, optical character recognition (OCR), bioinformatics; see, for example, [27] for a review. More recent applications include cognitive radio, for example, [34], spectrum cartography and network flow prediction [9], and image de-noising [13].

In [118], the notion of *kernel embedding* of conditional probabilities is reviewed, as a means to address challenging problems in graphical models. The notion of kernelization has also been extended in the context of *tensor-based* models, [49, 108, 133]. Kernel-based hypothesis testing is reviewed in [48]. In [122], the use of kernels in manifold learning is discussed in the framework of diffusion maps. The task of analyzing the performance of kernel techniques with regard to dimensionality, signal-to-noise ratio and local error bars is reviewed in [82]. In [121], a collection of articles related to kernel-based methods and applications is provided.

11.10.4 CHOICE OF HYPERPARAMETERS

One of the main issues associated with SVM/SVRs is the choice of the parameter, C , which controls the relative influence of the loss and the regularizing parameter in the cost function. Although some efforts have been made in developing theoretical tools for the respective optimization, the path that has survived in practice is that of cross-validation techniques against a test data set. Different values of C are used to train the model, and the value that results in the best performance over the test set is selected.

The other main issue is the choice of the kernel function. Different kernels lead to different performance. Let us look carefully at the expansion in (11.17). One can think of $\kappa(\mathbf{x}, \mathbf{x}_n)$ as a function that measures the *similarity* between \mathbf{x} and \mathbf{x}_n ; in other words, $\kappa(\mathbf{x}, \mathbf{x}_n)$ matches \mathbf{x} to the training sample \mathbf{x}_n . A kernel is local if $\kappa(\mathbf{x}, \mathbf{x}_n)$ takes relatively large values in a small region around \mathbf{x}_n . For example, when the Gaussian kernel is used, the contribution of $\kappa(\mathbf{x}, \mathbf{x}_n)$ away from \mathbf{x}_n decays exponentially fast, depending of the value of σ^2 . Thus, the choice of σ^2 is very crucial. If the function to be approximated is smooth, then large values of σ^2 should be employed. On the contrary, if the function is highly varying in input space, the Gaussian kernel may not be the best choice. As a matter of fact, if for such cases the Gaussian kernel is employed, one must have access to a large number of training data, in order to fill in the input space densely enough, so as to be able to obtain a good enough approximation of such a function. This brings into the scene another critical factor in machine learning, related to the size of the training set. The latter is not only dictated by the dimensionality of the input space (curse of

dimensionality) but it also depends on the type of variation that the unknown function undergoes (see, for example, [12]). In practice, in order to choose the right kernel function, one uses different kernels and after cross validation selects the “best” one for the specific problem.

A line of research is to design kernels that match the data at hand, based either on some prior knowledge or via some optimization path; see, for example, [28, 65]. Soon, in [Section 11.13](#) we will discuss techniques which use multiple kernels in an effort to optimally combine their individual characteristics.

11.11 COMPUTATIONAL CONSIDERATIONS

Solving a quadratic programming task, in general, requires $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ memory operations. To cope with such demands a number of decomposition techniques have been devised, for example, [20, 54], which “break” the task into a sequence of smaller ones. In [58, 90, 91], the *sequential minimal optimization* (SMO) algorithm breaks the task into a sequence of problems comprising two points, which can be solved analytically. Efficient implementation of such schemes lead to an empirical training time that scales between $\mathcal{O}(N)$ and $\mathcal{O}(N^{2.3})$.

The schemes derived in [75, 76] treat the task as a minimum distance points search between reduced convex hulls and end up with an iterative scheme, which projects the training points on hyperplanes. The scheme leads to even more efficient implementations compared to [58, 90]; moreover, the minimum distance search algorithm has a built-in enhanced parallelism.

The issue of parallel implementation is also discussed in [19]. Issues concerning complexity and accuracy are reported in [52]. In the latter, polynomial time algorithms are derived that produce approximate solutions with a guaranteed accuracy for a class of QP problems including SVM classifiers.

Incremental versions for solving the SVM task, which are dealing with sequentially arriving data have also appeared, for example, [24, 35, 101]. In the latter, at each iteration a new point is considered and the previously selected set of support vectors (active set) is updated accordingly by adding/removing samples.

Online versions that apply in the primal problem formulation, have also been proposed. In [85], an iterative reweighted LS approach is followed that alternates weight optimization with cost constraint forcing. A structurally and computationally simple scheme, named *PEGASOS: Primal Estimated sub-Gradient SOlver for SVM*, has been proposed in [106]. The algorithm is of an iterative subgradient form applied on the regularized empirical hinge loss function in (11.60) (see also, [Chapter 8](#)). The algorithm, for the case of linear kernels, exhibits very good convergence properties and it finds an ϵ -accurate solution in $\mathcal{O}\left(\frac{C}{\epsilon}\right)$ iterations.

In [41, 55] the classical technique of *cutting planes* for solving convex tasks has been employed in the context of SVMs in the primal domain. The resulting algorithm is very efficient and, in particular for the linear SVM case, the complexity becomes of order $O(N)$.

In [25], a comparative study between solving the SVM tasks in the primal and dual domains is carried out. The findings of the paper point out that both paths are equally efficient, for the linear as well as for the nonlinear cases. Moreover, when the goal is to resort to approximate solutions, opting for the primal task can offer certain benefits. In addition, working in the primal also offers the advantage of tuning the hyperparameters by resorting to joint optimization. One of the main advantages of resorting to the dual domain was the luxury of casting the task in terms of inner products. However, this is also possible in

the primal, by appropriate exploitation of the representer theorem. We will see such examples soon, in Section 11.12.

More recently, a version of SVM for distributed processing has been presented in [40]. In [100] the SVM task is solved in a subspace using the method of random projections.

11.11.1 MULTICLASS GENERALIZATIONS

The SVM classification task has been introduced in the context of a two-class classification task. The more general M -class case can be treated in various ways:

- *One-against-All*: One solves M two-class problems. Each time, one of the classes is classified against all the others using a different SVM. Thus, M classifiers are estimated, that is,

$$f_m(\mathbf{x}) = 0, \quad m = 1, 2, \dots, M,$$

which are trained so that $f_m(\mathbf{x}) > 0$ for $\mathbf{x} \in \omega_m$ and $f_m(\mathbf{x}) < 0$ if \mathbf{x} otherwise. Classification is achieved via the rule

$$\text{assign } \mathbf{x} \text{ in } \omega_k : \text{ if } k = \arg \max_m f_m(\mathbf{x}).$$

According to this method, there may be regions in space where more than one of the discriminant functions score a positive value, [125]. Moreover, another disadvantage of this approach is the so-called class imbalance problem; this may be caused by the fact that the number of training points in one of the classes (which comprises the data from $M - 1$ classes) is much larger than the points in the other. Issues related to the class imbalance problem are discussed in [125].

- *One-against-one*. According to this method, one solves $\frac{M(M-1)}{2}$ binary classification tasks by considering all classes in pairs. The final decision is taken on the basis of the majority rule.
- In [129], the SVM rationale is extended in estimating simultaneously M hyperplanes. However, this technique ends up with a large number of parameters, equal to $N(M - 1)$, which have to be estimated via a single minimization task; this turns out to be rather prohibitive for most practical problems.
- In [33], the multiclass task is treated in the context of error correcting codes. Each class is associated with a binary code word. If the code words are properly chosen, an error resilience is “embedded” into the process; see also [125]. For a comparative study of multiclass classification schemes, see, for example, [39] and the references therein.
- *Division and Clifford Algebras*: The SVM framework has also been extended to treat complex and hypercomplex data, both for the regression as well as the classification cases, using either division algebras [101], or Clifford algebras [8]. In [126], the case of quaternion RKH spaces is considered.

A more general method for the case of complex-valued data, which exploits the notion of *widely linear estimation* as well as *pure complex kernels*, has been presented in [14]. In this paper, it is shown that any complex SVM/SVR task is equivalent with solving two real SVM/SVR tasks exploiting a specific real kernel, which is generated by the chosen complex one. Moreover, in the classification case, it is shown that the proposed framework inherently splits the complex space into four parts. This leads naturally to solving the four-class task (quaternary classification), instead of the standard two-classes scenario of the real SVM. This rationale can be used in a multiclass problem as a split-class scenario.

11.12 ONLINE LEARNING IN RKHS

We have dealt with online learning in various parts of the book. Most of the algorithms that have been discussed can also be stated for general Hilbert spaces. The reason that we are going to dedicate this section specifically to RKHS is that developing online algorithms for such spaces poses certain computational obstacles. In parametric modeling in a Euclidean space \mathbb{R}^l , the number of unknown parameters remains fixed for all time instants. In contrast, modeling an unknown function to lie in an RKHS, makes the number of unknown parameters grow linearly with time; thus, complexity increases with time iterations and eventually will become unmanageable both in memory as well as in the number of operations.

We will discuss possible solutions to this problem in the context of three algorithms, which are popular and fall nicely within the framework that has been adopted throughout this book.

11.12.1 THE KERNEL LMS (KLMS)

This section is intended for more experienced readers, so we will cast the task in a general RKHS space, \mathbb{H} .

Let $\mathbf{x} \in \mathbb{R}^l$ and consider the feature map

$$\mathbf{x} \mapsto \phi(\mathbf{x}) = \kappa(\cdot, \mathbf{x}) \in \mathbb{H}.$$

The task is to estimate $f \in \mathbb{H}$, so as to minimize the expected risk

$$J(f) = \frac{1}{2} \mathbb{E} [|y - \langle f, \phi(\mathbf{x}) \rangle|^2], \quad (11.89)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product operation in \mathbb{H} . Differentiating¹¹ with respect to f , it turns out that

$$\nabla_f J(f) = -\mathbb{E} [\phi(\mathbf{x})(y - \langle f, \phi(\mathbf{x}) \rangle)].$$

Adopting the stochastic gradient rationale, and replacing random variables with observations, the following time update recursion for minimizing the expected risk results,

$$\begin{aligned} f_n &= f_{n-1} + \mu_n e_n \phi(\mathbf{x}_n), \\ &= f_{n-1} + \mu_n e_n \kappa(\cdot, \mathbf{x}_n), \end{aligned} \quad (11.90)$$

where

$$e_n = y_n - \langle f_{n-1}, \phi(\mathbf{x}_n) \rangle,$$

with (y_n, \mathbf{x}_n) $n = 1, 2, \dots$ being the received observations. Starting the iterations from $f_0 = 0$ and fixing $\mu_n = \mu$, as in the standard LMS, we obtain¹²

$$f_n = \mu \sum_{i=1}^n e_i \kappa(\cdot, \mathbf{x}_i), \quad (11.91)$$

¹¹ Differentiation here is in the context of Frechet derivatives. In analogy to the gradient, it turns out that $\nabla_f \langle f, g \rangle = g$, for $f, g \in \mathbb{H}$.

¹² In contrast to the LMS in Chapter 5 the time starts at $n = 1$, and not at $n = 0$, so as to be in line with the notation used in this chapter.

and the prediction of the output, based on information (training samples) received up to and including time instant $n - 1$, is given by

$$\begin{aligned}\hat{y}_n &= \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_n) \rangle \\ &= \mu \sum_{i=1}^{n-1} e_i \kappa(\mathbf{x}_n, \mathbf{x}_i).\end{aligned}$$

Note that the same equation results, if one starts from the standard form of LMS expressed in \mathbb{R}^l , and applies the kernel trick at the final stage (try it). Observe that (11.91) is in line with the representer theorem. Thus, replacing μe_i with θ_i , and plugging (11.91) in (11.90), it turns out that at every time instant, the KLMS comprises the following steps:

$$\begin{aligned}e_n &= y_n - \hat{y}_n = y_n - \sum_{i=1}^{n-1} \theta_i \kappa(\mathbf{x}_n, \mathbf{x}_i), \\ \theta_n &= \mu e_n.\end{aligned}$$

(11.92)

The *normalized* KLMS results if the update becomes

$$\theta_n = \mu \frac{e_n}{\kappa(\mathbf{x}_n, \mathbf{x}_n)}.$$

Observe that the memory grows unbounded. So, in practice, a *sparsification* rule has to be applied. There are various strategies that have been proposed. One is via regularization and the other one is via the formation of a *dictionary*. In the latter case, instead of using in the expansion in (11.91) all the training points up to time n , a subset is selected. This subset forms the dictionary, \mathcal{D}_n , at time n . Starting from an empty set, $\mathcal{D}_0 = \emptyset$, the dictionary can grow following different strategies. Some typical examples are

- *Novelty criterion* [68]:
 - Let us denote the current dictionary as \mathcal{D}_{n-1} , its cardinality by M_{n-1} , and its elements as \mathbf{u}_k , $k = 1, 2, \dots, M_{n-1}$. When a new observation arrives, its distance from all the points in the \mathcal{D}_{n-1} is evaluated

$$d(\mathbf{x}_n, \mathcal{D}_{n-1}) = \min_{\mathbf{u}_k \in \mathcal{D}_{n-1}} \{\|\mathbf{x}_n - \mathbf{u}_k\|\}.$$

If this distance is smaller than a threshold, δ_1 , then the new point is ignored and $\mathcal{D}_n = \mathcal{D}_{n-1}$.

- If not, the error is computed

$$e_n = y_n - \hat{y}_n = y_n - \sum_{k=1}^{M_{n-1}} \theta_k \kappa(\mathbf{x}_n, \mathbf{u}_k). \quad (11.93)$$

If $|e_n| < \delta_s$, where δ_s is a threshold, then the point is discarded. If not, \mathbf{x}_n is inserted in the dictionary and

$$\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{\mathbf{x}_n\}.$$

- *The Coherence Criterion*: According to this scheme, the point \mathbf{x}_n is added in the dictionary if its *coherence* is above a given threshold, ϵ_0 , that is,

$$\max_{\mathbf{u}_k \in \mathcal{D}_{n-1}} \{|\kappa(\mathbf{x}_n, \mathbf{u}_k)|\} > \epsilon_0.$$

It can be shown [96] that, under this rule, the cardinality of \mathcal{D}_n remains *finite*, as $n \rightarrow \infty$.

- *Surprise Criterion* [69]: The *surprise* of a new pair (y_n, \mathbf{x}_n) with respect to a learning system, \mathcal{T} , is defined as the log-likelihood of (y_n, \mathbf{x}_n) , i.e.,

$$S_{\mathcal{T}}(y_n, \mathbf{x}_n) = -\ln p((y_n, \mathbf{x}_n) | \mathcal{T}).$$

According to this measure, a data pair is classified to either of the following three categories:

- *Abnormal*: $S_{\mathcal{T}}(y_n, \mathbf{x}_n) > \delta_1$,
- *Learnable*: $\delta_1 \geq S_{\mathcal{T}}(y_n, \mathbf{x}_n) \geq \delta_2$,
- *Redundant*: $S_{\mathcal{T}}(y_n, \mathbf{x}_n) < \delta_2$, where δ_1, δ_2 are threshold values. For the case of the LMS with Gaussian inputs, the following is obtained,

$$S_{\mathcal{T}}(y_n, \mathbf{x}_n) = \frac{1}{2} \ln(r_n) + \frac{e_n^2}{2r_n},$$

where

$$r_n = \lambda + \kappa(\mathbf{x}_n, \mathbf{x}_n) - \max_{\mathbf{u}_k \in \mathcal{D}_{n-1}} \left\{ \frac{\kappa^2(\mathbf{x}_n, \mathbf{u}_k)}{\kappa(\mathbf{u}_k, \mathbf{u}_k)} \right\},$$

where λ is a user-defined regularization parameter.

A main drawback of all the previous techniques is that once a data point (e.g., \mathbf{x}_k) is inserted in the dictionary, it remains there forever; also, the corresponding coefficients, θ_k , in the expansion in (11.93) do not change. This can affect the tracking ability of the algorithm in time-varying environments.

A different technique, which gives the chance of changing the respective weights of the points in the dictionary, is the *quantization* technique, giving rise to the so-called *quantized* KLMS, given in **Algorithm 11.1** ([26]).

Algorithm 11.1 (The quantized kernel LMS).

- Initialize
 - $\mathcal{D} = \emptyset, M = 0$.
 - Select μ and then the quantization level δ .
 - $d(\mathbf{x}, \emptyset) := \Delta > \delta, \forall \mathbf{x}$.
- **For** $n = 1, 2, \dots$, **Do**
 - **If** $n = 1$ **then**
 - $\hat{y}_n = 0$
 - **else**
 - $\hat{y}_n = \sum_{k=1: \mathbf{u}_k \in \mathcal{D}}^M \theta_k \kappa(\mathbf{x}_n, \mathbf{u}_k)$
 - **End If**
 - $e_n = y_n - \hat{y}_n$
 - $d(\mathbf{x}_n, \mathcal{D}) = \min_{\mathbf{u}_k \in \mathcal{D}} \|\mathbf{x}_n - \mathbf{u}_k\| = \|\mathbf{x}_n - \mathbf{u}_{l_0}\|$ for some $l_0 \in \{1, 2, \dots, M\}$
 - **If** $d(\mathbf{x}_n, \mathcal{D}) > \delta$, **then**
 - $\theta_n = \mu e_n$; or $\theta_n = \frac{\mu e_n}{\kappa(\mathbf{x}_n, \mathbf{x}_n)}$, for NKLMS.
 - $M = M + 1$
 - $\mathbf{u}_M = \mathbf{x}_n$
 - $\mathcal{D} = \mathcal{D} \cup \{\mathbf{x}_M\}$
 - $\theta = [\theta^T, \theta_n]^T$; increase dimensionality of θ by one.

- **Else**
 - Keep dictionary unchanged.
 - $\theta_{l_0} = \theta_{l_0} + \mu e_n$; Update the weight of the nearest point.
- **End If**
- **End For**

The algorithm returns $\boldsymbol{\theta}$ as well as the dictionary, hence

$$f_n(\cdot) = \sum_{k=1}^M \theta_k \kappa(\cdot, \mathbf{u}_k)$$

and

$$\hat{y}_n = \sum_{k=1}^M \theta_k \kappa(\mathbf{x}_n, \mathbf{u}_k).$$

The KLMS without the sparsification approximation can be shown to converge asymptotically, in the mean sense, to the value that optimizes the mean-square cost function in (11.89) for small values of μ , [68]. A stochastic analysis of the KLMS for the case of the Gaussian kernel has been carried in [88].

An interesting result concerning the KLMS was derived in [68]. It has been shown that the KLMS trained on a *finite* number of points, N , turns out to be the stochastic approximation of the following constrained optimization task,

$$\min_f J(f) = \frac{1}{N} \sum_{i=1}^N (y_i - \langle f, \phi(\mathbf{x}_i) \rangle)^2 \quad (11.94)$$

$$\text{s.t. } \|f\|^2 \leq C, \quad (11.95)$$

where the value of C can be computed analytically. This result builds upon the H^∞ optimality of the LMS, as discussed in Chapter 5. Note that this is equivalent to having used regularization in (11.89). The constraint in (11.95) is in line with the conditions obtained from the regularization network theory [45, 92], for ensuring that the problem is well posed (Chapter 3) and is sufficient for consistency of the empirical error minimization forcing smoothness and stability [92]. In a nutshell, the KLMS is well-posed in an RKHS, without the need of an extra regularization term to penalize the empirical loss function.

11.12.2 THE NAIVE ONLINE R_{reg} MINIMIZATION ALGORITHM (NORMA)

The KLMS algorithm is a stochastic gradient algorithm for the case of the squared error loss function. In the current section, our interest moves to more general convex loss functions, such as those discussed in Chapter 8. The squared error loss function is just such an instance.

Given the loss function

$$\mathcal{L} : \mathbb{R} \times \mathbb{R} \mapsto [0, +\infty),$$

the goal is to obtain an $f \in \mathbb{H}$, where \mathbb{H} is an RKHS defined by a kernel $\kappa(\cdot, \cdot)$, such as to minimize the expected risk

$$J(f) = \mathbb{E} [\mathcal{L}(y, f(\mathbf{x}))]. \quad (11.96)$$

Instead, we turn our attention to selecting, f , so as to minimize the regularized empirical risk over the available training set

$$J_{emp,\lambda}(f, N) = J_{emp}(f, N) + \frac{\lambda}{2} \|f\|^2, \quad (11.97)$$

where

$$J_{emp}(f, N) = \frac{1}{N} \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)).$$

Following the same rationale as the one that has been adopted for finite dimensional (Euclidean) spaces, in order to derive stochastic gradient algorithms, the *instantaneous* counterpart of (11.97) is defined as

$$\boxed{\mathcal{L}_{n,\lambda}(f) := \mathcal{L}(y_n, f(\mathbf{x}_n)) + \frac{\lambda}{2} \|f\|^2 : \text{ Instantaneous Loss}} \quad (11.98)$$

and it is used in the time-recursive rule for searching for the optimum, that is,

$$f_n = f_{n-1} - \mu_n \frac{\partial}{\partial f} \mathcal{L}_{n,\lambda}(f)|_{f=f_{n-1}},$$

where $\frac{\partial}{\partial f}$ denotes the (sub)gradient with regard to f . However, note that $f(\mathbf{x}_n) = \langle f, \kappa(\cdot, \mathbf{x}_n) \rangle$. Applying the chain rule for differentiation, and recalling that

$$\frac{\partial}{\partial f} \langle f, \kappa(\cdot, \mathbf{x}_n) \rangle = \kappa(\cdot, \mathbf{x}_n),$$

and

$$\frac{\partial}{\partial f} \langle f, f \rangle = 2f,$$

we obtain that

$$\boxed{f_n = (1 - \mu_n \lambda) f_{n-1} - \mu_n \mathcal{L}'(y_n, f_{n-1}(\mathbf{x}_n)) \kappa(\cdot, \mathbf{x}_n)}, \quad (11.99)$$

where $\mathcal{L}'(y, z) := \frac{\partial}{\partial z} \mathcal{L}(y, z)$. If $\mathcal{L}(\cdot, \cdot)$ is not differentiable, $\mathcal{L}'(\cdot, \cdot)$ denotes any subgradient of $\mathcal{L}(\cdot, \cdot)$. Assuming, $f_0 = 0$ and applying (11.99) recursively, we obtain

$$f_n = \sum_{i=1}^n \theta_i \kappa(\cdot, \mathbf{x}_i). \quad (11.100)$$

Moreover, from (11.99) at time n , we obtain the equivalent time update of the corresponding coefficients, that is,

$$\boxed{\theta_n = -\mu_n \mathcal{L}'(y_n, f_{n-1}(\mathbf{x}_n)),} \quad (11.101)$$

$$\boxed{\theta_i^{\text{new}} = (1 - \mu_n \lambda) \theta_i, \quad i < n.} \quad (11.102)$$

Observe that for $\lambda = 0$, $\mu_n = \mu$ and $\mathcal{L}(\cdot, \cdot)$ being the squared loss ($\frac{1}{2}(y - f(\mathbf{x}))^2$), (11.101) and (11.102) break down into (11.92).

From the recursion in (11.99), it is readily apparent that a necessary condition that guarantees convergence is

$$\mu_n < \frac{1}{\lambda}, \quad \lambda > 0, \quad n = 1, 2, \dots$$

Let us now set $\mu_n = \mu < \frac{1}{\lambda}$; then, combining (11.100)–(11.102), it is easy to show recursively (Problem 11.12) that

$$f_n = - \sum_{i=1}^n \mu \mathcal{L}'(y_i, f_{i-1}(\mathbf{x}_i)) (1 - \mu \lambda)^{n-i} \kappa(\cdot, \mathbf{x}_i).$$

Observe that the effect of *regularization* is equivalent to imposing an *exponential forgetting factor* on past data. Thus, we can select a constant n_0 , sufficiently large, and keep in the expansion only the n_0 terms in the time window $[n - n_0 + 1, n]$. In this way, we achieve the propagation of a *fixed* number of parameters at every time instant, at the expense of an approximation/truncation error (Problem 11.13). The resulting scheme is summarized in Algorithm 11.2 [61].

Algorithm 11.2 (The NORMA algorithm).

- Initialize
 - Select λ and μ , $\mu < \frac{1}{\lambda}$.
- **For** $n = 1, 2, \dots$, **Do**
 - **If** $n = 1$ **then**
 - $f_0 = 0$
 - **else**
 - $f_{n-1}(\mathbf{x}_n) = - \sum_{i=\max\{1, n-n_0\}}^{n-1} \mu \mathcal{L}'(y_i, f_{i-1}(\mathbf{x}_i)) (1 - \mu \lambda)^{n-i-1} \kappa(\mathbf{x}_n, \mathbf{x}_i)$
 - **End If**
 - $\theta_n = -\mu \mathcal{L}'(y_n, f_{n-1}(\mathbf{x}_n))$
- **End For**

Note that if the functional form involves a constant bias term, that is, $\theta_0 + f(\mathbf{x})$, $f \in \mathbb{H}$, then the update of θ_0 follows the standard form

$$\theta_0(n) = \theta_0(n-1) - \mu \frac{\partial}{\partial \theta_0} \mathcal{L}(y_n, \theta_0 + f_{n-1}(\mathbf{x}_n))|_{\theta_0(n-1)},$$

with $\frac{\partial}{\partial \theta_0}$ the (sub)gradient with respect to θ_0 .

Classification: the hinge loss function

The hinge loss function was defined in (11.59) and its subgradient is given by

$$\mathcal{L}'_\rho(y, f(\mathbf{x})) = \begin{cases} -y, & yf(\mathbf{x}) \leq \rho, \\ 0, & \text{otherwise.} \end{cases}$$

Note that at the discontinuity, one of the subgradients is equal to 0, which is the one employed in the algorithm. This is plugged into Algorithm 11.2 in place of $\mathcal{L}'(y_n, f_{n-1}(\mathbf{x}_n))$ to result in

$$\theta_n = \mu \sigma_n y_n, \quad \sigma_n = \begin{cases} 1, & y_n f(\mathbf{x}_n) \leq \rho, \\ 0, & \text{otherwise,} \end{cases} \quad (11.103)$$

and if a bias term is present

$$\theta_0(n) = \theta_0(n-1) + \mu \sigma_n y_n.$$

If ρ is left as a free parameter, the online version of the ν -SVM [105] results, (Problem 11.14).

Regression: the linear ϵ -insensitive loss function

For this loss function, defined in (11.28), the subgradient is easily shown to be

$$\mathcal{L}'(y, f(\mathbf{x})) = \begin{cases} -\text{sgn}\{y - f(\mathbf{x})\}, & \text{if } |y - f(\mathbf{x})| > \epsilon, \\ 0, & \text{otherwise.} \end{cases}$$

Note that at the two points of discontinuity, the zero is a possible value for the subgradient and the corresponding update in Algorithm 11.2 becomes

$$\theta_n = \mu \text{sgn}\{y_n - f_{n-1}(\mathbf{x}_n)\}. \quad (11.104)$$

A variant of this algorithm is also presented in [61] by considering ϵ to be a free parameter and leave the algorithm to optimize with respect to it.

No doubt, any convex function can be used in place of $\mathcal{L}(\cdot, \cdot)$, such as the Huber, square ϵ -insensitive, and the squared error loss functions (Problem 11.15).

Error bounds and convergence performance

In [61], the performance analysis of an online algorithm, whose goal is the minimization of (11.96), is based on the cumulative instantaneous loss, after running the algorithm on N training samples, that is,

$$L_{cum}(N) := \sum_{n=1}^N \mathcal{L}(y_n, f_{n-1}(\mathbf{x}_n)) : \quad \text{Cumulative Loss.}$$

(11.105)

As already mentioned in Chapter 8, this is a natural criterion to test the performance. Note that f_{n-1} has been trained using samples up to and including time instants $n-1$, and it is tested against the sample (y_n, \mathbf{x}_n) , on which it was *not* trained. So $L_{cum}(N)$ can be considered as a measure of the generalization performance. A low value of $L_{cum}(N)$ is indicative of guarding against overfitting; see also [3, 130]. The following theorem has been derived in [61].

Theorem 11.4. Fix $\lambda > 0$ and $0 < \mu < \frac{1}{\lambda}$. Assume that $\mathcal{L}(\cdot, \cdot)$ is convex and satisfies the Lipschitz condition, that is,

$$|\mathcal{L}(y, z_1) - \mathcal{L}(y, z_2)| \leq c|z_1 - z_2|, \quad \forall y \in Y, \quad \forall z_1, z_2 \in \mathbb{R}$$

where Y is the respective domain of definition (e.g., $[-1, 1]$ or \mathbb{R}) and $c \in \mathbb{R}$. Also, let $\kappa(\cdot, \cdot)$ be bounded, that is,

$$\kappa(\mathbf{x}_n, \mathbf{x}_n) \leq B^2, \quad n = 1, 2, \dots, N.$$

Set $\mu_n = \frac{\mu}{\sqrt{n}}$. Then,

$$\frac{1}{N} \sum_{n=1}^N \mathcal{L}_{n,\lambda}(f_{n-1}) \leq J_{emp,\lambda}(f_*, N) + \mathcal{O}(N^{-1/2}), \quad (11.106)$$

where $\mathcal{L}_{n,\lambda}$ is defined in (11.98), $J_{emp,\lambda}$ is the regularized empirical loss in (11.97) and f_* is the minimizer, that is,

$$f_* = \arg \min_{f \in \mathbb{H}} J_{emp,\lambda}(f, N).$$

Such bounds have been treated in Chapter 8 in the context of regret analysis for online algorithms. In [61], more performance bounds are derived concerning the fixed learning rate case, $\mu_n = \mu$, appropriate for time-varying cases under specific scenarios. Also, bounds for the expected risk are provided.

Remarks 11.6.

- NORMA is similar to the ALMA algorithm presented in [43]; ALMA considers a time-varying step-size and regularization is imposed via a projection of the parameter vector on a closed ball. This idea of projection is similar with that used in PEGASOS discussed in Chapter 8, [106]. As a matter of fact, PEGASOS without the projection step coincides with NORMA. The difference lies in the choice of the step-size.

In the ALMA algorithm, the possibility of normalizing the input samples and the unknown parameter vector via different norms, that is, $\|\cdot\|_q$ and $\|\cdot\|_p$ (p and q being dual norms) in the context of the so-called p -norm margin classifiers is exploited. p -norm algorithms can be useful in learning sparse hyperplanes (see also [62]).

- For the special case of $\rho = 0$ and $\lambda = 0$, the NORMA using the hinge loss breaks down to the *kernel perceptron*, to be discussed in more detail in Chapter 18.

11.12.3 THE KERNEL APSM ALGORITHM

The APSM algorithm was introduced as an alternative path for parameter estimation in machine learning tasks, which springs from the classical POCS theory; it is based solely on (numerically robust) projections. Extending its basic recursions (8.39) to the case of a RKH space of functions (as said there, the theory holds for a general Hilbert space), we obtain

$$f_n = f_{n-1} + \mu_n \left(\frac{1}{q} \sum_{k=n-q+1}^n P_k(f_{n-1}) - f_{n-1} \right), \quad (11.107)$$

where the weights for the convex combination of projections are set (for convenience) equal, that is, $\omega_k = \frac{1}{q}$, $k \in [n-q+1, n]$. P_k is the projection operator on the respective convex set. Two typical examples of convex sets for regression and classification are described next.

Regression

In this case, as treated in Chapter 8, a common choice is to project on hyperslabs, defined by the points $(y_n, \kappa(\cdot, \mathbf{x}_n))$ and the parameter ϵ , which controls the width of the hyperslab; its value depends on the noise variance, without being very sensitive to it. For such a choice, we get (see (8.34)),

$$P_k(f_{n-1}) = f_{n-1} + \beta_k \kappa(\cdot, \mathbf{x}_k), \quad (11.108)$$

with

$$\beta_k = \begin{cases} \frac{y_k - \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle - \epsilon}{\kappa(\mathbf{x}_k, \mathbf{x}_k)}, & \text{if } \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle - y_k < -\epsilon, \\ 0, & \text{if } |\langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle - y_k| \leq \epsilon, \\ \frac{y_k - \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle + \epsilon}{\kappa(\mathbf{x}_k, \mathbf{x}_k)}, & \text{if } \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle - y_k > \epsilon. \end{cases} \quad (11.109)$$

Recall from [Section 8.6](#) that, the hyperslab defined by $(y_n, \kappa(\cdot, \mathbf{x}_n))$ and ϵ is the respective 0-level set of the linear ϵ -insensitive loss function $\mathcal{L}(y_n, f(\mathbf{x}_n))$ defined in [\(11.28\)](#).

Classification

In this case, a typical choice is to project on the half-space, [Section 8.6.2](#), formed by the hyperplane

$$y_n \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_n) \rangle = \rho,$$

and the corresponding projection operator becomes

$$P_k(f_{n-1}) = f_{n-1} + \beta_k \kappa(\cdot, \mathbf{x}_k),$$

where

$$\beta_k = \begin{cases} \frac{\rho - y_k \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle}{\kappa(\mathbf{x}_k, \mathbf{x}_k)}, & \text{if } \rho - y_k \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle > 0, \\ 0, & \text{otherwise.} \end{cases} \quad (11.110)$$

Recall that the corresponding half-space is the 0-level set of the *hinge loss* function, $\mathcal{L}_\rho(y_n, f(\mathbf{x}_n))$ defined in [\(11.59\)](#).

Thus, for both cases, regression as well as classification, the recursion in [\(11.107\)](#) takes a common formulation

$$f_n = f_{n-1} + \mu_n \sum_{k=n-q+1}^n \beta_k \kappa(\cdot, \mathbf{x}_k), \quad (11.111)$$

where $1/q$ has been included in β_k . Applying the above recursively from $f_0 = 0$, f_n can be written as an expansion in the form of [\(11.100\)](#) and the corresponding updating of the parameters, θ_i , become

$$\theta_n = \mu_n \beta_n, \quad (11.112)$$

$$\theta_i^{\text{new}} = \theta_i + \mu_n \beta_i, \quad i = n-q+1, \dots, n-1, \quad (11.113)$$

$$\theta_i^{\text{new}} = \theta_i, \quad i \leq n-q, \quad (11.114)$$

where for the computation in [\(11.110\)](#) and [\(11.109\)](#) the following equality has been employed,

$$\begin{aligned} \langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle &= \sum_{i=1}^{n-1} \theta_i \langle \kappa(\cdot, \mathbf{x}_i), \kappa(\cdot, \mathbf{x}_k) \rangle \\ &= \sum_{i=1}^{n-1} \theta_i \kappa(\mathbf{x}_i, \mathbf{x}_k). \end{aligned} \quad (11.115)$$

Comparing [\(11.112\)-\(11.114\)](#) with [\(11.101\)-\(11.102\)](#) and setting $q = 1$, (for APSM) and $\lambda = 0$ (for NORMA), we see that the parameter updates look very similar; only the values of the involved variables are obtained differently. As we will see in the simulations section, this difference can have a significant effect on the respective performances in practice.

Sparsification: Sparsification of the APSM can be achieved either by regularization or via the use of a dictionary, in a similar way as explained before in [Section 11.12.1](#).

For regularization, the projection path is adopted, which is in line with the rationale that has inspired the method. In other words, we impose on the desired sequence of hypothesis the following constraint:

$$\|f_n\| \leq \delta.$$

Recall that if one has to perform a minimization of a loss function under this constraint, it is equivalent to performing a regularization of the loss (as in (11.98)) for appropriate choices of λ (δ) (see also Chapter 3). Under the previous constraints, the counterpart of (11.107) becomes

$$f_n = P_{B[0,\delta]} \left(f_{n-1} + \mu_n \left(\frac{1}{q} \sum_{k=n-q+1}^n P_k(f_{n-1}) - f_{n-1} \right) \right),$$

or

$$f_n = P_{B[0,\delta]} \left(f_{n-1} + \mu_n \sum_{k=n-q+1}^n \beta_k \kappa(\cdot, \mathbf{x}_k) \right), \quad (11.116)$$

where $P_{B[0,\delta]}$ is the projection on the closed ball defined as $B[0, \delta]$ (Example 8.1),

$$B[0, \delta] = \{f \in \mathbb{H} : \|f\| \leq \delta\},$$

and the projection is given by (8.14)

$$P_{B[0,\delta]}(f) = \begin{cases} f, & \text{if } \|f\| \leq \delta, \\ \frac{\delta}{\|f\|} f, & \text{if } \|f\| > \delta, \end{cases}$$

which together with (11.116) adds an extra step in the updates (11.112)–(11.113), that is,

$$\theta_i = \frac{\delta \theta_i}{\|f_n\|}, \quad i = 1, 2, \dots, n,$$

whenever needed. The computation of $\|f_n\|$ can be performed recursively, and adds no extra kernel evaluations (which is the most time-consuming part of all the algorithms presented so far), to those used in the previous steps of the algorithm (Problem 11.16). Note that the projection step is applied if, $\delta < \|f_n\|$; thus, after a repeated application of the projection (multiplications of smaller than one values), renders the parameters associated with samples of the “remote” past to small values and they can be neglected. This leads us to the same conclusions as in NORMA; hence, only the most recent terms in a time window $[n - n_0 + 1, n]$ can be kept and updated [110, 111].

The other alternative for sparsification is via the use of dictionaries. In this case, the expansion of each hypothesis takes place in terms of the elements in the dictionary comprising \mathbf{u}_m , $m = 1, 2, \dots, M_{n-1}$. Under such a scenario, (11.115) becomes

$$\langle f_{n-1}, \kappa(\cdot, \mathbf{x}_k) \rangle = \sum_{m=1}^{M_{n-1}} \theta_m \kappa(\mathbf{x}_k, \mathbf{u}_m), \quad (11.117)$$

as only elements included in the dictionary are involved. The resulting scheme is given in Algorithm 11.3, [109].

Algorithm 11.3 (The quantized kernel APSM).

- Initialization
 - $\mathcal{D} = \emptyset$.
 - Select $q \geq 1, \delta$; the quantization level.
 - $d(\mathbf{x}, \emptyset) := \Delta > \delta, \forall \mathbf{x}$.

- **FOR** $n = 1, 2, \dots$, **Do**
- $d(\mathbf{x}_n, \mathcal{D}) = \inf_{\mathbf{u}_k \in \mathcal{D}} \|\mathbf{x}_n - \mathbf{u}_k\| = \|\mathbf{x}_n - \mathbf{u}_{l_0}\|$, for some $l_0 \in \{1, 2, \dots, M\}$
- **If** $d(\mathbf{x}_n, \mathcal{D}) > \delta$ **then**
 - $\mathbf{u}_{M+1} = \mathbf{x}_n$; includes the new observation in the dictionary.
 - $\mathcal{D} = \mathcal{D} \cup \{\mathbf{u}_{M+1}\}$
 - $\boldsymbol{\theta} = [\boldsymbol{\theta}, \theta_{M+1}]$, $\theta_{M+1} := 0$; increases the size of $\boldsymbol{\theta}$ and initializes the new weight to zero.
 - $\mathcal{J} := \{\max\{1, M - q + 2\}, M + 1\}$; identifies the q most recent samples in the dictionary.
- **Else**
 - **If** $l_0 \geq \max\{1, M - q + 1\}$, **then**
 - $\mathcal{J} = \{\max\{1, M - q + 1\}, \dots, M\}$; identifies the q most recent samples in the dictionary, which also includes l_0 .
 - **Else**
 - $\mathcal{J} = \{l_0, M - q + 2, \dots, M\}$; it takes care \mathbf{u}_{l_0} to be in \mathcal{J} , if it is not in the q most recent ones.
 - **End If**
- **End If**
- Compute β_k , $k \in \mathcal{J}$; (11.117) and (11.110) or (11.109)
- Select μ_n
- $\theta_i = \theta_i + \mu_n \beta_i$, $i \in \mathcal{J}$
- **If** $d(\mathbf{x}_n, \mathcal{D}) > \delta$ **then**
 - $M = M + 1$
- **End If**
- **END For**

The algorithm returns the dictionary and $\boldsymbol{\theta}$ and

$$f_n = \sum_{i=1}^M \theta_i \kappa(\cdot, \mathbf{u}_i)$$

$$\hat{y}_n = \sum_{i=1}^M \theta_i \kappa(\mathbf{x}_n, \mathbf{u}_i)$$

Remarks 11.7.

- Note that for the case of hyperslabs, if one sets $\epsilon = 0$ and $q = 1$ the *normalized KLMS* results.
- If a bias term needs to be included, this is achieved by the standard extension of the dimensionality of the space by one. This results in replacing $\kappa(\mathbf{x}_n, \mathbf{x}_n)$ with $1 + \kappa(\mathbf{x}_n, \mathbf{x}_n)$ in the denominator in (11.110) or (11.109) and the bias updating is obtained as (Problem 11.18)

$$\theta_0(n) = \theta_0(n-1) + \mu_n \sum_{k=n-q+1}^n \beta_k.$$

- For the selection of μ_n , one can employ a constant or a time-varying sequence, as in any algorithm of the stochastic gradient rationale. Moreover, for the case of APSM, there is a theoretically computed interval that guarantees convergence; that is, μ_n must lie in $(0, 2M_n)$, where,

$$M_n = \frac{\sum_{i \in \mathcal{J}} q \beta_i^2 \kappa(\mathbf{u}_i, \mathbf{u}_i)}{\sum_{i \in \mathcal{J}} \sum_{j \in \mathcal{J}} \beta_i \beta_j \kappa(\mathbf{u}_i, \mathbf{u}_j)}$$

if

$$\sum_{i \in \mathcal{J}} \sum_{j \in \mathcal{J}} \beta_i \beta_j \kappa(\mathbf{u}_i, \mathbf{u}_j) \neq 0.$$

Otherwise, $M_n = 1$.

- Besides the three basic schemes described before, kernelized versions of the RLS and APA have been proposed, see, for example, [36, 114]. However, these schemes need an inversion of a matrix of the order of the dictionary (RLS) and of the order of q (APA). An online kernel RLS-type algorithm, from a Bayesian perspective, is given in [127]. For an application of online kernel-based algorithms in time series prediction, see, for example, [96].
- One of the major advantages of the projection-based algorithms is the fairly easy way to accommodate constraints; in some cases even nontrivial constraints. This also applies to the KAPSM. For example, in [112] the nonlinear robust beamforming is treated. This task corresponds to an infinite set of linear inequality constraints; in spite of that, it can be solved in the context of APSM with a linear complexity, with respect to the number of unknown parameters per time update. Extensions to multiregression with application to MIMO nonlinear channel equalization is presented in [113].
- All that has been said concerning convergence in Chapter 8 about APSM can be extended and it is valid for the case of RKH spaces.

Example 11.5. Nonlinear Channel Equalization

In this example, the performance of the previously described online kernel algorithms is studied, in the context of a nonlinear equalization task.

Let an information sequence, denoted as s_n , be transmitted to a nonlinear channel. For example, such channels are typical in satellite communications. The nonlinear channel is simulated as a combination of a linear one, whose input is the information sequence, that is,

$$t_n = -0.9s_n + 0.6s_{n-1} - 0.7s_{n-2} + 0.2s_{n-3} + 0.1s_{n-4},$$

followed by a memoryless nonlinearity (see Section 11.3), that is,

$$t'_n = 0.15t_n^2 + 0.03t_n^3.$$

In the sequel, a white noise sequence is added,

$$x_n = t'_n + \eta_n,$$

where for our example, η_n is a zero-mean white Gaussian noise, with $\sigma_\eta^2 = 0.02$, corresponding to a signal-to-noise ratio of 15 dBs. The input information sequence was generated according to a zero-mean Gaussian with variance equal to $\sigma_s^2 = 0.64$.

The received sequence at the receiver end is the sequence x_n . The task of the equalizer is to provide an estimate of the initially transmitted information sequence s_n , based on x_n . As is the case for the linear equalizer,¹³ Chapter 4, at each time instant, l successive samples are presented as input to the

¹³ There, the input was denoted as u_n ; here, because we do not care if the input is a process or not, we use x_n , to be in line with the notation used in this chapter.

equalizer, which form the input vector \mathbf{x}_n . In our case, we chose $l = 5$. Also, in any equalization scheme, a delay, D , is involved to account for the various delays associated with the communications system; that is, at time n , an estimate of s_{n-D} is obtained. Thus, the output of the equalizer is the nonlinear mapping

$$\hat{s}_{n-D} = f(\mathbf{x}_n).$$

In the training phase, we assume the transmitted symbols are known and our learning algorithms have available the training sequence, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots$, where $y_n = s_{n-D}$. For this example, the best (after experimentation) delay was found to be $D = 2$.

The quantized KLMS was used with $\mu = 1/2$, the KAPSM with (fixed) $\mu_n = 1/2$, $\epsilon = 10^{-5}$ (the algorithm is fairly insensitive to the choice of ϵ) and $q = 5$. For the KRLS, the ALD accuracy parameter $v = 0.1$ [36] was used. In all algorithms, the Gaussian kernel has been employed with $\sigma = 5$. The LS loss function was employed, and the best performance for the NORMA was obtained for $\lambda = 0$, which makes it equivalent to the KLMS, when no sparsification is applied. However, in order to follow the suggestion in [61] and to make a fair comparison with the other three methods, we carefully tuned the regularization parameter λ . This allows the use of the sliding window method for keeping only a finite number of processing samples (i.e., n_0) at each time instant. In our experiments, we set the window size of NORMA as $n_0 = 80$, to keep it comparable to the dictionary sizes of QKLMS, QKAPSM, and KRLS, and found that the respective λ is equal to $\lambda = 0.01$; also, $\mu_n = 1/4$ was found to provide the best possible performance.

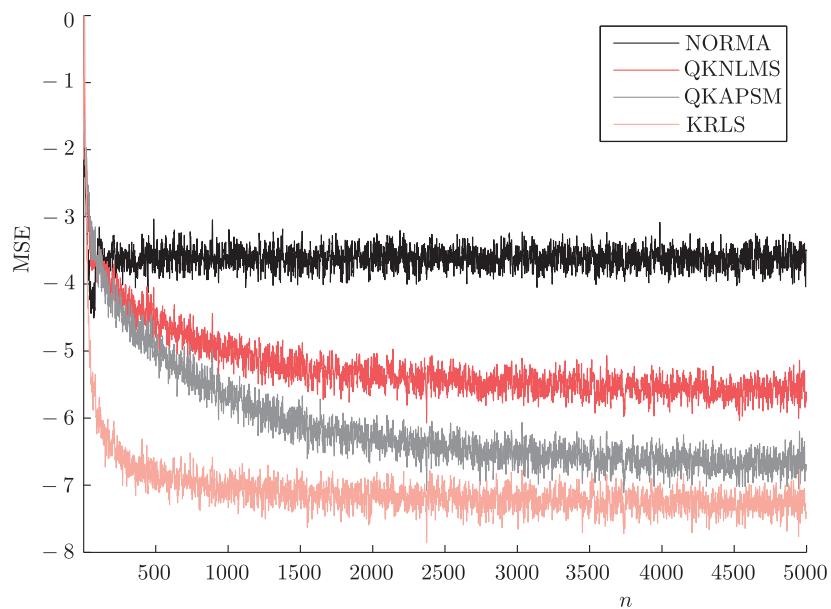
Figure 11.19 shows the obtained MSE averaged over 1000 realizations, in dBs ($10 \log_{10}(e_n^2)$) as a function of iterations. The improved performance of the KAPSM compared to the KLMS is clearly seen, due to the data reuse rationale, at a slightly higher computational cost. The KRLS has clearly superior convergence performance, converging faster and at lower error rates, albeit at higher complexity. The NORMA has a rather poor performance. For all cases, the parameters used were optimized via extensive simulations. The superior performance of the KAPSM, in the context of a classification task, when the property sets are built around half-spaces, compared to the NORMA, have also been demonstrated in [110], both in stationary as well as in time-varying environments. Figure 11.20 corresponds to the case where, after convergence, the channel suddenly changes to

$$t_n = 0.8s_n - 0.7s_{n-1} + 0.6s_{n-2} - 0.2s_{n-3} - 0.2s_{n-4},$$

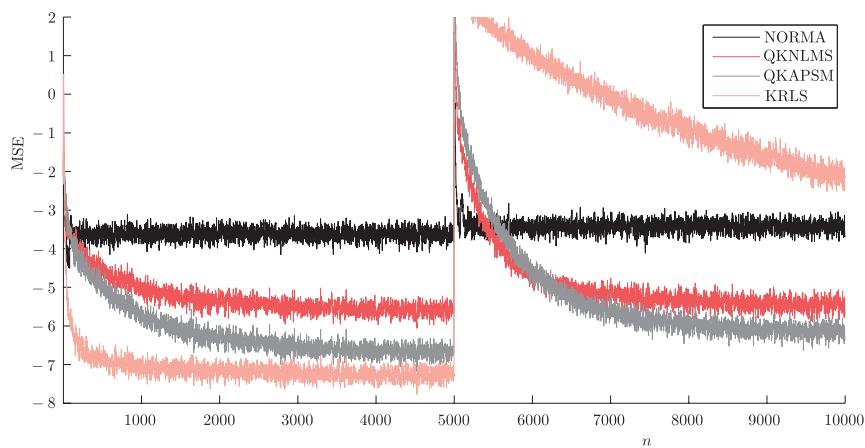
and

$$x_n = 0.12t_n^2 + 0.02t_n^3 + \eta_n.$$

Observe that the KRLS really has a problem for tracking this time variation. The difficulty of KRLS in tracking time variations has also been observed and discussed in [127], where a KRLS is also derived via the Bayesian framework and an exponential forgetting factor is employed to cope with variations. Observe that, just after the jump of the system, the KLMS tries to follow the change faster compared to the KAPSM. This is natural, as the KAPSM employs past data reuse, which somehow slows down its agility to track; however, soon after, it recovers and leads to improved performance.

**FIGURE 11.19**

Mean-square error in dBs, as a function of iterations, for the data of [Example 11.5](#).

**FIGURE 11.20**

MSE in dBs, as a function of iterations, for the time-varying nonlinear channel of [Example 11.5](#).

11.13 MULTIPLE KERNEL LEARNING

A major issue in all kernel-based algorithmic procedures is the selection of a suitable kernel as well as the computation of its defining parameters. Usually, this is carried out via cross validation, where a number of different kernels are used on a separate, from the training data, validation set (see Chapter 3 for different methods concerning validation) and the one with the best performance is selected. It is obvious that this is not a universal approach, it is time-consuming and definitely is not theoretically appealing. The ideal would be to have a set of different kernels (this also includes the case of the same kernel with different parameters) and let the optimization procedure decide how to choose the proper kernel, or the proper combination of kernels. This is the scope of an ongoing activity, which is usually called *multiple kernel learning* (MKL). To this end, a variety of MKL methods have been proposed to treat several kernel-based algorithmic schemes. A complete survey of the field is outside the scope of this book. Here, we will provide a brief overview of some of the major directions in MKL methods that relate to the content of this chapter. The interested reader is referred to [47] for a comparative study of various techniques.

One of the first attempts to develop an efficient MKL scheme is the one presented in [65], where the authors considered a linear combination of kernel matrices, that is, $K = \sum_{m=1}^M a_m K_m$. Because we require the new kernel matrix to be positive definite, it is reasonable to impose in the optimization task some additional constraints. For example, one may adopt the general constraint $K \geq 0$ (the inequality indicating semidefinite positiveness), or a more strict one, for example, $a_m > 0$, for all $m = 1, \dots, M$. Furthermore, one needs to bound the norm of the final kernel matrix. Hence, the general MKL SVM task can be cast as

$$\begin{aligned} \text{minimize with respect to } K & \quad \omega_C(K), \\ \text{subject to} & \quad K \geq 0, \\ & \quad \text{trace}\{K\} \leq c, \end{aligned} \tag{11.118}$$

where $\omega_C(K)$ is the solution of the dual SVM task, given in (11.75)–(11.77), which can be written in a more compact form as

$$\omega_C(K) = \max_{\lambda} \left\{ \lambda^T \mathbf{1} - \frac{1}{2} \lambda^T G(K) \lambda : 0 \leq \lambda_i \leq C, \lambda^T \mathbf{y} = 0 \right\},$$

with each element of $G(K)$ given as $[G(K)]_{i,j} = [K]_{i,j} y_i y_j$. λ denotes the vector of the Lagrange multipliers and $\mathbf{1}$ is the vector having all its elements equal to one. In [65], it is shown how (11.118) can be transformed to a semidefinite programming optimization (SDP) task and solved accordingly.

Another path that has been exploited by many authors is to assume that the modeling nonlinear function is given as a summation

$$\begin{aligned} f(\mathbf{x}) &= \sum_{m=1}^M a_m f_m(\mathbf{x}) = \sum_{m=1}^M a_m \langle f_m, \kappa_m(\cdot, \mathbf{x}) \rangle_{\mathbb{H}_m} + b \\ &= \sum_{m=1}^M \sum_{n=1}^N \theta_{m,n} a_m \kappa_m(\mathbf{x}, \mathbf{x}_n) + b, \end{aligned}$$

where each one of the functions, f_m , $m = 1, 2, \dots, M$, lives in a different RKHS, \mathbb{H}_m . The respective composite kernel matrix, associated with a set of training data, is given by $K = \sum_{m=1}^M a_m^2 K_m$, where

K_1, \dots, K_M are the kernel matrices of the individual RKH spaces. Hence, assuming a data set $\{(y_n, \mathbf{x}_n), n = 1, \dots, N\}$, the MKL learning task can be formulated as follows:

$$\min_f \sum_{n=1}^N \mathcal{L}(y_n, f(\mathbf{x}_n)) + \lambda \Omega(f), \quad (11.119)$$

where \mathcal{L} represents a loss function and $\Omega(f)$ the regularization term. There have been two major trends following this rationale. The first one gives priority toward a sparse solution, while the second aims at improving performance.

In the context of the first trend, the solution is constrained to be sparse, so that the kernel matrix is computed fast and the strong similarities between the data set are highlighted. Moreover, this rationale can be applied to the case where the type of kernel has been selected beforehand and the goal is to compute the optimal kernel parameters. One way (e.g., [117], [4]) is to employ a regularization term of the form $\Omega(f) = \left(\sum_{m=1}^M a_m \|f_m\|_{\mathbb{H}_m} \right)^2$, which has been shown to promote sparsity among the set $\{a_1, \dots, a_M\}$, as it is associated to the group LASSO, when the LS loss is employed in place of \mathcal{L} (see Chapter 10).

In contrast to the sparsity promoting criteria, another trend (e.g., [29, 63, 94]) revolves around the argument that, in some cases, the sparse MKL variants may not exhibit improved performance compared to the original learning task. Moreover, some data sets contain multiple similarities between individual data pairs that cannot be highlighted by a single type of kernel, but require a number of different kernels to improve learning. In this context, a regularization term of the form $\Omega(f) = \sum_{m=1}^M a_m \|f_m\|_{\mathbb{H}_m}^2$ is preferred. There are several variants of these methods that either employ additional constraints to the task (e.g., $\sum_{m=1}^M a_m = 1$), or define the summation of the spaces a little differently (e.g., $f(\cdot) = \sum_{m=1}^M \frac{1}{a_m} f_m(\cdot)$). For example, in [94] the authors reformulate (11.119) as follows:

$$\begin{aligned} & \text{minimize with respect to } \mathbf{a} && J(\mathbf{a}), \\ & \text{subject to} && \sum_{m=1}^M a_m = 1, \\ & && a_m \geq 0, \end{aligned} \quad (11.120)$$

where

$$J(\mathbf{a}) = \min_{f_i, M, \xi, b} \left\{ \begin{array}{l} \frac{1}{2} \sum_{m=1}^M \frac{1}{a_m} \|f_m\|_{\mathbb{H}_m}^2 + C \sum_{n=1}^N \xi_n, \\ y_i \left(\sum_{m=1}^M f_m(\mathbf{x}_n) + b \right) \geq 1 - \xi_n, \\ \xi_n \geq 0. \end{array} \right\}$$

The optimization is cast in RKH spaces; however, the problem is always formulated in such a way so that the kernel trick can be mobilized.

11.14 NONPARAMETRIC SPARSITY-AWARE LEARNING: ADDITIVE MODELS

We have already pointed out that the representer theorem, as summarized by (11.17), provides an approximation of a function, living in an RKHS, in terms of the respective kernel centered at the points $\mathbf{x}_1, \dots, \mathbf{x}_N$. However, we know that the accuracy of any interpolation/approximation method depends on the number of points, N . Moreover, as discussed in Chapter 3, how large or small N needs to be

depends heavily on the dimensionality of the space, exhibiting an exponential dependence on it (curse of dimensionality); basically, one has to fill in the input space with “enough” data in order to be able to “learn” with good enough accuracy the associated function.¹⁴ In [Chapter 7](#), the naive Bayes classifier was discussed; the essence behind this method is to consider each dimension of the input random vectors, $\mathbf{x} \in \mathbb{R}^l$, *individually*. Such a path breaks the problem into a number, l , of *one-dimensional* tasks. The same idea runs across the so-called *additive* models approach.

According to the additive models rationale, the unknown function is constrained within the family of *separable* functions, that is,

$$f(\mathbf{x}) = \sum_{i=1}^l \phi_i(x_i) : \quad \text{Additive Model}, \quad (11.121)$$

where $\mathbf{x} = [x_1, \dots, x_l]^T$. Recall that a special case of such expansions is the linear regression, where $f(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$.

We will further assume that each one of the functions, $\phi_i(\cdot)$, belongs to an RKHS, \mathbb{H}_i defined by a respective kernel, $\kappa_i(\cdot, \cdot)$,

$$\kappa_i(\cdot, \cdot) : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}.$$

Let the corresponding norm be denoted as $\|\cdot\|_i$. For the regularized LS cost [\[95\]](#), the optimization task is now cast as

$$\text{minimize with respect to } \boldsymbol{\theta} \quad \frac{1}{2} \sum_{n=1}^N (y_n - f(\mathbf{x}_n))^2 + \lambda \sum_{i=1}^l \|\phi_i\|_i, \quad (11.122)$$

$$\text{s.t.} \quad f(\mathbf{x}) = \sum_{i=1}^l \phi_i(x_i). \quad (11.123)$$

If one plugs (11.123) into (11.122) and following arguments similar to those used in [Section 11.6](#), it is readily obtained that we can write

$$\hat{\phi}_i(\cdot) = \sum_{n=1}^N \theta_{i,n} \kappa_i(\cdot, x_{i,n}), \quad (11.124)$$

where $x_{i,n}$ is the i th component of \mathbf{x}_n . Moving along the same path as that adopted in [Section 11.7](#), the optimization can be rewritten in terms of

$$\boldsymbol{\theta}_i = [\theta_{i,1}, \dots, \theta_{i,N}]^T, \quad i = 1, 2, \dots, l,$$

as

$$\{\hat{\theta}_i\}_{i=1}^l = \arg \min_{\{\boldsymbol{\theta}_i\}_{i=1}^l} J(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_l),$$

where

$$J(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_l) := \frac{1}{2} \left\| \mathbf{y} - \sum_{i=1}^l \mathcal{K}_i \boldsymbol{\theta}_i \right\|^2 + \lambda \sum_{i=1}^l \sqrt{\boldsymbol{\theta}_i^T \mathcal{K}_i \boldsymbol{\theta}_i}, \quad (11.125)$$

and \mathcal{K}_i , $i = 1, 2, \dots, l$, are the respective $N \times N$ kernel matrices

¹⁴ Recall from the discussion in [Section 11.10.4](#) that the other factor that ties accuracy and N together is the rate of variation of the function.

$$\mathcal{K}_i := \begin{bmatrix} \kappa_i(x_{i,1}, x_{i,1}) & \cdots & \kappa_i(x_{i,1}, x_{i,N}) \\ \vdots & \ddots & \vdots \\ \kappa_i(x_{i,N}, x_{i,1}) & \cdots & \kappa_i(x_{i,N}, x_{i,N}) \end{bmatrix}.$$

Observe that (11.125) is a (weighted) version of the *group LASSO*, defined in Section 10.3. Thus, the optimization task enforces sparsity by pushing some of the vectors θ_i , to zero values. Any algorithm developed for the group LASSO can be employed here as well, see, for example, [10, 95].

Besides the squared error loss, other loss functions can also be employed. For example, in [95] the *logistic regression* model is also discussed. Moreover, if the separable model in (11.121) cannot adequately capture the whole structure of f , models involving combination of components can be considered, such as the ANOVA model, e.g. [67].

The analysis of variance (ANOVA) is a method in statistics to analyse interactions among variables. According to this technique, a function $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^l$, $l > 1$, is decomposed into a number of terms; each term is given as a sum of functions involving a *subset* of the components of \mathbf{x} . From this point of view, separable functions of the form in (11.121) are a special case of an ANOVA decomposition. A more general decomposition would be

$$f(\mathbf{x}) = \theta_0 + \sum_{i=1}^l \phi_i(x_i) + \sum_{i < j=1}^l \phi_{ij}(x_i, x_j) + \sum_{i < j < k=1}^l \phi_{ijk}(x_i, x_j, x_k), \quad (11.126)$$

and this can be generalized to involve larger number of components. Comparing (11.126) with (11.2), ANOVA decomposition can be considered as a generalization of the polynomial expansion. The idea of ANOVA decomposition has already been used to decompose kernels, $\kappa(\mathbf{x}, \mathbf{y})$, in terms of other kernels that are functions of a subset of the involved components of \mathbf{x} and \mathbf{y} , giving rise to the so-called ANOVA kernels, [116].

Techniques involving sparse additive kernel-based models have been used in a number of applications, such as matrix and tensor completion, completion of gene expression, kernel-based dictionary learning, network flow, and spectrum cartography; see [10] for a related review.

11.15 A CASE STUDY: AUTHORSHIP IDENTIFICATION

Text mining is the part of data mining that analyzes textual data to detect, extract, represent, and evaluate patterns that appear in texts and can be transformed into real-world knowledge. A few examples of text mining applications include spam e-mail detection, topic-based classification of texts, sentiment analysis in texts, text authorship identification, text indexing, and text summarization. In other words, the goal can be on detecting basic morphology idiosyncrasies in a text that identify its author (authorship identification), on identifying complexly expressed emotions (sentiment analysis), or on condensing redundant information from multiple texts to a concise summary (summarization).

In this section, our focus will be on the case of authorship identification, for example, [115], which is a special case of text classification. The task is to determine the author of a given text, given a set of training texts labeled with their corresponding author. To fulfill this task, one needs to represent and act upon textual data. Thus, the first decision related to text mining is how one represents the data.

It is very common to represent texts in a vector space, following the vector space model or VSM [98]. According to VSM, a text document, T , is represented by a set of terms w_i , $0 \leq i \leq k$, $k \in \mathbb{N}$ each

mapped to a dimension of the vector $\mathbf{t} = [t_1, t_2, \dots, t_k]^T \in \mathbb{R}^k$. The elements, t_i , in each dimension, indicate the importance of the corresponding term w_i when describing the document. For example, each w_i can be one word in the vocabulary of a language. Thus, in practical applications, k can be very large, as large as the number of words that are considered sufficient for the specific task of interest. Typical examples of k are of the order of 10^5 .

Widely used approaches to assign values to t_i are as follows:

- *Bag-of-words, frequency approach:* The bag-of-words approach to text relies on the assumption that a text T is nothing more than a histogram of the words it contains. Thus, in the bag-of-words assumption, we do not care about the order of the words in the original text. What we care about is how many times term w_i appears in the document. Thus, t_i is assigned the number of occurrences of w_i in T .
- *Bag-of-words, boolean approach:* The boolean approach to the bag-of-words VSM approach restricts the t_i values such as, $t_i \in \{0, 1\}$. t_i is assigned a value of 1, if w_i was found at least once in T , otherwise $t_i = 0$.

The bag-of-words approaches have proven efficient, for example, in early spam filtering applications, where individual words were clear indicators of whether an e-mail is spam. However, the spammers soon adapted and changed the texts they sent out by breaking words using spaces: the word “pills” would be replaced by “p i l l s.” The bag-of-words approach then needed to apply preprocessing to deal with such cases, or even to deal with changes in different word forms (via word stemming or lemmatization) or spelling mistakes (by using dictionaries to correct the errors).

An alternative representation, with high resilience to noise, is that of the *character n-grams*. This representation is based on character subsequences of a text of length n (also called the order of an n -gram). To represent a text using n -grams, we split the text T in (usually contiguous) groups of characters of length n , that are then mapped to dimensions in the vector space (bag-of- n -grams). Below we give an example on how n -grams can be extracted from a given text.

Example 11.6. Given the input text $T = \text{A_fine_day_today}$, the character and word 2-grams ($n = 2$) would be as follows:

- Unique character 2-grams: “A_”, “_f”, “fi”, “in”, “ne”, “e_”, “_d”, “da”, “ay”, “y_”, “_t”, “to”, “od”
- Unique word 2-grams: “A_fine”, “fine_day”, “day_today”

Observe that “ay” and “da” appear twice in the phrase, yet only once in the sequence, since each n -gram is uniquely represented. Even though this approach has proven very useful [18], still the sequence of the n -grams is not exploited.

In the following, we describe a more complex representation, which takes into account the sequence of n -grams, and at the same time allows for noise. The representation is termed *n-gram graph* [44]. An n -gram graph represents the way on how n -grams *co-occur* in a text.

Definition 11.4. If $S = \{S_1, S_2, \dots\}$, $S_k \neq S_l$, for $k \neq l$, $k, l \in \mathbb{N}$ is the set of *distinct n-grams* extracted from a text, T , and S_i is the i th extracted n -gram. Then, $G = \{V, E, W\}$ is a graph where $V = S$ is the set of vertices v , E is the set of edges e of the form $e = \{v_1, v_2\}$, and $W : E \rightarrow \mathbb{R}$ is a function assigning a weight w_e to every edge.

To generate an n -gram graph, we first extract the unique n -grams from a text, creating one vertex for each. Then, given a user-defined parameter distance, D , we consider the n -grams that are found *within* a distance, D , of each other in the text; these are considered *neighbors* and their respective vertices are

connected with an edge. For each edge, a weight is assigned. The edge weighting function that is most commonly used in n -gram graphs is the number of co-occurrences of the linked (in the graph) n -grams in the text.

Two examples of 2-gram graphs, with directed neighborhood edges, with $D = 2$ are presented in Figures 11.21 and 11.22.

Between two n -gram graphs, G^i and G^j , there exists a symmetric, normalized similarity function called *value similarity* (VS). This measure quantifies the ratio of common edges between two graphs, taking into account the ratio of weights of common edges. In this measure, each matching edge e , having weights w_e^i and w_e^j , in graphs G^i and G^j , respectively, contributes to VS the amount, $\frac{\text{VR}(e)}{\max(|G^i|, |G^j|)}$, where, $|\cdot|$ indicates cardinality with respect to the number of edges and

$$\text{VR}(e) := \frac{\min(w_e^i, w_e^j)}{\max(w_e^i, w_e^j)}. \quad (11.127)$$

Not matching edges have no contribution, (consider that for an edge $e \notin G^i(G^j)$ we define $w_e^i = 0$ ($w_e^j = 0$)). The equation indicates that the *ValueRatio* takes values in $[0, 1]$, and is symmetric. Thus, the full equation for VS is

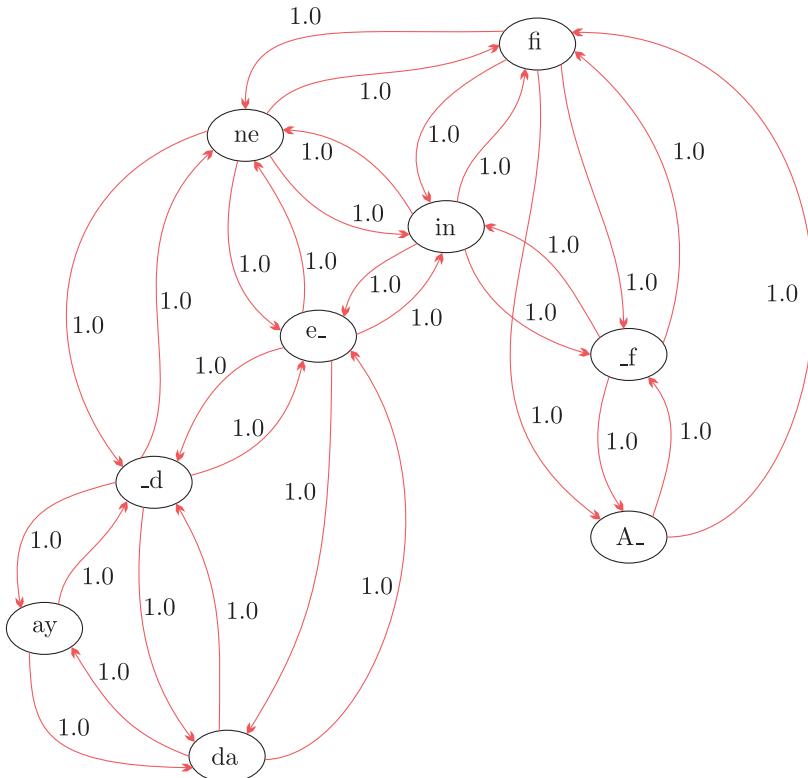
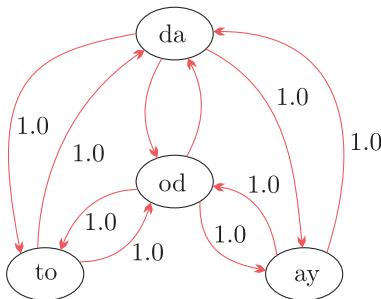


FIGURE 11.21

The 2-gram graph ($n = 2$), with $D = 2$ of the string "A fine day."

**FIGURE 11.22**

The 2-gram graph ($n = 2$), with $D = 2$ of the string “today.”

$$\text{VS}(G^i, G^j) = \frac{\sum_{e \in G^i} \min(w_e^i, w_e^j)}{\max(|G^i|, |G^j|)} \quad (11.128)$$

VS takes a value close to 1 for graphs that share many edges with similar weights. A value of $\text{VS} = 1$ indicates a perfect match between the compared graphs. For the two graphs shown in Figures 11.21 and 11.22, the calculation of the VS returns a value of $\text{VS} = 0.067$ (verify it).

For our case study, VS will be used to build an SVM kernel classifier, in order to assign (classify) texts to their authors. We should stress here that the VS function is a similarity function, but not a kernel. Fortunately, VS on the specific data set is an (ϵ, γ) -good similarity function as defined in (cf. [6, Theorem 3]) and hence, this allows its use as a kernel function, in line with what we have already said about string kernels in Section 11.5.2.

It is interesting to note that, given the VSM representation of a text, one can also employ any vector-based kernel on strings. However, there have been several works that use *string kernels* [70] to avoid any loss of information in the transformation between the original string and its VSM equivalent. In this example, we demonstrate how one can combine even a complex representation with the SVM via an appropriate kernel function.

The dataset we used to examine the effectiveness of this combination is a subset of the [Reuter_50_50 Data Set](#) from the UCI repository [5]. This dataset contains 2500 training and 2500 test texts from 50 different authors (50 training and 50 test texts per author).

- First, we used the `ngg2svm` command-line tool¹⁵ to represent the strings as n -gram graphs and generate a precomputed kernel matrix file. The tool extracts are—by default—3-gram graphs ($n = 3, D = 3$) from training texts. It then uses VS to estimate a kernel and calculates the kernel values over all pairs of training instances into a kernel matrix. The matrix is the output of this phase.
- Given the kernel matrix, we can use the LibSVM software [22] to design a classifier, without caring about the original texts, but only relying on precomputer kernel values.

Then, a 10-fold crossvalidation over the data was performed, using the “`svmtrain`” program in LibSVM. The achieved accuracy of the cross validation in our example was 94%.

¹⁵ You can download the tool from <https://github.com/ggianna/ngg2svm>.

PROBLEMS

- 11.1** Derive the formula for the number of groupings $\mathcal{O}(N, l)$ in Cover's theorem.

Hint. Show first the following recursion:

$$\mathcal{O}(N + 1, l) = \mathcal{O}(N, l) + \mathcal{O}(N, l - 1).$$

To this end, start with N points and add an extra one. Show that the extra number of linear dichotomies is solely due to those, for the N data point case, which could be drawn via the new point.

- 11.2** Show that if $N = 2(l + 1)$, the number of linear dichotomies in Cover's theorem is equal to 2^{2l+1} .

Hint. Use the identity

$$\sum_{i=1}^j \binom{j}{i} = 2^j$$

and recall that

$$\binom{2n+1}{n-i+1} = \binom{2n+1}{n+i}.$$

- 11.3** Show that the reproducing kernel is a positive definite one.

- 11.4** Show that if $\kappa(\cdot, \cdot)$ is the reproducing kernel in an RKHS, \mathbb{H} , then

$$\mathbb{H} = \overline{\text{span}\{\kappa(\cdot, \mathbf{x}), \mathbf{x} \in \mathcal{X}\}}.$$

- 11.5** Show the Cauchy-Schwarz inequality for kernels, that is,

$$\|\kappa(\mathbf{x}, \mathbf{y})\|^2 \leq \kappa(\mathbf{x}, \mathbf{x})\kappa(\mathbf{y}, \mathbf{y}).$$

- 11.6** Show that if

$$\kappa_i(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}, i = 1, 2,$$

are kernels then

- $\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(\mathbf{x}, \mathbf{y}) + \kappa_2(\mathbf{x}, \mathbf{y})$ is also a kernel.
- $a\kappa(\mathbf{x}, \mathbf{y})$, $a > 0$ is also a kernel.
- $\kappa(\mathbf{x}, \mathbf{y}) = \kappa_1(\mathbf{x}, \mathbf{y})\kappa_2(\mathbf{x}, \mathbf{y})$ is also a kernel.

- 11.7** Derive Eq. (11.25).

- 11.8** Show that the solution for the parameters, $\hat{\theta}$, for the kernel ridge regression, if a bias term, b , is present, is given by

$$\begin{bmatrix} \mathcal{K} + CI & \mathbf{1} \\ \mathbf{1}^T \mathcal{K} & N \end{bmatrix} \begin{bmatrix} \theta \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ \mathbf{y}^T \mathbf{1} \end{bmatrix},$$

where $\mathbf{1}$ is the vector with all its elements being equal to one. Invertibility of the kernel matrix has been assumed.

- 11.9** Derive Eq. (11.56).

- 11.10** Derive the dual cost function associated with the linear ϵ -insensitive loss function.

- 11.11** Derive the dual cost function for the separable class SVM formulation.
11.12 Prove that in the NORMA, the hypothesis at time n has an equivalent expansion

$$f_n = \sum_{i=1}^n \theta_i^{(i)} (1 - \mu\lambda)^{n-i} \kappa(\cdot, \mathbf{x}_i),$$

where $\theta_i^{(i)} = -\mu \mathcal{L}'(y_i, f_{i-1}(\mathbf{x}_i))$.

- 11.13** Derive a bound on the approximation error associated with the truncation of the expansion

$$f_n = \sum_{i=1}^n \theta_i^{(i)} (1 - \mu\lambda)^{n-i} \kappa(\cdot, \mathbf{x}_i).$$

adopted in the NORMA algorithm. Assume that

$$\left| \frac{\partial}{\partial z} \mathcal{L}(y, z) \right| \leq c, \quad \|\kappa(\cdot, \mathbf{x})\| \leq B, \quad \forall \mathbf{x} \in \mathcal{X}, z \in \mathbb{R}.$$

- 11.14** In the hinge loss function, assume ρ is a free parameter. Derive a version of NORMA that updates for this parameter as well.
11.15 Derive the subgradient for the Huber loss function and the respective weight updates for the NORMA.
11.16 Show that the formula for the recursive computation of the norm $\|f_n\|$, in the context of the kernel APSM algorithm, is given by

$$\begin{aligned} \|f_n\|^2 &= \|f_{n-1}\|^2 + \mu_n^2 \sum_{k=n-q+1}^n \sum_{l=n-q+1}^n \beta_k \beta_l \kappa(\mathbf{x}_k, \mathbf{x}_l) \\ &\quad - 2\mu_n \sum_{i=1}^n \sum_{k=n-q+1}^n \beta_i \beta_k \kappa(\mathbf{x}_i, \mathbf{x}_k). \end{aligned}$$

- 11.17** Derive the formula for the bound M_n used in [Algorithm 11.3](#).
11.18 Derive the update recursion for the KAPSM if a bias term is employed.

MATLAB Exercises

- 11.19** Consider the regression problem described in [Example 11.2](#). Read an audio file using MATLAB's wavread function and take 100 data samples (use *Blade Runner*, if possible, and take 100 samples starting from the 100,000th sample). Then add white gaussian noise at a 15 dB level and randomly “hit” 10 of the data samples with outliers (set the outlier values to 80% of the maximum value of the data samples).
- (a) Find the reconstructed data samples using the unbiased kernel ridge regression method, that is, Eq. (11.27). Employ the Gaussian kernel with $\sigma = 0.004$ and set $C = 0.0001$. Plot the fitted curve of the reconstructed samples together with the data used for training.
 - (b) Find the reconstructed data samples using the biased kernel ridge regression method (employ the same parameters as for the unbiased case). Plot the fitted curve of the reconstructed samples together with the data used for training.
 - (c) Repeat the steps [11.19a](#), [11.19b](#) using $C = 10^{-6}, 10^{-5}, 0.0005, 0.001, 0.01$, and 0.05 .

- (d) Repeat the steps 11.19a, 11.19b using $\sigma = 0.001, 0.003, 0.008, 0.01$, and 0.05 .
 (e) Comment on the results.

- 11.20** Consider the regression problem described in Example 11.2. Read the same audio file as in Problem 11.19. Then add white gaussian noise at a 15 dB level and randomly “hit” 10 of the data samples with outliers (set the outlier values to 80% of the maximum value of the data samples).
- (a) Find the reconstructed data samples obtained by the support vector regression (you can use libsvm¹⁶ for training). Employ the Gaussian kernel with $\sigma = 0.004$ and set $\epsilon = 0.003$ and $C = 1$. Plot the fitted curve of the reconstructed samples together with the data used for training.
 (b) Repeat step 11.20a using $C = 0.05, 0.1, 0.5, 5, 10$, and 100 .
 (c) Repeat step 11.20a using $\epsilon = 0.0005, 0.001, 0.01, 0.05$, and 0.1 .
 (d) Repeat step 11.20a using $\sigma = 0.001, 0.002, 0.01, 0.05$, and 0.1 .
 (e) Comment on the results.

- 11.21** Consider the two-class two-dimensional classification task described in Example 11.4. The data set comprises $N = 150$ points uniformly distributed in the region $[-5, 5] \times [-5, 5]$. For each point, $\mathbf{x}_n = [x_{n,1}, x_{n,2}]^T$, $n = 1, 2, \dots, N$, compute

$$y_n = 0.5x_{n,1}^3 + 0.5x_{n,1}^2 + 0.5x_{n,1} + \eta$$

where η denotes zero-mean Gaussian noise of variance $\sigma_\eta^2 = 4$. Assign each point to either of the two classes, depending on which side of the graph of the function

$$f(x) = 0.5x^3 + 0.5x^2 + 0.5x + 1,$$

in the two-dimensional space, y_n lies.

- (a) Plot the points (x_n, y_n) using different colors for each class.
 (b) Obtain the SVM classifier using libsvm or any other related MATLAB package. Use the Gaussian kernel with $\sigma = 20$ and set $C = 1$. Plot the classifier and the margin (for the latter you can employ MATLAB’s contour function). Moreover, find the support vectors (i.e., the points with nonzero Lagrange multipliers that contribute to the expansion of the classifier) and plot them as circled points.
 (c) Repeat step 11.21b using $C = 0.5, 0.1$, and 0.05 .
 (d) Repeat step 11.21b using $C = 5, 10, 50$, and 100 .
 (e) Comment on the results.

- 11.22** Consider the nonlinear equalization task described in Example 11.5.

- (a) Create MATLAB functions that realize the QKLMS, QKAPSM, and NORMA algorithms using the Gaussian kernel. These functions should take as inputs a training sequence (y_n, \mathbf{x}_n) , the kernel parameter σ , the sparsification parameter s , and the learning rate μ . NORMA needs an extra parameter, the regularization parameter λ and QKAPSM the parameter, ϵ , and q , that is, the number of samples that are concurrently processed. For the QKAPSM and QKLMS, s denotes the quantization size δ while for the NORMA s is the window size, that is, n_0 . The functions should provide as outputs the dictionary

¹⁶ <http://www.csie.ntu.edu.tw/cjlin/libsvm/>.

centers and coefficients (that are involved in the expansion of the respective solutions, for example, (11.117)) and the error e_n at each step.

- (b) Create an input signal comprising 5000 values generated by the Gaussian distribution with $\sigma = 0.8$ and zero-mean value. Then construct the sequence t_n using

$$t_n = -0.9s_n + 0.6s_{n-1} - 0.7s_{n-2} + 0.2s_{n-3} + 0.1s_{n-4},$$

and the sequence

$$x_n = 0.15t_n^2 + 0.03t_n^3 + \eta_n,$$

where η_n denotes a zero-mean white Gaussian noise with $\sigma = 0.14$. Then create the sequence of the data that will be used to train the equalizers, that is, (y_n, \mathbf{x}_n) , where $\mathbf{x}_n = [x_n, x_{n-1}, \dots, x_{n-4}]^T$ (i.e., $l = 5$) and $y_n = s_{n-2}$ (i.e., $D = 2$).

- (c) Using a For-loop, create 100 training sequences as described in step 11.22a. Feed the functions you developed with the aforementioned training sequences. Plot (in a single figure) the MSEs returned by each algorithm (averaged over the 100 training sequences). Use $\sigma = 5$ as the parameter of the Gaussian kernel, $\mu = 1/2$ for the QKLMS and QKAPSM, and $\mu = 1/4$ for NORMA, $\delta = 7$ for the quantization for both QKLMS and QKAPSM, and $n_0 = 80$ for the NORMA. Also, set $\epsilon = 10^{-5}$ and $q = 5$ for QKAPSM and $\lambda = 0.01$ for NORMA. Count the size of the dictionary for each algorithm. Comment on the results.
 (d) Repeat step 11.22c for $\mu = 1/16, 1/8, 1/4, 1, 2, 8, 16$. Keep in mind that if we want the experiment to be fair, NORMA should be carried out with the value of μ half the size of the one used for KLMS and KAPSM. Comment on the results.
 (e) Repeat step 11.22c for $\sigma = 1, 2, 10, 15, 20$. Comment on the results.
 (f) Repeat step 11.22c for $\delta = 1, 2, 10, 15$, and $n_0 = 800, 350, 40, 20$. Comment on the results.
 (g) Repeat step 11.22a to create 5000 data samples. Then change the linear part of the channel to

$$t_n = 0.8s_n - 0.7s_{n-1} + 0.6s_{n-2} - 0.2s_{n-3} - 0.2s_{n-4}$$

and the nonlinear part to

$$x_n = 0.12t_n^2 + 0.02t_n^3 + \eta_n,$$

and create another set of 5000 samples. Repeat step 11.22c, feeding the MATLAB functions you created with training sequences of the aforementioned form. Comment on the results.

- 11.23** Consider the authorship identification problem described in Section 11.15.

- (i) Using the training texts of the authors whose name start with “T” in the dataset, perform 10-fold cross validation using the n -gram graph representation of the texts and the value similarity as a kernel function.
- (ii) Using the same subset of the dataset, create the vector space model representation of the texts and apply classification using a Gaussian kernel. Compare the results with the results of (i).

REFERENCES

- [1] A. Argyriou, C.A. Micchelli, M. Pontil, When is there a representer theorem? Vector versus matrix regularizers, *J. Machine Learn. Res.* 10 (2009) 2507-2529.
- [2] N. Aronszajn, Theory of reproducing kernels, *Trans. Amer. Math. Soc.* 68(3) (1950) 337-404.
- [3] P. Auer, G. Gentile, Adaptive and self-confident on-line learning algorithms, *J. Comput. Syst. Sci.* 64(1) (2002) 48-75.
- [4] F.R. Bach, Consistency of the group LASSO and multiple kernel learning, *J. Machine Learn. Res.* 9 (2008) 1179-1225.
- [5] K. Bache, M. Lichman, UCI Machine Learning Repository, <http://archive.ics.uci.edu/ml>, University of California, Irvine, School of Information and Computer Sciences, 2013.
- [6] M.F. Balcan, A. Blum, N. Srebro, A theory of learning with similarity functions, *Machine Learn.* 72(1-2) (2008) 89-112.
- [7] A.R. Barron, Universal approximation bounds for superposition of a sigmoid function, *Trans. Informat. Theory* 39(3) (1993) 930-945.
- [8] E.J. Bayro-Corrochano, N. Arana-Daniel, Clifford support vector machines for classification, regression and recurrence, *IEEE Trans. Neural Networks* 21(11) (2010) 1731-1746.
- [9] J.A. Bazerque, G.B. Giannakis, Nonparametric basis pursuit via sparse kernel-based learning, *IEEE Signal Process. Mag.* 30(4) (2013) 112-125.
- [10] J.A. Bazerque, G. Mateos, G.B. Giannakis, Group-LASSO on splines for spectrum cartography, *Trans. Signal Process.* 59(10) (2011) 4648-4663.
- [11] S. Beneteto, E. Bigleiri, *Principles of Digital Transmission*, Springer, 1999.
- [12] Y. Bengio, O. Delalleau, N. Le Roux, The curse of highly variable functions for local kernel machines, in: Y. Weiss, B. Scholkopf, J. Platt (Eds.), *Advances in Neural Information Processing Systems*, NIPS, MIT Press, 2006, pp. 107-114.
- [13] P. Bouboulis, K. Slavakis, S. Theodoridis, Adaptive kernel-based image denoising employing semi-parametric regularization, *IEEE Trans. Image Process.* 19(6) (2010) 1465-1479.
- [14] P. Bouboulis, S. Theodoridis, C. Mavroforakis, L. Dalla, Complex support vector machines for regression and quaternary classification, *IEEE Trans. Neural Networks Learning Syst.*, to appear, 2014.
- [15] P. Bouboulis, S. Theodoridis, Kernel methods for image denoising, in: J.A.K. Suykens, M. Signoretto, A. Argyriou (Eds.), *Regularization, Optimization, Kernels, and Support Vector Machines*, Chapman and Hall/CRC, Boca Raton, FL, 2014.
- [16] O. Bousquet, A. Elisseeff, Stability and generalization, *J. Machine Learn. Res.* 2 (2002) 499-526.
- [17] C.J.C. Burges, Geometry and invariance in kernel based methods, in: B. Scholkopf, C.J. Burges, A.J. Smola (Eds.), *Advances in Kernel Methods*, MIT Press, 1999, pp. 90-116.
- [18] W.B. Cavnar, J.M. Trenkle et al., N-gram-based text categorization, in: *Symposium on Document Analysis and Information Retrieval*, University of Nevada, Las Vegas, 1994, pp. 161-176.
- [19] L.J. Cao, S.S. Keerthi, C.J. Ong, J.Q. Zhang, U. Periyathamby, X.J. Fu, H.P. Lee, Parallel sequential minimal optimization for the training of support vector machines, *IEEE Trans. Neural Networks* 17(4) (2006) 1039-1049.
- [20] C.C. Chang, C.W. Hsu, C.J. Lin, The analysis of decomposition methods for SVM, *IEEE Trans. Neural Networks* 11(4) (2000) 1003-1008.
- [21] C.C. Chang, C.J. Lin, Training ν -support vector classifiers: Theory and algorithms, *Neural Comput.* 13(9) (2001) 2119-2147.
- [22] C.-C. Chang, C.J. Lin, LIBSVM: A library for support vector machines, *ACM Trans. Intell. Syst. Tech.* 2(3) (2011) 27:1-27:27.
- [23] R. Caruana, A. Niculescu-Mizil, An empirical comparison of supervised learning algorithms, in: *International Conference on Machine Learning*, 2006.

- [24] G. Cauwenberghs, T. Poggio, Incremental and decremental support vector machine learning, in: Advances in Neural Information Processing Systems, NIPS, MIT Press, 2001, pp. 409-415.
- [25] O. Chapelle, Training a support vector machine in the primal, *Neural Comput.* 19(5) (2007) 1155-1178.
- [26] B. Chen, S. Zhao, P. Zhu, J.C. Principe, Quantized kernel least mean square algorithm, *IEEE Trans. Neural Networks Learn. Syst.* 23(1) (2012) 22-32.
- [27] N. Cristianini, J. Shawe-Taylor, An introduction to Support Vector Machines, Cambridge University Press, 2000.
- [28] C. Cortes, P. Haffner, M. Mohri, Rational kernels: Theory and algorithms, *J. Machine Learn. Res.* 5 (2004) 1035-1062.
- [29] C. Cortes, M. Mohri, A. Rostamizadeh, L_2 Regularization for learning kernels, in: Proc. of the 25th Conference on Uncertainty in Artificial Intelligence, 2009, pp. 187-196.
- [30] T.M. Cover, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, *IEEE Trans. Electron. Comput.* 14 (1965) 326.
- [31] P. Crama, J. Schoukens, Hammerstein-Wiener system estimator initialization, *Automatica* 40(9) (2004) 1543-1550.
- [32] D.J. Crisp, C.J.C. Burges, A geometric interpretation of ν -SVM classifiers, in: Proceedings of Neural Information Processing, NIPS, Vol. 12, MIT Press, 1999.
- [33] T.G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, *J. Artif. Intell. Res.* 2 (1995) 263-286.
- [34] G. Ding, Q. Wu, Y.-D. Yao, J. Wang, Y. Chen, Kernel-based learning for statistical signal processing in cognitive radio networks, *IEEE Signal process. Mag.* 30(4) (2013) 126-136.
- [35] C. Domeniconi, D. Gunopulos, Incremental support vector machine construction, in: IEEE International Conference on Data Mining, San Jose, USA, 2001, pp. 589-592.
- [36] Y. Engel, S. Mannor, R. Meir, The kernel recursive least-squares algorithm, *IEEE Transactions on Signal Processing* 52(8) (2004) 2275-2285.
- [37] T. Evgeniou, M. Pontil, T. Poggio, Regularization networks and support vector machines, *Adv. Comput. Math.* 13 (2000) 1-50.
- [38] T. Evgeniou, C.A. Michelli, M. Pontil, Learning multiple tasks with kernel methods, *J. Machine Learn. Res.* 6 (2005) 615-637.
- [39] B. Fei, J. Liu, Binary tree of SVM: a new fast multiclass training and classification algorithm, *IEEE Trans. Neural Networks* 17(5) (2006) 696-704.
- [40] P.A. Forero, A. Cano, G.B. Giannakis, Consensus-based distributed support vector machines, *J. Machine Learn. Res.* 11 (2010) 1663-1707.
- [41] V. Franc, S. Sonnenburg, Optimized cutting plane algorithm for large-scale risk minimization, *J. Machine Learn. Res.* 10 (2009) 2157-2192.
- [42] M. Fréchet, Sur les fonctionnelles continues, *Ann. Sci. de l'Ecole Super.* 27 (1910) 193-216.
- [43] C. Gentile, A new approximate maximal margin classification algorithm, *J. Machine Learn. Res.* 2 (2001) 213-242.
- [44] G. Giannakopoulos, V. Karkaletsis, G. Vouros, P. Stamatopoulos, Summarization system evaluation revisited: N-gram graphs, *ACM Trans. Speech Lang. Process.* 5(3) (2008) 1-9.
- [45] F. Girosi, M. Jones, T. Poggio, Regularization theory and neural networks architectures, *Neural Comput.* 7(2) (1995) 219-269.
- [46] F. Girosi, An equivalence between sparse approximation and support vector machines, *Neural Comput.* 10(6) (1998) 1455-1480.
- [47] M. Gonen, E. Alpaydin, Multiple kernel learning algorithms, *J. Machine Learn. Res.* 12 (2011) 2211-2268.
- [48] Z. Harchaoui, F. Bach, O. Cappe, E. Moulines, Kernel-based methods for hypothesis testing, *IEEE Signal Process. Mag.* 30(4) (2013) 87-97.

- [49] D. Hardoon, J. Shawe-Taylor, Decomposing the tensor kernel support vector machine for neuroscience data with structured labels, *Neural Networks* 24(8) (2010) 861-874.
- [50] J.R. Higgins, *Sampling Theory in Fourier and Signal Analysis Foundations*, Oxford Science Publications, 1996.
- [51] T. Hofmann, B. Scholkopf, A. Smola, Kernel methods in machine learning, *Ann. Stat.* 36(3) (2008) 1171-1220.
- [52] D. Hush, P. Kelly, C. Scovel, I. Steinwart, QP algorithms with guaranteed accuracy and run time for support vector machines, *J. Machine Learn. Res.* 7 (2006) 733-769.
- [53] P. Huber, Robust estimation of location parameters, *Ann. Math. Stat.* 35(1) (1964) 73-101.
- [54] T. Joachims, Making large scale support vector machine learning practical, in: B. Scholkopf, C.J. Burges, A.J. Smola (Eds.), *Advances in Kernel Methods: Support Vector Learning*, MIT Press, 1999.
- [55] T. Joachims, T. Finley, C.-N. Yu, Cutting-plane training of structural SVMs, *Machine Learn.* 77(1) (2009) 27-59.
- [56] N. Kalouptsidis, *Signal Processing Systems: Theory and Design*, John Wiley, 1997.
- [57] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy, A fast iterative nearest point algorithm for support vector machine classifier design, *IEEE Trans. Neural Networks* 11(1) (2000) 124-136.
- [58] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, K.R.K. Murthy, Improvements to Platt's SMO algorithm for SVM classifier design, *Neural Comput.* 13 (2001) 637-649.
- [59] A.Y. Kibangou, G. Favier, Wiener-Hammerstein systems modeling using diagonal Volterra kernels coefficients, *Signal Process. Lett.* 13(6) (2006) 381-384.
- [60] G. Kimeldorf, G. Wahba, Some results on Tchebycheffian spline functions, *J. Math. Anal. Appl.* 33 (1971) 82-95.
- [61] J. Kivinen, A.J. Smola, R.C. Williamson, Online learning with kernels, *IEEE Trans. Signal Process.* 52(8) (2004) 2165-2176.
- [62] J. Kivinen, M.K. Warmuth, B. Hassibi, The p -norm generalization of the LMS algorithm for adaptive filtering, *IEEE Trans. Signal Process.* 54(5) (2006) 1782-1793.
- [63] M. Kloft, U. Brefeld, S. Sonnenburg, P. Laskov, K.R. Müller, A. Zien, Efficient and accurate l_p -norm multiple kernel learning, *Adv. Neural Informat. Process. Syst.* 22 (2009) 997-1005.
- [64] S.Y. Kung, *Kernel Methods and Machine Learning*, Cambridge University Press, 2014.
- [65] G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, M. Jordan, Learning the kernel matrix with semidefinite programming, in: C. Sammut, A.G. Hoffman (Eds.), *Proceedings of the 19th International Conference on Machine Learning, ICML*, Morgan Kaufmann, 2002, pp. 323-330.
- [66] J.L. Lázaro, J. Dorronsoro, Simple proof of convergence of the SMO algorithm for different SVM variants, *IEEE Trans. Neural Networks Learning Syst.* 23(7) (2012) 1142-1147.
- [67] Y. Lin, H.H. Zhang, Component selection and smoothing in multivariate nonparametric regression, *Ann. Stat.* 34(5) (2006) 2272-2297.
- [68] W. Liu, P. Pokharel, J. Principe, The kernel least mean squares algorithm, *IEEE Trans. Signal Process.* 56(2) (2008) 543-554.
- [69] W. Liu, J.C. Principe, S. Haykin, *Kernel Adaptive Filtering: A Comprehensive Introduction*, John Wiley, 2010.
- [70] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, C. Watkins, Text classification using string kernels, *J. Machine Learn. Res.* 2 (2002) 419-444.
- [71] V.J. Mathews, G.L. Sicuranza, *Polynomial Signal Processing*, John Wiley, New York, 2000.
- [72] G. Mateos, G.B. Giannakis, Robust nonparametric regression via sparsity control with application to load curve data cleansing, *IEEE Trans. Signal Process.* 60(4) (2012) 1571-1584.
- [73] P.Z. Marmarelis, V.Z. Marmarelis, *Analysis of Physiological Systems – The White Noise Approach*, Plenum, New York, 1978.

- [74] M. Mavroforakis, S. Theodoridis, Support vector machine classification through geometry, in: Proceedings XII European Signal Processing Conference, EUSIPCO, Anatlya, Turkey, 2005.
- [75] M. Mavroforakis, S. Theodoridis, A geometric approach to support vector machine classification, IEEE Trans. Neural Networks 17(3) (2006) 671-682.
- [76] M. Mavroforakis, M. Sdralis, S. Theodoridis, A geometric nearest point algorithm for the efficient solution of the SVM classification task, IEEE Trans. Neural Networks 18(5) (2007) 1545-1549.
- [77] J. Mercer, Functions of positive and negative type and their connection with the theory of integral equations, Phil. Trans. R. Soc. Lond. 209 (1909) 415-446.
- [78] D. Meyer, F. Leisch, K. Hornik, The support vector machine under test, Neurocomputing 55 (2003) 169-186.
- [79] C.A. Micceli, Interpolation of scattered data: distance matrices and conditionally positive definite functions, Construct. Approximat. 2 (1986) 11-22.
- [80] C.A. Miccheli, M. Pontil, Learning the kernel function via regularization, J. Machine Learn. Res. 6 (2005) 1099-1125.
- [81] K. Mitra, A. Veeraraghavan, R. Chellappa, “Analysis of sparse regularization based robust regression approaches,” IEEE Transactions on Signal Processing, Vol. 61(5), pp. 1249–1257, 2013.
- [82] G. Montavon, M.L. Braun, T. Krueger, K.R. Müller, Analysing local structure in kernel-based learning, IEEE Signal Process. Mag. 30(4) (2013) 62-74.
- [83] E.H. Moore, On properly positive Hermitian matrices, Bull. Amer. Math. Soc. 23 (1916) 59.
- [84] K. Muller, S. Mika, G. Ratsch, K. Tsuda, B. Scholkopf, An introduction to kernel-based learning algorithms, IEEE Trans. Neural Networks 12(2) (2001) 181-201.
- [85] A. Navia-Vázquez, F. Perez-Cruz, A. Artes-Rodriguez, A. Figueiras-Vidal, Weighted least squares training of support vector classifiers leading to compact and adaptive schemes, IEEE Trans. Neural Networks 15(5) (2001) 1047-1059.
- [86] G. Papageorgiou, P. Bouboulis, S. Theodoridis, Robust linear regression analysis - a greedy approach, IEEE Trans. Signal Process. 2015.
- [87] G. Papageorgiou, P. Bouboulis, S. Theodoridis, K. Themelis “Robust Linear Regression Analysis - A Greedy Approach,” arXiv:1409.4279 [cs.IT] 2014.
- [88] W.D. Parreira, J.C.M. Bermudez, C. Richard, J.-Y. Tourneret, Stochastic behavior analysis of the Gaussian kernel least-mean-square algorithm, IEEE Trans. Signal Process. 60(5) (2012) 2208-2222.
- [89] V.I. Paulsen, An introduction to theory of reproducing kernel Hilbert spaces, Notes, 2009.
- [90] J. Platt, Sequential minimal optimization: A fast algorithm for training support vector machines, Technical Report, Microsoft Research, MSR-TR-98-14, April 21, 1998.
- [91] J.C. Platt, Using analytic QP and sparseness to speed training of support vector machines, in: Proceedings Neural Information Processing Systems, NIPS, 1999.
- [92] T. Poggio, S. Smale, The mathematics of learning: dealing with data, Amer. Math. Soc. Notice 50(5) (2003) 537-544.
- [93] M.J.D. Powell, Radial basis functions for multivariate interpolation: a review, in: J.C. Mason, M.G. Cox (Eds.), Algorithms for Approximation, Clarendon Press, Oxford, 1987, pp. 143-167.
- [94] A. Rakotomamonjy, F.R. Bach, S. Canu, Y. Grandvalet, SimpleMKL, J. Machine Learn. Res. 9 (2008) 2491-2521.
- [95] P. Ravikumar, J. Lafferty, H. Liu, L. Wasserman, Sparse additive models, J. R. Stat. Soc. B 71(5) (2009) 1009-1030.
- [96] C. Richard, J. Bermudez, P. Honeine, Online prediction of time series data with kernels, IEEE Trans. Signal Process. 57(3) (2009) 1058-1067.
- [97] W. Rudin, Principles of Mathematical Analysis, third ed., McGraw-Hill, 1976.
- [98] G. Salton, A. Wong, C.-S. Yang, A vector space model for automatic indexing, Commun. ACM 18(11) (1975) 623-620.

- [99] C. Saunders, A. Gammerman, V. Vovk, Ridge regression learning algorithm in dual variables, in: J. Shavlik (Ed.), Proceedings 15th International Conference on Machine Learning, ICML'98, Morgan Kaufman, 1998.
- [100] P. Saurabh, C. Boutsidis, M. Magdon-Ismail, P. Drineas "Random projections for support vector machines," Proceedings of the 16th International Conference on Artificial Intelligence and Statistics (AISTATS) Scottsdale, AZ, USA., 2013.
- [101] A. Shilton, M. Palaniswami, D. Ralph, A.C. Tsoi, Incremental training of support vector machines, IEEE Trans. Neural Networks 16(1) (2005) 114-131.
- [102] M. Schetzen, Nonlinear system modeling based on the Wiener theory, Proc. IEEE 69(12) (1981) 1557-1573.
- [103] M. Schetzen, The Volterra and Wiener Theories of Nonlinear Systems, John Wiley, New York, 1980.
- [104] B. Schölkopf, A.J. Smola, Learning with Kernels, MIT Press, Cambridge, 2001.
- [105] B. Schölkopf, A.J. Smola, R.C. Williamson, P.L. Bartlett, New support vector algorithms, Neural Comput. 12 (2000) 1207-1245.
- [106] S. Shalev-Shwartz, Y. Singer, N. Srebro, A. Cotter, PEGASOS: primal estimated sub-gradient solver for SVM, Math. Program. 127(1) (2011) 3-30.
- [107] J. Shawe-Taylor, N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge University Press, 2004.
- [108] M. Signoretto, L. De Lathauwer, J. Suykens, A kernel-based framework to tensorial data analysis, Neural Networks 24(8) (2011) 861-874.
- [109] K. Slavakis, P. Bouboulis, S. Theodoridis, Online learning in reproducing kernel Hilbert spaces, Academic Press Library in Signal Processing, Signal Process. Theory Machine Learn. 1 (2013) 883-987.
- [110] K. Slavakis, S. Theodoridis, I. Yamada, Online kernel-based classification using adaptive projections algorithms, IEEE Trans. Signal Process. 56(7) (2008) 2781-2796.
- [111] K. Slavakis, S. Theodoridis, Sliding window generalized kernel affine projection algorithm using projection mappings, EURASIP J. Adv. Signal Process. Article ID 735351, doi:10.1155/2008/735351, 2008.
- [112] K. Slavakis, S. Theodoridis, I. Yamada, Adaptive constrained learning in reproducing kernel Hilbert spaces, IEEE Trans. Signal Process. 5(12) (2009) 4744-4764.
- [113] K. Slavakis, A. Bouboulis, S. Theodoridis, Adaptive multiregression in reproducing kernel Hilbert spaces, IEEE Trans. Neural Networks Learning Syst. 23(2) (2012) 260-276.
- [114] K. Slavakis, A. Bouboulis, S. Theodoridis, Online learning in reproducing kernel Hilbert spaces, in: S. Theodoridis, R. Chellappa (Eds.), E-reference for Signal Processing, Academic Press, 2013.
- [115] E. Stamatatos, A survey of modern authorship attribution methods, J. Amer. Soc. Informat. Sci. Tech. 60(3) (2009) 538-556.
- [116] M.O. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, J. Weston, Support vector regression with ANOVA decomposition kernels, in: B. Scholkopf, C.J. Burges, A.J. Smola (Eds.), Advances in Kernel Methods: Support Vector Learning, MIT Press, 1999.
- [117] S. Sonnenburg, G. Rätsch, C. Schaffner, B. Scholkopf, Large scale multiple kernel learning, J. Machine Learn. Res. 7 (2006) 1531-1565.
- [118] L. Song, K. Fukumizu, A. Gretton, Kernel embeddings of conditional distributions, IEEE Signal Process. Mag. 30(4) (2013) 98-111.
- [119] J.A.K. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural Process. Lett. 9 (1999) 293-300.
- [120] J.A.K. Suykens, T. van Gestel, J. de Brabanter, B. de Moor, J. Vandewalle, Least Squares Support Vector Machines, World Scientific, Singapore, 2002.
- [121] J.A.K. Suykens, M. Signoretto, A. Argyriou (Eds.), *Regularization, Optimization, Kernels, and Support Vector Machines*, Chapman and Hall/CRC, Boca Raton, FL, 2014.

- [122] R. Talmon, I. Cohen, S. Gannot, R. Coifman, Diffusion maps for signal processing, *IEEE Signal Process. Mag.* 30(4) (2013) 75-86.
- [123] Q. Tao, G.-W. Wu, J. Wang, A generalized S-K algorithm for learning ν -SVM classifiers, *Pattern Recogn. Lett.* 25(10) (2004) 1165-1171.
- [124] S. Theodoridis, M. Mavroforakis, Reduced convex hulls: a geometric approach to support vector machines, *Signal Process. Mag.* 24(3) (2007) 119-122.
- [125] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, fourth ed., Academic Press, 2009.
- [126] F.A. Tobar, D.P. Mandic "Quaternion Reproducing Kernel Hilbert Spaces: Existence and Uniqueness Conditions," *IEEE Transactions on Information Theory*, Vol. 60(9), pp. 5736-5749, 2014.
- [127] S. Van Vaerenbergh, M. Lazaro-Gredilla, I. Santamaria, Kernel recursive least-squares tracker for time-varying regression, *IEEE Trans. Neural Networks Learn. Syst.* 23(8) (2012) 1313-1326.
- [128] V.N. Vapnik, *The Nature of Statistical Learning*, Springer, 2000.
- [129] V.N. Vapnik, *Statistical Learning Theory*, Wiley, 1998.
- [130] M.K. Warmuth, A.K. Jagota, Continuous and discrete-time nonlinear gradient descent: Relative loss bounds and convergence, in: E. Boros, R. Greiner (Eds.), *Proceedings Fifth International Symposium on Artificial Intelligence and Mathematics*, 1998.
- [131] N. Wiener, *Nonlinear Problems in Random Theory*, Technology Press, MIT and Wiley, 1958.
- [132] G.-X. Yuan, K.W. Chang, C.-J. Hsieh, C.J. Lin, A comparison of optimization methods and software for large-scale ℓ_1 -regularized linear classification, *J. Machine Learn. Res.* 11 (2010) 3183-3234.
- [133] Q. Zhao, G. Zhou, T. Adali, L. Zhang, A. Cichocki, Kernelization of tensor-based models for multiway data analysis, *IEEE Signal Process. Mag.* 30(4) (2013) 137-148.

This page intentionally left blank

BAYESIAN LEARNING: INFERENCE AND THE EM ALGORITHM

12

CHAPTER OUTLINE

12.1 Introduction	594
12.2 Regression: A Bayesian Perspective	594
12.2.1 The Maximum Likelihood Estimator	595
12.2.2 The MAP Estimator	596
12.2.3 The Bayesian Approach	597
12.3 The Evidence Function and Occam's Razor Rule	602
<i>Laplacian Approximation and the Evidence Function</i>	605
12.4 Exponential Family of Probability Distributions.....	609
12.4.1 The Exponential Family and the Maximum Entropy Method	614
12.5 Latent Variables and the EM Algorithm	615
12.5.1 The Expectation-Maximization Algorithm	615
12.5.2 The EM Algorithm: A Lower Bound Maximization View	617
12.6 Linear Regression and the EM Algorithm	619
12.7 Gaussian Mixture Models.....	622
12.7.1 Gaussian Mixture Modeling and Clustering	626
12.8 Combining Learning Models: A Probabilistic Point of View	630
12.8.1 Mixing Linear Regression Models	631
<i>Mixture of Experts</i>	633
<i>Hierarchical Mixture of Experts</i>	634
12.8.2 Mixing Logistic Regression Models	634
Problems.....	637
<i>MATLAB Exercises</i>	638
12.9 Appendix to Chapter 12	640
12.9.1 PDFs with Exponent of Quadratic Form	640
12.9.2 The Conditional from the Joint Gaussian Pdf	641
12.9.3 The Marginal from the Joint Gaussian Pdf	642
12.9.4 The Posterior from Gaussian Prior and Conditional Pdfs	643
References.....	646

12.1 INTRODUCTION

The Bayesian approach to parameter inference was introduced in [Chapter 3](#). In contrast to other methods for parameter estimation we have covered, the Bayesian method adopts a radically different viewpoint. The unknown set of parameters are treated as random variables instead of as a set of fixed (yet unknown) values. This was a revolutionary idea, at the time it was introduced by Bayes and later on by Laplace, as pointed out in [Chapter 3](#). Even now, after more than two centuries, it may seem strange to assume that a physical phenomenon/mechanism is controlled by a set of random parameters. However, there is a subtle point here. Treating the underlying set of parameters as random variables, θ , we do not really imply a random nature for them. The associated randomness, in terms of the prior distribution $p(\theta)$, encapsulates our *uncertainty* about their values, prior to receiving any measurements/observations. Stated differently, the prior distribution represents our *belief* about the different possible values, although only one of them is actually true. From this perspective, probabilities are viewed in a more open-minded way, that is, as measures of uncertainty, as discussed in the beginning of [Chapter 2](#).

Recall that parameter learning from data is an inverse problem. Basically, all we do is to deduce the “causes” (parameters) from the “effects” (observations). Bayes theorem can be seen as an inversion procedure expressed in a probabilistic context. Indeed, given the set of observations, say, \mathcal{X} , which are controlled by the unknown set of parameters, we write

$$p(\theta|\mathcal{X}) = \frac{p(\mathcal{X}|\theta)p(\theta)}{p(\mathcal{X})}.$$

All that is needed for the above inversion is to have a guess about $p(\theta)$. This term has brought a lot of controversy in the statistical community for a number of years. However, once a reasonable guess of the prior is available, a number of advantages associated with the Bayesian approach emerge, compared to the alternative route. The latter embraces methods that view the parameters deterministically as constants of unknown values, and they are also referred to as *frequentist* techniques. The term comes from the more classical view of probabilities as frequencies of occurrence of repeatable events. A typical example of this family of methods is the maximum likelihood approach, which estimates the values of the parameters by maximizing $p(\mathcal{X}|\theta)$; its value is solely controlled by the obtained observations in a sequence of experiments.

This is the first of two chapters dedicated to Bayesian learning. We present the main concepts and philosophy behind Bayesian inference. We introduce the expectation-maximization (EM) algorithm and apply it in some typical machine learning parametric modeling tasks, such as regression, mixture modeling, and mixture of experts. Finally, the exponential family of distributions is introduced and the notion of conjugate priors is discussed.

12.2 REGRESSION: A BAYESIAN PERSPECTIVE

The Bayesian inference treatment of the linear regression task was introduced in [Chapter 3](#). In the current chapter, we go beyond the basic definitions and reveal and exploit various possibilities that the Bayesian philosophy offers to the study of this important machine learning task. Let us first summarize the findings of [Chapter 3](#) and then start building upon them.¹

¹ Recall our adopted notation: random variables and vectors are denoted with roman and their respected measured values/observations with Times Roman fonts.

Recall the (generalized) linear regression task, as it was introduced in previous chapters, that is,

$$y = \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) + \eta = \theta_0 + \sum_{k=1}^{K-1} \theta_k \phi_k(\mathbf{x}) + \eta, \quad (12.1)$$

where $y \in \mathbb{R}$ is the output random variable, $\mathbf{x} \in \mathbb{R}^l$ is the input random vector, $\eta \in \mathbb{R}$ is the noise disturbance, $\boldsymbol{\theta} \in \mathbb{R}^K$ is the unknown parameter vector, and

$$\boldsymbol{\phi}(\mathbf{x}) := [\phi_1(\mathbf{x}), \dots, \phi_{K-1}(\mathbf{x}), 1]^T$$

where $\phi_k(\cdot)$, $k = 1, \dots, K - 1$, are some (fixed) basis functions. As we already know, typical examples of such functions can be the Gaussian function, splines, monomials, and others. We are given a set of N output-input training points, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$. In our current setting, we assume that the respective (unobserved) noise values, η_n , $n = 1, 2, \dots, N$, are samples of jointly Gaussian distributed random variables with covariance matrix Σ_η , that is,

$$p(\eta) = \frac{1}{(2\pi)^{N/2} |\Sigma_\eta|^{1/2}} \exp\left(-\frac{1}{2} \eta^T \Sigma_\eta^{-1} \eta\right), \quad (12.2)$$

where $\boldsymbol{\eta} = [\eta_1, \eta_2, \dots, \eta_N]^T$.

12.2.1 THE MAXIMUM LIKELIHOOD ESTIMATOR

The maximum likelihood (ML) method was introduced in [Chapter 3](#). According to the method, the unknown parameter is treated as a deterministic variable $\boldsymbol{\theta}$, which parameterizes the pdf describing the output vector of observations

$$y = \Phi \boldsymbol{\theta} + \eta, \quad (12.3)$$

where

$$\Phi = \begin{bmatrix} \boldsymbol{\phi}^T(\mathbf{x}_1) \\ \boldsymbol{\phi}^T(\mathbf{x}_2) \\ \vdots \\ \boldsymbol{\phi}^T(\mathbf{x}_N) \end{bmatrix}, \quad (12.4)$$

and

$$\mathbf{y} = [y_1, y_2, \dots, y_N]^T.$$

A simple replacement of X with Φ in [\(3.59\)](#) changes the ML estimate to

$$\hat{\boldsymbol{\theta}}_{\text{ML}} = (\Phi^T \Sigma_\eta^{-1} \Phi)^{-1} \Phi^T \Sigma_\eta^{-1} \mathbf{y}. \quad (12.5)$$

For the simple case of uncorrelated noise samples of equal variance σ_η^2 ($\Sigma_\eta = \sigma_\eta^2 I$), Eq. [\(12.5\)](#) becomes identical to the least-squares (LS) solution

$$\hat{\boldsymbol{\theta}}_{\text{ML}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y} = \hat{\boldsymbol{\theta}}_{\text{LS}}. \quad (12.6)$$

A major drawback of the ML approach is that it is vulnerable to overfitting, because no care is taken for complex models that try to “learn” the specificities of the particular training set, as already discussed in [Chapter 3](#).

12.2.2 THE MAP ESTIMATOR

According to the maximum a posteriori probability (MAP) method, the unknown set of parameters is treated as a random vector θ and its posterior, for a given set of output observations, y , is expressed as

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}, \quad (12.7)$$

where $p(\theta)$ is the associated prior pdf. We have eliminated from the notation the dependence on \mathcal{X} , to make it look simpler. We emphasize that the input set, $\mathcal{X} = \{x_1, \dots, x_N\}$, is considered fixed, so all the randomness associated with y is due to the noise source. Assuming both the prior as well as the conditional pdfs to be Gaussians,² that is,

$$p(\theta) = \mathcal{N}(\theta|\theta_0, \Sigma_\theta), \quad (12.8)$$

and

$$p(y|\theta) = \mathcal{N}(y|\Phi\theta, \Sigma_\eta), \quad (12.9)$$

where (12.2), (12.3) have been used, the posterior $p(\theta|y)$ turns out also to be Gaussian with mean vector

$$\mu_{\theta|y} := \mathbb{E}[\theta|y] = \theta_0 + \left(\Sigma_\theta^{-1} + \Phi^T \Sigma_\eta^{-1} \Phi \right)^{-1} \Phi^T \Sigma_\eta^{-1} (y - \Phi\theta_0). \quad (12.10)$$

Because the maximum of a Gaussian coincides with its mean, we have that

$$\hat{\theta}_{\text{MAP}} = \mathbb{E}[\theta|y]. \quad (12.11)$$

In the chapter's appendix, [Section 12.9](#), an analytical proof of (12.10) is provided.³ It suffices to replace in (12.140) $t \rightarrow y$, $z \rightarrow \theta$, $A \rightarrow \Phi$, $\Sigma_{t|z} \rightarrow \Sigma_\eta$, and $\Sigma_z \rightarrow \Sigma_\theta$. Note that the MAP estimate is a regularized version of $\hat{\theta}_{\text{ML}}$. Regularization is achieved via θ_0 and Σ_θ , which are imposed by the prior $p(\theta)$. If one assumes $\Sigma_\theta = \sigma_\theta^2 I$, $\Sigma_\eta = \sigma_\eta^2 I$, and $\theta_0 = \mathbf{0}$, then (12.10) coincides with the solution of the regularized LS (ridge) regression,⁴

$$\hat{\theta}_{\text{MAP}} = (\lambda I + \Phi^T \Phi)^{-1} \Phi^T y, \quad (12.12)$$

where we have set $\lambda := \frac{\sigma_\eta^2}{\sigma_\theta^2}$. We already know from [Chapter 3](#) that the value of λ is critical to the performance of the estimator with respect to the mean-square error (MSE) performance. The main issue now becomes how to choose a good value for λ , or equivalently for Σ_θ , Σ_η in the more general case. In practice, the cross-validation method ([Chapter 3](#)) is employed; different values of λ are tested and the one that leads to the best MSE (or some other criterion) is selected. However, this is a computationally costly procedure, especially for complex models, where a large number of parameters is involved. Moreover, such a procedure forces us to use only a fraction of the available data for training, to reserve the rest for testing. The reader may wonder why we do not use the training data to optimize with

² Because in this chapter many random variables will be involved, we explicitly state the name of the variable to which we refer in $\mathcal{N}(\cdot|\cdot, \cdot)$.

³ Because the appendix serves the needs of various parts of the book, each time involving different variables, one has to make the necessary notational substitutions.

⁴ Recall from [Section 3.8](#), that this is valid if either the data have been centered or the intercept (bias) is involved in the regularizing norm term.

respect to both the unknown parameter θ as well as the regularization parameters. Let us consider as an example the simpler case of ridge regression, for centered data ($\theta_0 = 0$). The cost function comprises two terms, one that is data-dependent and measures the misfit, whereas the second one depends only on the unknown parameter

$$J(\theta, \lambda) = \|y - \Phi\theta\|^2 + \lambda\|\theta\|^2. \quad (12.13)$$

It is obvious that the only value of λ that leads to the minimum squared error fit *over the training data set* (empirical loss) is $\lambda = 0$. Any other value of λ would result in an estimate of θ which scores larger values of the squared error term; this is natural, because for $\lambda \neq 0$ the optimization has to take care for the extra regularizing term, too. It is only when *test data sets* are employed, where values of $\lambda \neq 0$ lead to an overall decrease of the mean-square error (not the empirical one).

12.2.3 THE BAYESIAN APPROACH

The Bayesian approach to regression attempts to overcome the previously reported drawbacks, which are associated with the overfitting. All the involved parameters can be estimated on the training set. In this vein, the parameters will be treated as random variables. At the same time, because the main task now becomes that of inferring the pdf that describes the unknown set of parameters, instead of obtaining a single vector estimate, one has more information at her/his disposal. Having said that, it does not mean that Bayesian techniques are necessarily free from cross-validation; this will be needed to assess their overall performance. We will comment further on this in the Remarks at the end of [Section 12.3](#).

As we know, the starting point is the same as that for MAP, and in particular [\(12.7\)](#). However, instead of taking just the maximum of the numerator in [\(12.7\)](#), we will make use of $p(\theta|y)$ as a whole. Most of the secrets here lie in the denominator $p(y)$, which is basically the normalizing constant,

$$p(y) = \int p(y|\theta)p(\theta) d\theta. \quad (12.14)$$

As we will soon see, there is much more information hidden in $p(y)$ that goes beyond the need of just computing $p(\theta|y)$. The difficulty with [\(12.14\)](#) is that, in general, the evaluation of the integral cannot be performed analytically. In such cases, one has to resort to approximate techniques, to obtain the required information. To this end, a number of approaches are available, and a large part of this book is dedicated to their study. More specifically, the following methods have been proposed and will be considered:

- The Laplacian approximation method, presented in [Section 12.3](#).
- The variational approximation method, presented in [Section 13.2](#).
- The variational bound approximation method, presented in [Section 13.8](#).
- Monte Carlo techniques for the evaluation of the integral, which are discussed in [Chapter 14](#).
- Message passing algorithms, to be discussed in [Chapter 15](#).

For the case under study in this section, where $p(y|\theta)$ and $p(\theta)$ are both assumed to be Gaussians, $p(y)$ can be evaluated analytically; it turns out that the joint distribution $p(y, \theta)$ is also Gaussian and hence the marginal $p(y)$ is Gaussian as well. All these are shown in detail in the appendix of [Section 12.9](#), at the end of the chapter. Indeed, if we set in [\(12.143\)](#) and [\(12.148\)](#) of [Section 9](#) $z \rightarrow \theta$, $t \rightarrow y$, and $A \rightarrow \Phi$, it turns out that for the regression model of [\(12.3\)](#) and the prior pdf in [\(12.8\)](#) as well as the noise model of [\(12.2\)](#), we obtain that,

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y} | \Phi\boldsymbol{\theta}_0, \Sigma_\eta + \Phi\Sigma_\theta\Phi^T). \quad (12.15)$$

Moreover, the posterior $p(\boldsymbol{\theta}|\mathbf{y})$ is also Gaussian,

$$p(\boldsymbol{\theta}|\mathbf{y}) = \mathcal{N}(\boldsymbol{\theta} | \boldsymbol{\mu}_{\boldsymbol{\theta}|\mathbf{y}}, \Sigma_{\boldsymbol{\theta}|\mathbf{y}}), \quad (12.16)$$

where $\boldsymbol{\mu}_{\boldsymbol{\theta}|\mathbf{y}}$ is given by (12.10) and the covariance matrix results from (12.144), after the appropriate notational substitutions, that is,

$$\Sigma_{\boldsymbol{\theta}|\mathbf{y}} = \left(\Sigma_\theta^{-1} + \Phi^T \Sigma_\eta^{-1} \Phi \right)^{-1}. \quad (12.17)$$

The posterior pdf in (12.16) encapsulates our knowledge about $\boldsymbol{\theta}$, after the observations \mathbf{y} have been obtained. Hence, our uncertainty about $\boldsymbol{\theta}$ has been reduced, which is the main reason that (12.16) is different from the prior pdf in (12.8); the latter represents only our initial guess. The covariance matrix in (12.17) provides the information about our uncertainty with respect to $\boldsymbol{\theta}$. If the Gaussian in (12.16) is very broad around its mean $\boldsymbol{\mu}_{\boldsymbol{\theta}|\mathbf{y}}$, it indicates that in spite of the reception of the observations still much uncertainty about $\boldsymbol{\theta}$ remains. This can be due (a) to the nature of the problem, for example, high noise variance, as this is conveyed by Σ_η , (b) and/or to the number of observations, N , which may not be enough, (c) and/or to modeling inaccuracies, as this is conveyed by Φ in (12.17). The opposite comments are in order if the posterior pdf is sharply peaked around its mean.

As we have already stated in Chapter 3, the Bayesian philosophy provides the means for a direct inference of the output variable, which in many applications is the quantity of interest; given the input vector, the task is to predict the output. In such cases, estimating a value for the unknown $\boldsymbol{\theta}$ is only the means to an end. To formulate the prediction task directly, without involving $\boldsymbol{\theta}$, one has to integrate the contribution of $\boldsymbol{\theta}$. Having learned the posterior $p(\boldsymbol{\theta}|\mathbf{y})$, then given a new input vector \mathbf{x} , for the regression model in (12.1), the conditional pdf of the output variable, y , given the set of observations is written as,

$$p(y|\mathbf{x}, \mathbf{y}) = \int p(y|\mathbf{x}, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathbf{y}) d\boldsymbol{\theta}. \quad (12.18)$$

Note that we have used $p(y|\mathbf{x}, \boldsymbol{\theta}) = p(y|\mathbf{x}, \boldsymbol{\theta})$ because y is conditionally independent of \mathbf{y} given the value of $\boldsymbol{\theta}$. As it has already been stated, strictly speaking, the posterior should have been denoted as $p(\boldsymbol{\theta}|\mathbf{y}; \mathcal{X})$ to indicate the dependence on the input training samples. However, the dependence on \mathcal{X} has been suppressed to unclutter notation.

In the sequel, and in order to simplify algebra and focus on the concepts, we assume that the noise model in (12.2) is such that $\Sigma_\eta = \sigma_\eta^2 I$ and also $\Sigma_\theta = \sigma_\theta^2 I$ for the prior pdf in (12.8). Then, we have that

$$p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y | \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}), \sigma_\eta^2),$$

and (12.17), (12.10) for the posterior covariance matrix and mean, respectively, become

$$\Sigma_{\boldsymbol{\theta}|\mathbf{y}} = \left(\frac{1}{\sigma_\theta^2} I + \frac{1}{\sigma_\eta^2} \Phi^T \Phi \right)^{-1}, \quad (12.19)$$

$$\boldsymbol{\mu}_{\boldsymbol{\theta}|\mathbf{y}} = \boldsymbol{\theta}_0 + \frac{1}{\sigma_\eta^2} \left(\frac{1}{\sigma_\theta^2} I + \frac{1}{\sigma_\eta^2} \Phi^T \Phi \right)^{-1} \Phi^T (\mathbf{y} - \Phi \boldsymbol{\theta}_0). \quad (12.20)$$

The integration in (12.18) can now be carried out analytically as in (12.133), (12.134), and using (12.147), (12.148) in Section 12.9, with $z \rightarrow \theta$, $t \rightarrow y$, $A \rightarrow \phi^T$, $\mu_z \rightarrow \mu_{\theta|y}$, $\Sigma_z \rightarrow \Sigma_{\theta|y}$, $\Sigma_{t|z} \rightarrow \sigma_\eta^2$, and we obtain that,

$$p(y|\mathbf{x}, \mathbf{y}) = \mathcal{N}(y|\mu_y, \sigma_y^2) : \quad \text{Predictive Distribution,} \quad (12.21)$$

where

$$\mu_y = \phi^T(\mathbf{x})\mu_{\theta|y}, \quad (12.22)$$

$$\begin{aligned} \sigma_y^2 &= \sigma_\eta^2 + \phi^T(\mathbf{x})\Sigma_{\theta|y}\phi \\ &= \sigma_\eta^2 + \phi^T(\mathbf{x})\left(\frac{1}{\sigma_\theta^2}I + \frac{1}{\sigma_\eta^2}\Phi^T\Phi\right)^{-1}\phi \\ &= \sigma_\eta^2 + \sigma_\eta^2\sigma_\theta^2\phi^T(\mathbf{x})\left(\sigma_\eta^2I + \sigma_\theta^2\Phi^T\Phi\right)^{-1}\phi. \end{aligned} \quad (12.23)$$

Hence, given \mathbf{x} one can predict the respective value of y using the most probable value, that is, μ_y in (12.22). Note that the same prediction value would result via the MAP estimate in (12.10) (or (12.12), if θ_0 , also obtained via the ridge regression task). Have we then gained anything extra by adopting the Bayesian approach? The answer is in the affirmative. *More information concerning the predicted value is now available, because (12.23) quantifies the associated uncertainty.*

To investigate (12.23) further, let us simplify it by adopting the following approximation

$$R_\phi := \mathbb{E}[\phi(\mathbf{x})\phi^T(\mathbf{x})] \simeq \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n)\phi^T(\mathbf{x}_n) = \frac{1}{N}\Phi^T\Phi,$$

or

$$\Phi^T\Phi \simeq NR_\phi, \quad (12.24)$$

where R_ϕ is the autocorrelation matrix of the random vector $\phi(\mathbf{x})$. Employing (12.24) into (12.23), leads to

$$\sigma_y^2 \simeq \sigma_\eta^2 \left(1 + \sigma_\theta^2\phi^T(\mathbf{x})\left(\sigma_\eta^2I + N\sigma_\theta^2R_\phi\right)^{-1}\phi\right), \quad (12.25)$$

which for large enough N becomes

$$\sigma_y^2 \simeq \sigma_\eta^2 \left(1 + \frac{1}{N}\phi^T(\mathbf{x})R_\phi^{-1}\phi\right).$$

Thus, for a large number of observations, $\sigma_y^2 \rightarrow \sigma_\eta^2$, and our uncertainty is contributed by the noise source, which cannot be reduced anymore. For smaller values of N , there is extra uncertainty associated with the parameter θ , measured by σ_θ^2 in (12.25).

So far in this section, we dealt with Gaussians, which led to tractable and analytically computed integrals. Are there ways to attack more general cases? Moreover, even in the case of Gaussian pdfs, we have assumed the covariance matrices Σ_θ , Σ_η to be known. In practice, they are not. Even if one assumes that Σ_η can be experimentally measured, there still remains Σ_θ . Can one select the related parameters via an optimization process? If the answer is yes, can this optimization be carried out on

the training set, or one would necessarily run into problems similar to the ones we faced with the regularization approach? We will indulge in all these challenges in the sections to follow.

Remarks 12.1.

- The MAP estimator is sometimes referred to as *Type I* estimator, to be distinguished from the Type II estimation method, which will be discussed in [Remarks 12.2](#), in the next section.
- The posterior mean in (12.10) can be met in different variants, which are obtained via the application of the matrix inversion lemmas given in [Appendix A.1](#). In [Section 12.9](#), it is shown that (Eq. (12.149))

$$\boldsymbol{\mu}_{\theta|y} = \left(\boldsymbol{\Sigma}_{\theta}^{-1} + \boldsymbol{\Phi}^T \boldsymbol{\Sigma}_{\eta}^{-1} \boldsymbol{\Phi} \right)^{-1} \left(\boldsymbol{\Phi}^T \boldsymbol{\Sigma}_{\eta}^{-1} \mathbf{y} + \boldsymbol{\Sigma}_{\theta}^{-1} \boldsymbol{\theta}_0 \right) \quad (12.26)$$

or (Eq. (12.145))

$$\boldsymbol{\mu}_{\theta|y} = \boldsymbol{\theta}_0 + \boldsymbol{\Sigma}_{\theta} \boldsymbol{\Phi}^T \left(\boldsymbol{\Sigma}_{\eta} + \boldsymbol{\Phi} \boldsymbol{\Sigma}_{\theta} \boldsymbol{\Phi}^T \right)^{-1} (\mathbf{y} - \boldsymbol{\Phi} \boldsymbol{\theta}_0). \quad (12.27)$$

Also, using Woodbury's identity from [Appendix A.1](#), we can readily see that

$$\boldsymbol{\Sigma}_{\theta|y} = \boldsymbol{\Sigma}_{\theta} - \boldsymbol{\Sigma}_{\theta} \boldsymbol{\Phi}^T \left(\boldsymbol{\Sigma}_{\eta} + \boldsymbol{\Phi} \boldsymbol{\Sigma}_{\theta} \boldsymbol{\Phi}^T \right)^{-1} \boldsymbol{\Phi} \boldsymbol{\Sigma}_{\theta}. \quad (12.28)$$

In practice, one uses the most computationally convenient form, depending on the dimensionality of the involved matrices to invert the one of lower dimension.

Example 12.1. This example demonstrates the prediction task summarized in (12.22), (12.23). Data are generated based on the following nonlinear model,

$$y_n = \theta_0 + \theta_1 x_n + \theta_2 x_n^2 + \theta_3 x_n^3 + \theta_5 x_n^5 + \eta_n, \quad n = 1, 2, \dots, N,$$

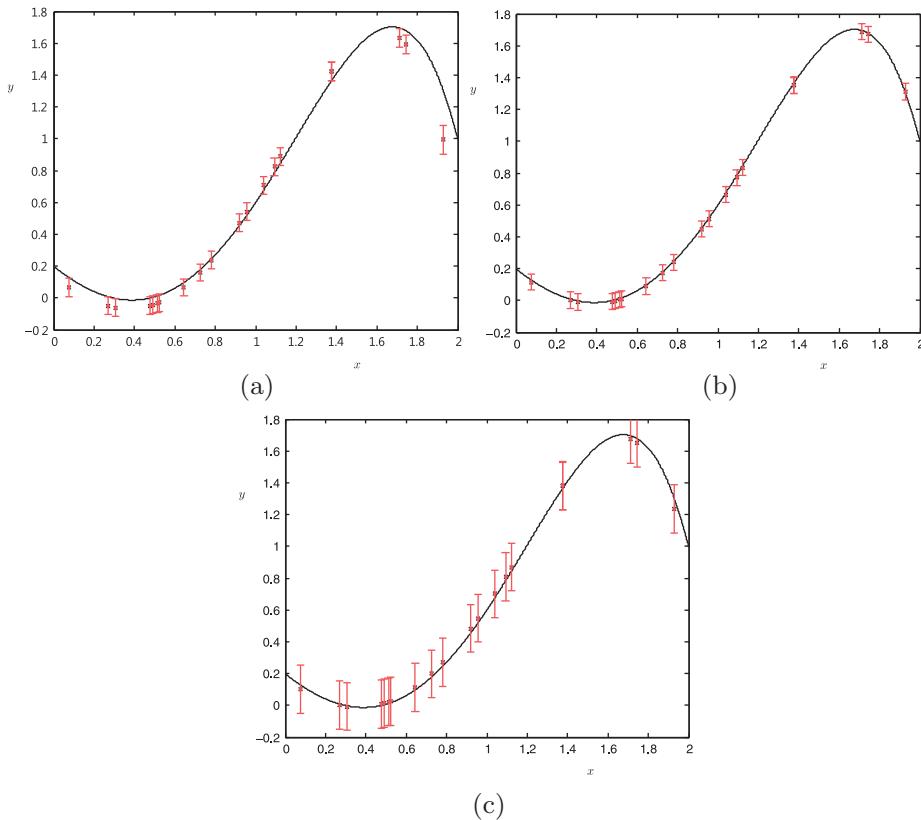
where η_n are i.i.d. noise samples drawn from a zero mean Gaussian with variance σ_{η}^2 . Samples x_n are equidistant points in the interval $[0, 2]$. The goal of the task is to predict the value y given a measured value x , using (12.22). The parameter values used to generate the data were equal to,

$$\theta_0 = 0.2, \theta_1 = -1, \theta_2 = 0.9, \theta_3 = 0.7, \theta_5 = -0.2.$$

- (a) In the first set of experiments, a Gaussian prior for the unknown Θ was used with mean $\boldsymbol{\theta}_0$ equal to the previous true set of parameters and $\boldsymbol{\Sigma}_{\theta} = 0.1I$. Also, the true model structure was used to construct the matrix $\boldsymbol{\Phi}$. [Figure 12.1a](#) shows the points (y, x) in red together with the error bars, as measured by the computed σ_y^2 , for the case of $N = 20$ training points and $\sigma_{\eta}^2 = 0.05$. [Figure 12.1b](#) demonstrates the obtained improvement when the training points are increased to $N = 500$, while keeping the values of the other two parameters unchanged. [Figure 12.1c](#) corresponds to the latter case, where the noise variance is increased to $\sigma_{\eta}^2 = 0.15$.
- (b) In the second set of experiments, we kept the correct model, however, the mean of the prior was given a different value to that of the true model, namely,

$$\boldsymbol{\theta}_0 = [-10.54, 0.465, 0.0087, -0.093, -0.004]^T.$$

[Figure 12.2a](#) corresponds to the case of $\sigma_{\eta}^2 = 0.05$, $N = 20$, and $\sigma_{\theta}^2 = 0.1$. Note the improvement that is obtained when increasing $\sigma_{\theta}^2 = 2$, shown in [Figure 12.2b](#), while N and σ_{η}^2 remain the same as before; this is because the model takes into consideration our uncertainty about the prior mean being away from the true value. [Figure 12.2c](#) corresponds to $\sigma_{\eta}^2 = 0.05$, $N = 500$, and $\sigma_{\theta}^2 = 0.1$ and shows the advantage of using a large number of training points.

**FIGURE 12.1**

Each one of the red points, (y, x) , indicates the prediction (y) corresponding to the input value, (x) . The error bars are dictated by the computed variance, σ_y^2 . The mean values used in the Gaussian prior are equal to the true values of the unknown model. (a) $\sigma_\eta^2 = 0.05$, $N = 20$, $\sigma_\theta^2 = 0.1$. (b) $\sigma_\eta^2 = 0.05$, $N = 500$, $\sigma_\theta^2 = 0.1$. (c) $\sigma_\eta^2 = 0.15$, $N = 500$, $\sigma_\theta^2 = 0.1$. Observe that the larger the data set, the better the predictions are and the larger the noise variance, the larger the error bars become.

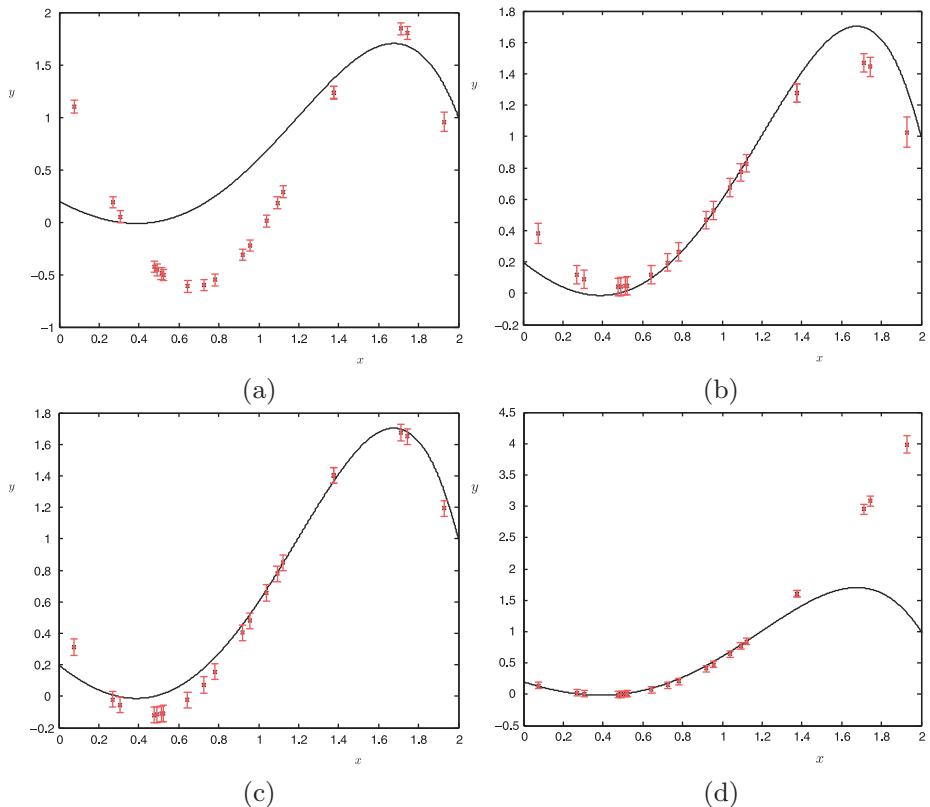
(c) Figure 12.2d, corresponds to the case where the adopted model for prediction is the wrong one, that is,

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \eta.$$

The used values were $\sigma_\eta^2 = 0.05$, $N = 500$, and $\sigma_\theta^2 = 2$. Observe that once a wrong model has been adopted, one must not have “high expectations” for good prediction performance.

12.3 THE EVIDENCE FUNCTION AND OCCAM'S RAZOR RULE

In the previous section, we made a comment about the importance of the marginal pdf $p(y)$. This section is fully dedicated to this quantity. In the notation used in (12.14), we did silently suppress the dependence on the adopted model. For example, the Gaussian assumption for the prior in (12.8) and for

**FIGURE 12.2**

In this set of figures the mean values of the prior are different from that of the true model. (a) $\sigma_\eta^2 = 0.05$, $N = 20$, $\sigma_\theta^2 = 0.1$. (b) $\sigma_\eta^2 = 0.05$, $N = 20$, $\sigma_\theta^2 = 2$; observe the effect of using larger variance for the prior. (c) $\sigma_\eta^2 = 0.05$, $N = 500$, $\sigma_\theta^2 = 0.1$; observe the effect of the larger training data set. (d) The points correspond to a wrong model.

the conditional in (12.9) should have been reflected in the marginal as $p(\mathbf{y}; \boldsymbol{\phi}, \Sigma_\eta, \Sigma_\theta)$, because different Gaussians, different basis functions, and different orders, K , of the model can be used. Furthermore, non-Gaussian pdfs can also be adopted. In a more general setting, let us make the dependence on the model explicit as $p(\mathbf{y}|\mathcal{M}_i)$. Assuming the choice of a model to be random, then mobilizing Bayes theorem once more, we have

$$P(\mathcal{M}_i|\mathbf{y}) = \frac{P(\mathcal{M}_i)p(\mathbf{y}|\mathcal{M}_i)}{p(\mathbf{y})}, \quad (12.29)$$

where

$$p(\mathbf{y}) = \sum_i P(\mathcal{M}_i)p(\mathbf{y}|\mathcal{M}_i), \quad (12.30)$$

and $P(\mathcal{M}_i)$ is the prior probability of \mathcal{M}_i . $P(\mathcal{M}_i)$ provides a measure of the subjective prior over all possible models, which expresses our guess on how plausible a model is with respect to alternative

ones, prior to the data arrival. Because the denominator in (12.29) is independent of the model, one can obtain the most probable model, after observing \mathbf{y} , by maximizing the numerator. If one assigns to all possible models equal probabilities, then detecting the most probable model under the given set of observations becomes a task of maximizing $p(\mathbf{y}|\mathcal{M}_i)$. This is the reason that this pdf is known as the *evidence function* for the model or simply as the *evidence*. In practice, we content ourselves with using the most probable model, although an orthodox Bayesian would suggest to average all obtained quantities over all possible models, as in (12.30). In an ideal Bayesian setting, one does not choose among models; predictions are performed by summing over all possible models, each one weighted by the respective probability. However, in many practical problems we may have reasons to suggest that the evidence function is strongly peaked around a specific model; after all, such an assumption may simplify the task considerably.

We now turn our attention to what is hidden behind the optimization of $p(\mathbf{y}|\mathcal{M}_i)$ with respect to different models. Before we proceed, it is worth making a comment. A superficial first look may lead one to think whether this is any different from maximizing the likelihood $p(\mathbf{y}; \boldsymbol{\theta})$, as it was done in Section 12.2.1. As a matter of fact, the two cases belong to two different worlds. ML maximizes with respect to a single (vector) parameter within an adopted model, and this is the weak point that makes ML vulnerable to overfitting. Maximizing the evidence is an optimization task with respect to the model itself, a wise alternative that guards us against overfitting, as we explain next.

From (12.14) we have

$$p(\mathbf{y}|\mathcal{M}_i) = \int p(\mathbf{y}|\mathcal{M}_i, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{M}_i)d\boldsymbol{\theta} : \text{ Evidence Function.} \quad (12.31)$$

Let us assume for simplicity that $\boldsymbol{\theta}$ is a scalar, $\boldsymbol{\theta} \in \mathbb{R}$, and that the integrand in (12.31), which according to the Bayes theorem is analogous to the posterior $p(\boldsymbol{\theta}|\mathbf{y}, \mathcal{M}_i)$, peaks around a value; this is obviously the value that would result as the MAP estimate, $\hat{\boldsymbol{\theta}}_{\text{MAP}}$. Figure 12.3 illustrates the respective graphs. Thus, (12.31) can be approximated by

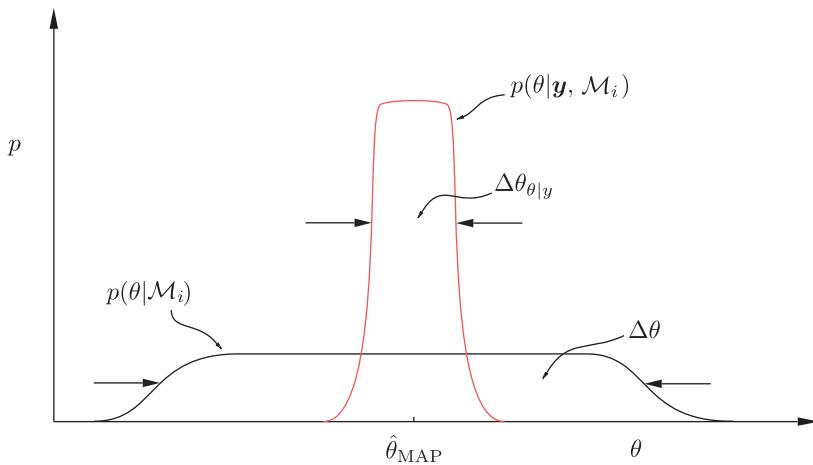
$$p(\mathbf{y}|\mathcal{M}_i) \simeq p(\mathbf{y}|\mathcal{M}_i, \hat{\boldsymbol{\theta}}_{\text{MAP}})p(\hat{\boldsymbol{\theta}}_{\text{MAP}}|\mathcal{M}_i)\Delta\theta_{\boldsymbol{\theta}|\mathbf{y}}. \quad (12.32)$$

To get a better feeling for each one of the factors involved in (12.32), let us also assume that the prior pdf is (almost) uniform with a width equal to $\Delta\boldsymbol{\theta}$. Then, (12.32) is rewritten as

$$p(\mathbf{y}|\mathcal{M}_i) \simeq p(\mathbf{y}|\mathcal{M}_i, \hat{\boldsymbol{\theta}}_{\text{MAP}}) \frac{\Delta\theta_{\boldsymbol{\theta}|\mathbf{y}}}{\Delta\boldsymbol{\theta}}. \quad (12.33)$$

The first factor in the product on the right-hand side in (12.33) coincides with the likelihood function at its optimal value, because for this case of uniform prior, $\hat{\boldsymbol{\theta}}_{\text{MAP}} = \hat{\boldsymbol{\theta}}_{\text{ML}}$. In other words, this factor provides us with the best fit that model \mathcal{M}_i can achieve on the given set of observations. However, now, in contrast to the ML method, the evidence function also depends on the second factor, $\frac{\Delta\theta_{\boldsymbol{\theta}|\mathbf{y}}}{\Delta\boldsymbol{\theta}}$. As it has been pointed out in the insightful papers [14, 28, 30], this term accounts for the complexity of the model, and it is named the Occam factor for obvious reasons. Let us elaborate on this a bit more by following the reasoning given in [30].

The Occam factor penalizes those models, which are finely tuned to the received observations. As an example, if two different models \mathcal{M}_i and \mathcal{M}_j have a similar range of values for their prior pdfs, then if, say, $\Delta\theta_{\boldsymbol{\theta}|\mathbf{y}}(\mathcal{M}_i) \ll \Delta\theta_{\boldsymbol{\theta}|\mathbf{y}}(\mathcal{M}_j)$ then \mathcal{M}_i will be penalized more; only a small range of values for $\boldsymbol{\theta}$ survive (i.e., correspond to high probability values) after the reception of \mathbf{y} . So, if this fine-tuned

**FIGURE 12.3**

The posterior peaks around the value $\hat{\theta}_{\text{MAP}}$ and the posterior pdf can be approximated by $p(\hat{\theta}_{\text{MAP}}|y; \mathcal{M}_i)$ over an interval of values equal to $\Delta\theta_{\theta|y}$.

(to the data) model, \mathcal{M}_i , had resulted in a large value of the ML term, it is not certain that the evidence would be maximized for it, because the Occam factor would be small. Which model, between the two, finally wins it depends on the product of the two involved terms. Soon we will see that the Occam term is also related to the number of parameters; that is, to the complexity of the adopted model.

Laplacian approximation and the evidence function

To investigate the evidence function for the general multiparameter case, we will employ the method of Laplacian approximation of a pdf. This is a general methodology that approximates any pdf *locally* in terms of a Gaussian one. To this end, define⁵

$$g(\boldsymbol{\theta}) = \ln(p(y|\mathcal{M}_i, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{M}_i)). \quad (12.34)$$

Use Taylor's expansion around $\hat{\boldsymbol{\theta}}_{\text{MAP}}$ and keep terms up to the second order,

$$\begin{aligned} g(\boldsymbol{\theta}) &= g(\hat{\boldsymbol{\theta}}_{\text{MAP}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^T \frac{\partial g(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{\text{MAP}}} \\ &\quad + \frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^T \frac{\partial^2 g(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2} \Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{\text{MAP}}} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}}) \\ &= g(\hat{\boldsymbol{\theta}}_{\text{MAP}}) - \frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^T \Sigma^{-1} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}}), \end{aligned} \quad (12.35)$$

where

$$\Sigma^{-1} := -\frac{\partial^2 g(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^2} \Big|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}_{\text{MAP}}},$$

⁵ Similarly, to obtain the Laplacian approximation of a general pdf, $p(\mathbf{x})$, we set $g(\mathbf{x}) = \ln p(\mathbf{x})$.

which leads to the approximation,

$$\begin{aligned} p(\mathbf{y}|\mathcal{M}_i, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{M}_i) &\simeq p(\mathbf{y}|\mathcal{M}_i, \hat{\boldsymbol{\theta}}_{\text{MAP}})p(\hat{\boldsymbol{\theta}}_{\text{MAP}}|\mathcal{M}_i) \times \\ &\quad \exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^T \Sigma^{-1}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})\right). \end{aligned} \quad (12.36)$$

Plugging (12.36) into the integral of (12.31) we obtain

$$p(\mathbf{y}|\mathcal{M}_i) = p(\mathbf{y}|\mathcal{M}_i, \hat{\boldsymbol{\theta}}_{\text{MAP}})p(\hat{\boldsymbol{\theta}}_{\text{MAP}}|\mathcal{M}_i)(2\pi)^{\frac{K}{2}}|\Sigma|^{1/2}, \quad (12.37)$$

and taking the logarithms we have

$$\underbrace{\ln p(\mathbf{y}|\mathcal{M}_i)}_{\text{Evidence}} = \underbrace{\ln p(\mathbf{y}|\mathcal{M}_i, \hat{\boldsymbol{\theta}}_{\text{MAP}}) + \ln p(\hat{\boldsymbol{\theta}}_{\text{MAP}}|\mathcal{M}_i)}_{\text{Best likelihood fit}} + \underbrace{\frac{K}{2}\ln(2\pi) + \frac{1}{2}\ln|\Sigma|}_{\text{Occam factor}}. \quad (12.38)$$

The direct dependence of the Occam term on the complexity (number of basis functions) of the adopted model is now readily spotted. Moreover, the complexity-related Occam term depends on the prior pdf and the second derivatives (via Σ) of the posterior pdf, too; that is, it depends on how “sharp” the shape of the latter is in the K -dimensional space. In other words, the covariance term provides the “error bar” information. Hence, in a single equation, besides the number of parameters and the associated best-fit term, the evidence also takes into account information related to the associated variance; maximizing the evidence leads to the best trade-off. Figure 12.4 illustrates the essence behind the evidence maximization for model selection. If the model is too complex, it can fit well a wide range of data sets, and because

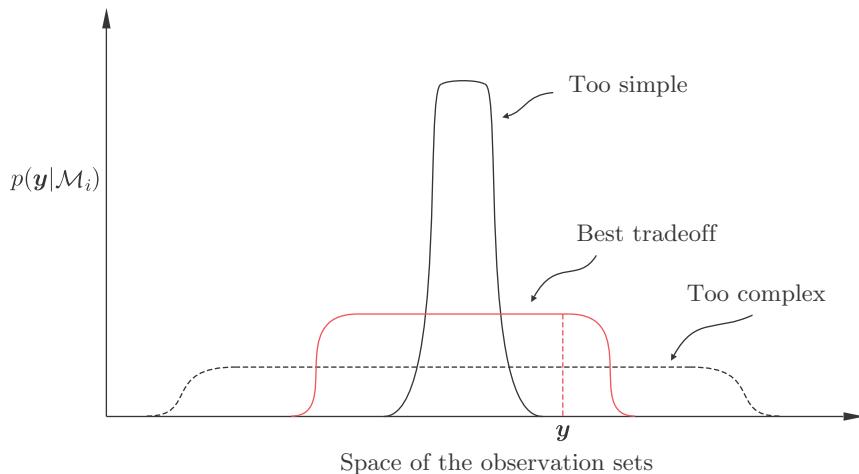


FIGURE 12.4

Too simple models can explain well a very small range of data. On the other hand, too complex models can explain a wide range of data; however, they do not provide any confidence because they assign low probability to all data sets. For the observation set, \mathbf{y} , the evidence is maximized for the model with intermediate complexity.

$p(\mathbf{y}|\mathcal{M}_i)$ has to integrate to one, its value for any value of \mathbf{y} is expected to be low. The opposite is true for models that are too simple; such models can model well some data sets but not a wide range of them, and consequently, the evidence function peaks sharply around a value in the space of observation sets. Thus, selecting a data set at random it is rather unlikely that this has been generated by such a model. Having said that, it is important to emphasize, once more, the Occam term does not depend *solely* on the number of parameters; hence, complexity here should be interpreted in a more “open-minded” way. This robustness against overfitting, which is intrinsic in the Bayesian inference approach, is the consequence of integrating the parameters for any specific model in (12.31); this integration penalizes models of high complexity because such models can model a large range of data.

Historically, the Occam’s razor rule in its Bayesian interpretation was first demonstrated in [14] and later on in [28, 39], although the foundations go back to the pioneering work of Sir Harold Jeffreys in the 1930s, [24]. Two insightful reviews on the Bayesian inference approach that are well worth reading are given in [23, 27].

Returning to (12.14) and assuming for simplicity that

$$p(\mathbf{y}|\boldsymbol{\theta}) = \mathcal{N}\left(\mathbf{y}|\Phi\boldsymbol{\theta}, \sigma_\eta^2 I\right) \quad \text{and} \quad p(\boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{\theta}|\boldsymbol{\theta}_0, \sigma_\theta^2 I\right), \quad (12.39)$$

we can express the evidence as $p(\mathbf{y}; \sigma_\eta^2, \sigma_\theta^2)$, which, for this case, turns out to be Gaussian (appendix in Section 12.9); thus, it is available in closed form. Hence for this specific case, the model space is described via σ_η^2 , σ_θ^2 and maximization of the evidence with respect to these (unknown) model parameters can take place iteratively, for the given set of observations stacked in \mathbf{y} , see, for example, [28]. However, being able to express the evidence in closed form is not the case in general. The EM algorithm, which is described in Section 12.5, is a popular way and a powerful tool to this end. One could also resort to the Laplacian approximation to approximate the involved pdfs as Gaussians, but this approximation turns out not always to be a good choice; furthermore, in high dimensional parameter spaces, the computations of the second order derivatives and the determinant can become burdensome [3].

Finally, let us make a final comment concerning the Laplacian approximation. In the discussion above, our goal was to get an approximation of the integral (normalizing constant/evidence) of $p(\mathbf{y}|\mathcal{M}_i, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{M}_i)$. However, if our interest were to approximate the pdf itself, we should be careful in selecting the normalizing constant, which by the nature of the Gaussian function leads to

$$p(\mathbf{y}|\mathcal{M}_i, \boldsymbol{\theta})p(\boldsymbol{\theta}|\mathcal{M}_i) \simeq \frac{1}{(2\pi)^{K/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})^T \Sigma^{-1} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}_{\text{MAP}})\right).$$

This is also the case for the Laplacian approximation for any pdf $p(\cdot)$.

Remarks 12.2.

- In the Bayesian approach, one makes all the modeling assumptions explicit, and it is then left to the rules of probability theory to provide the answers. One does not have to “worry” about the choice of an optimizing criterion, where different criteria lead to different estimators, and there is not an objective, systematic way to decide which criterion is best. On the other hand, in the Bayesian approach one has to make sure to select the prior that explains the data in the best possible way.
- The choice of the prior pdf is very critical in the performance of Bayesian methods and must be carried out in such a way to encapsulate prior knowledge as fully as possible. In practice, different alternatives can be adopted [3].

- *Subjective priors.* According to this path, we choose the prior $p(\theta)$ to make the manipulation of the integration tractable, by employing conjugate priors (Section 3.11.1) within the exponential family of pdfs. This family of pdfs will be presented in the next section.
- *Hierarchical priors.* Each one of the components of θ_k , $k = 0, 2, \dots, K - 1$ of θ , is controlled by a different parameter; for example, all θ_k 's may be assumed to be independent Gaussian variables, each one with a different variance. In turn, variances are considered to be random variables that follow a statistical distribution controlled by another set of deterministic (not random) parameters known as *hyperparameters*; thus, a hierarchy of priors is adopted. As we will see later on, hierarchical priors are often designed using conjugate pairs of pdfs.
- *Noninformative or objective priors.* The choice of the prior is done in such a way to embed as little extra information as possible and to exploit knowledge that is conveyed only by the available data. One way to construct such priors is to resort to information theoretic arguments. For example, one can estimate $p(\theta)$ by minimizing its Kullback-Leibler distance from $p(\theta|y)$.
- The fact that the Bayesian approach allows the recovery of all the desired information from a single data set does not suggest that the method is “cross-validation” free. Maximizing the evidence, which at the same time guards against overfitting, *does not* necessarily mean that the performance of the designed estimator is optimized. This is more true in practice where, as we are going to see very soon, most often a bound of the evidence is optimized instead, to bypass computational obstacles. As it is always the case in life, the proof of the cake is in the eating. Thus, the final verdict should only come from the generalization ability of the designed estimator; that is, its ability to make reliable predictions using data unseen to it before. However, there is no reason to suggest that the evidence may be a reliable predictor of the generalization performance. This has been known and explicitly stated since the method’s infancy stage, see, for example, [28]. The generalization performance depends heavily on whether the adopted prior matches the “true” distribution of the unknown parameters. This is nicely demonstrated with a toy example in [18]. It is shown that the Bayesian average is optimal only if the adopted prior coincides with the true one. The situation is less clear when this is not the case. A more theoretical treatment of the topic, when there is a mismatch between the true and the selected prior, can be found in [19]. Thus, to be able to assess the generalization performance of a model learned via Bayesian inference, cross-validation is required, unless an independent test set can be afforded, for example, [46].

To avoid the need for cross-validation, an alternative way has been adopted by a number of authors. The cost function, to be minimized in (12.13), is built to quantify the generalization performance of an estimator; optimization then takes place concurrently for the unknown weights as well as the regularization parameter, see, for example, [16, 34]. In general, this leads to a nonconvex optimization task and such techniques have not, yet, been widely embraced by the machine learning community.

- The Laplacian approximation to the evidence function is closely related to the *Bayesian information criterion* (BIC) [41] for model selection, which is expressed as,

$$\ln p(y|\mathcal{M}_i) \approx \ln p(y|\mathcal{M}_i, \hat{\theta}_{\text{MAP}}) - \frac{1}{2}K \ln N.$$

BIC is obtained as a large N approximation to (12.38), assuming a broad enough Gaussian prior, and manipulating a bit on the determinant involved in the last term. For a discussion including other related criteria, see [3, 43].

- The Bayesian framework is also closely related to the minimum description length (MDL) methods. The log-evidence is associated with the number of bits in the shortest message that encodes the data via model \mathcal{M}_i , for example, [47].
- *Type II maximum likelihood*: Note that the evidence is the marginal likelihood function after integrating out the parameters Θ . To distinguish it from the MAP method, when the evidence function is maximized with respect to a set of unknown parameters, it is usually referred to as *generalized maximum likelihood* or *Type II maximum likelihood* and sometimes as *empirical Bayes*. Recall from Remarks 12.1, that the MAP was also named as Type I estimator.

12.4 EXPONENTIAL FAMILY OF PROBABILITY DISTRIBUTIONS

We will treat the topic of the exponential family of probability distributions in a general setting. Let $\mathbf{x} \in \mathbb{R}^l$ be a random vector and $\boldsymbol{\theta} \in \mathbb{R}^K$ a random (parameter) vector. We say that the parameterized pdf $p(\mathbf{x}|\boldsymbol{\theta})$ is of the exponential form if

$$p(\mathbf{x}|\boldsymbol{\theta}) = g(\boldsymbol{\theta})f(\mathbf{x}) \exp(\boldsymbol{\phi}^T(\boldsymbol{\theta})\mathbf{u}(\mathbf{x})), \quad (12.40)$$

where

$$g(\boldsymbol{\theta}) = \frac{1}{\int f(\mathbf{x}) \exp(\boldsymbol{\phi}^T(\boldsymbol{\theta})\mathbf{u}(\mathbf{x})) d\mathbf{x}}, \quad (12.41)$$

is the normalizing constant of the pdf. A similar definition holds if \mathbf{x} is a discrete random variable and the respective function represents the probability mass function $P(\mathbf{x}|\boldsymbol{\theta})$; in this case, the integration in (12.41) becomes a summation. The vector $\boldsymbol{\phi}(\boldsymbol{\theta})$ comprises the set of the so-called *natural parameters*, and f , \mathbf{u} are functions defining the distribution. It is readily seen from the factorization theorem in Section 3.7 that $\mathbf{u}(\mathbf{x})$ is a *sufficient statistic* for the parameter $\boldsymbol{\theta}$. Note that an attribute of the exponential family is that the number of sufficient statistics, that is, the dimensionality of \mathbf{u} , is finite and remains independent of the number of observations. If $\boldsymbol{\phi}(\boldsymbol{\theta}) = \boldsymbol{\theta}$, then the exponential family is said to be in *canonical* form. A number of widely used distributions belongs to the exponential family, for example, the normal, exponential, gamma, chi-squared, Beta, Dirichlet, Bernoulli, binomial, multinomial distributions. Examples of distributions that do not belong in this family are the uniform with unknown bounds, the student's-t and most mixture distributions, for example, [42, 48].

An advantage of the exponential family is that one can find conjugate priors for $\boldsymbol{\theta}$; that is, priors that lead to posteriors, $p(\boldsymbol{\theta}|\mathcal{X})$, of the same functional form as $p(\boldsymbol{\theta})$ (Section 3.11.1).

Given (12.40) its conjugate prior is given by

$$p(\boldsymbol{\theta}; \lambda, \mathbf{v}) = h(\lambda, \mathbf{v})(g(\boldsymbol{\theta}))^\lambda \exp(\boldsymbol{\phi}^T(\boldsymbol{\theta})\mathbf{v}), \quad (12.42)$$

where $\lambda > 0$ and \mathbf{v} are known as *hyperparameters*; that is, parameters that control other parameters. The factor $h(\lambda, \mathbf{v})$ is an appropriate normalizing constant. It is easy to see that defining the prior as in (12.42) and the likelihood function as in (12.40), the posterior $p(\boldsymbol{\theta}|\mathbf{x})$ is of the same form as in (12.42).

Before we give some examples, let us investigate a bit more the role played by λ and \mathbf{v} , as well as the presence of $g(\boldsymbol{\theta})$ and $\boldsymbol{\phi}(\boldsymbol{\theta})$ in both (12.42) and (12.40). Assume that \mathbf{x} and $\boldsymbol{\theta}$ obey (12.40)–(12.42) and let $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ be a set of i.i.d. observations. Then,

$$p(\mathcal{X}|\boldsymbol{\theta}) = (g(\boldsymbol{\theta}))^N \prod_{n=1}^N f(\mathbf{x}_n) \exp \left(\boldsymbol{\phi}^T(\boldsymbol{\theta}) \sum_{i=1}^N \mathbf{u}(\mathbf{x}_i) \right), \quad (12.43)$$

and

$$p(\boldsymbol{\theta}|\mathcal{X}) \propto p(\mathcal{X}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \propto (g(\boldsymbol{\theta}))^{\lambda+N} \exp \left(\boldsymbol{\phi}^T(\boldsymbol{\theta}) \left(\mathbf{v} + \sum_{n=1}^N \mathbf{u}(\mathbf{x}_n) \right) \right). \quad (12.44)$$

In other words, the posterior has hyperparameters equal to

$$\tilde{\lambda} = \lambda + N, \quad \tilde{\mathbf{v}} = \mathbf{v} + \sum_{n=1}^N \mathbf{u}(\mathbf{x}_n). \quad (12.45)$$

Interpreting (12.45), one can view λ as being the effective number of observations that, implicitly, the prior information contributes to the Bayesian learning process and \mathbf{v} is the total amount of information that these (implicit) λ observations contribute to the sufficient statistic. Their exact values, basically, quantify the amount of prior knowledge that the designer wants to embed into the problem.

Example 12.2. *The Gaussian-gamma pair:* Let our random variable x be a scalar and assume that

$$p(x|\sigma^2) = \mathcal{N}(x|\mu, \sigma^2), \quad (12.46)$$

where μ is known and σ^2 is an unknown parameter. We will show that

1. $p(x|\sigma^2)$ belongs to the exponential family.

It is algebraically more convenient to work with the precision $\beta = \frac{1}{\sigma^2}$. Thus,

$$p(x|\beta) = \frac{\beta^{1/2}}{\sqrt{2\pi}} \exp \left(-\frac{1}{2}\beta(x-\mu)^2 \right). \quad (12.47)$$

Thus, $p(x|\beta)$ belongs to the exponential family with

$$f(x) = \frac{1}{\sqrt{2\pi}}, \quad \phi(\beta) = -\beta, \quad u(x) = \frac{1}{2}(x-\mu)^2,$$

and

$$g(\beta) = \frac{1}{\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} \exp(-\frac{1}{2}\beta(x-\mu)^2) dx} = \beta^{1/2}.$$

2. The conjugate prior of (12.46) follows the gamma distribution.

The respective conjugate prior from (12.42) becomes

$$p(\beta; \lambda, v) = h(\lambda, v)\beta^{\frac{\lambda}{2}} \exp(-\beta v). \quad (12.48)$$

This has the form of

$$\text{Gamma}(\beta|a, b) = \frac{1}{\Gamma(a)} b^a \beta^{a-1} \exp(-b\beta), \quad (12.49)$$

with parameters (Chapter 2) $a = \frac{\lambda}{2} + 1$ and $b = v$ and the normalizing constant, $h(\lambda, v)$, being necessarily equal to $b^a / \Gamma(a)$. The function $\Gamma(a)$ is defined as

$$\Gamma(a) = \int_0^\infty x^{a-1} e^{-x} dx.$$

If we are given multiple observations x_n , $n = 1, 2, \dots, N$, then the resulting posterior according to (12.44) and (12.45) will be a gamma distribution with

$$\tilde{b} = b + \frac{1}{2} \sum_{n=1}^N (x_n - \mu)^2 = b + \frac{N}{2} \hat{\sigma}_{\text{ML}}^2,$$

where $\hat{\sigma}_{\text{ML}}^2$ denotes the maximum likelihood estimate of the variance (Problem 3.20). Hence, the physical meaning of b is that it quantifies our prior estimate about the unknown variance. This also ties nicely with what we have said in Section 3.7; $\hat{\sigma}_{\text{ML}}^2$ is a sufficient statistic for the variance, if this is the unknown parameter in a Gaussian. It can easily be shown that the conjugate prior with respect to μ , if σ^2 is known, is a Gaussian (Problem 12.4).

In case of a multivariate Gaussian of known mean μ and unknown covariance matrix Σ (precision matrix $Q = \Sigma^{-1}$), it can also be shown that it is of the exponential form and its conjugate prior is given by the Wishart distribution,

$$\mathcal{W}(Q|W, v) = h|Q|^{\frac{v-l-1}{2}} \exp\left(-\frac{1}{2}\text{trace}\left\{W^{-1}Q\right\}\right), \quad (12.50)$$

where h is the normalizing constant (Problem 12.5) and W is an $l \times l$ matrix. The normalizing constant is given by

$$h = |W|^{-\frac{v}{2}} \left(2^{\frac{v}{2}} \pi^{\frac{l(l-1)}{4}} \prod_{i=1}^l \Gamma\left(\frac{v+1-i}{2}\right)\right)^{-1}, \quad (12.51)$$

which admittedly is quite intimidating; however, in Bayesian learning we have the luxury of bypassing the computation of the normalizing factor in (12.50). Once we express a pdf in terms of Q as in (12.50), then the normalizing constant has to be given by (12.51). The Wishart distribution is a multivariate analogue of the gamma distribution.

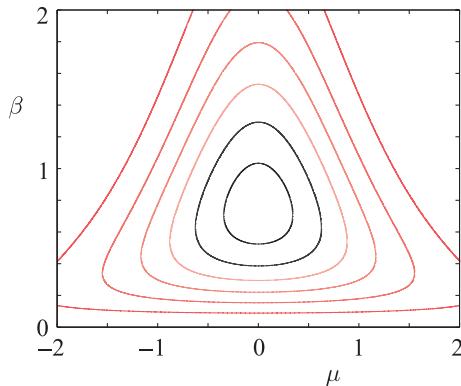
Example 12.3. *The Gaussian-Gaussian-gamma pair:* We will show that

1. $p(x; \mu, \sigma^2) = \mathcal{N}(x|\mu, \sigma^2)$ is also of an exponential form. Indeed, for this case we have

$$p(x; \mu, \sigma^2) = p(x; \mu, \beta^{-1}) = \frac{\beta^{1/2} \exp\left(-\beta \frac{\mu^2}{2}\right)}{\sqrt{2\pi}} \exp\left(\left[-\frac{\beta}{2}, \beta\mu\right] \begin{bmatrix} x^2 \\ x \end{bmatrix}\right).$$

Hence,

$$\boldsymbol{\theta} = [\beta, \mu]^T, \quad \boldsymbol{\phi}(\boldsymbol{\theta}) = \begin{bmatrix} -\frac{\beta}{2} \\ \beta\mu \end{bmatrix}, \quad \boldsymbol{u}(\boldsymbol{x}) = \begin{bmatrix} x^2 \\ x \end{bmatrix},$$

**FIGURE 12.5**

Contour plots of the Gauss-gamma distribution with parameter values $\lambda = 2, v_1 = 4, v_2 = 0$.

and performing the respective integration, we obtain

$$f(x) = \frac{1}{\sqrt{2\pi}}, g(\theta) = \beta^{1/2} \exp\left(-\frac{\beta\mu^2}{2}\right),$$

which proves the claim.

- 2.** The conjugate prior of $p(x|\mu, \sigma^2)$ is of a Gaussian-gamma form.

We have that

$$p(\mu, \beta; \lambda, v) = h(\lambda, v) \beta^{\frac{\lambda}{2}} \exp\left(-\frac{\lambda\beta\mu^2}{2}\right) \exp\left(\left[-\frac{\beta}{2}, \beta\mu\right] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}\right),$$

which after some trivial algebra ([Problem 12.6](#)) gives

$$p(\mu, \beta; \lambda, v) = \mathcal{N}\left(\mu \middle| \frac{v_2}{\lambda}, (\lambda\beta)^{-1}\right) \text{Gamma}\left(\beta \middle| \frac{\lambda+1}{2}, \frac{v_1}{2} - \frac{v_2^2}{2\lambda}\right), \quad (12.52)$$

which is known as the Gaussian-gamma distribution with the Gaussian having mean value $\mu_0 = \frac{v_2}{\lambda}$ and variance $\sigma_\mu^2 = (\lambda\beta)^{-1}$ and the defining parameters of the gamma pdf are, $a = \frac{\lambda+1}{2}$ and $b = \frac{v_1}{2} - \frac{v_2^2}{2\lambda}$. [Figure 12.5](#) shows the contour plot of the Gauss-gamma distribution of (12.6).

For the more general case of a multivariate Gaussian, $\mathcal{N}(x|\mu, \Sigma)$, it turns out that it is also of an exponential form and its conjugate prior is of the Gaussian-Wishart form ([Problem 12.7](#)), that is,

$$p(\mu, Q; \mu_0, \lambda, W, v) = \mathcal{N}\left(\mu \middle| \mu_0, (\lambda Q)^{-1}\right) \mathcal{W}(Q|W, v),$$

where, $Q = \Sigma^{-1}$.

Example 12.4. We now turn our attention into discrete variables and we will show that the multinomial distribution is of an exponential form and that its conjugate prior is given by the Dirichlet distribution.

1. Let z_1, z_2, \dots, z_K be K mutually exclusive and exhaustive events. Let P_1, P_2, \dots, P_K be the respective probabilities, hence $\sum_{k=1}^K P_k = 1$. Let the experiment be repeated N times. Then, the probability of the joint event: z_1 occurred x_1 times, z_2 occurred x_2 times, and so on, is given by the multinomial distribution.

$$P(x_1, x_2, \dots, x_K) = \binom{N}{x_1 \dots x_K} \prod_{k=1}^K P_k^{x_k}, \quad (12.53)$$

where

$$\binom{N}{x_1 \dots x_K} = \frac{N!}{x_1! x_2! \dots x_K!}.$$

Defining $\mathbf{P} = [P_1, \dots, P_K]^T$, Eq. (12.53) can be rewritten as

$$\begin{aligned} P(x_1, \dots, x_K | \mathbf{P}) &= \binom{N}{x_1 \dots x_K} \prod_{k=1}^K \exp(x_k \ln P_k) \\ &= \binom{N}{x_1, \dots, x_K} \exp\left(\sum_{k=1}^K x_k \ln P_k\right). \end{aligned} \quad (12.54)$$

Thus, the multinomial is of an exponential form with

$$\begin{aligned} \boldsymbol{\phi}(\mathbf{P}) &= [\ln P_1, \ln P_2, \dots, \ln P_K]^T, \\ \mathbf{u}(\mathbf{x}) &= [x_1, x_2, \dots, x_K]^T, \end{aligned}$$

and because probabilities sum to one, we obtain

$$g(\mathbf{P}) = 1, f(\mathbf{x}) = \binom{N}{x_1 \dots x_K}.$$

2. The conjugate prior of (12.54) can then be written as

$$\begin{aligned} p(\mathbf{P}; \lambda, \mathbf{v}) &= h(\lambda, \mathbf{v}) \exp\left(\sum_{k=1}^K v_k \ln P_k\right) \\ &\propto \prod_{k=1}^K P_k^{v_k}, \end{aligned} \quad (12.55)$$

which is a Dirichlet pdf. If we let $v_k := a_k - 1$ we bring (12.55) in the more standard formulation

$$p(\mathbf{P}; \mathbf{a}) = \frac{\Gamma(\bar{a})}{\Gamma(a_1) \dots \Gamma(a_K)} \prod_{k=1}^K P_k^{a_k-1}, \quad \sum_{k=1}^K P_k = 1, \quad (12.56)$$

where the normalization constant (Chapter 2) has been plugged in, with

$$\bar{a} := \sum_{k=1}^K a_k.$$

12.4.1 THE EXPONENTIAL FAMILY AND THE MAXIMUM ENTROPY METHOD

Besides the computational advantages associated with the exponential family, there is another reason that justifies its high popularity. Assume that we are given a set of observations, $x_n \in \mathcal{A}_x \subseteq \mathbb{R}$, $n = 1, 2, \dots, N$, drawn from a distribution whose functional form is unknown. Our goal is to estimate the unknown pdf; however, we require that it will respect certain empirical expectations, which are computed from the available observations, that is,

$$\hat{\mu}_i := \frac{1}{N} \sum_{n=1}^N u_i(x_n), \quad i \in \mathcal{I}, \quad (12.57)$$

where \mathcal{I} is an index set and $u_i : \mathcal{A}_x \rightarrow \mathbb{R}$, $i \in \mathcal{I}$ are specific functions. For example, if $u_i(x) = x$, then $\hat{\mu}_i$ is the sample mean. In such cases, it is not sensible to adopt a parametric functional form for the pdf and try to optimize with respect to the unknown parameters, for example, via the maximum likelihood method; in general, we cannot know if an adopted functional form can comply with the available empirical expectations.

The maximum entropy (ME) method (sometimes called principle) offers a possible way to estimate the unknown pdf, subject to the set of the available constraints, [22]. According to this method, the cost function to be maximized is the *entropy* (Section 2.5.2) associated with the pdf, that is,

$$H := - \int_{\mathcal{A}_x} p(x) \ln p(x) dx. \quad (12.58)$$

It is well known from Shannon's information theory that the entropy is a measure of uncertainty or randomness. Maximizing the entropy, with respect to $p(x)$, results to the most random pdf, subject to the available constraints. Seen from another point of view, such a procedure guarantees that the estimation of an unknown pdf is carried out by adopting the least number of assumptions; that is, only the available set of constraints. For our case, the maximum entropy estimation method is cast as follows:

$$\begin{aligned} &\text{maximize with respect to } p(x) && - \int_{\mathcal{A}_x} p(x) \ln p(x) dx, \\ &\text{subject to} && \mathbb{E}[u_i(x)] = \int_{\mathcal{A}_x} p(x) u_i(x) dx, \quad i \in \mathcal{I}. \end{aligned} \quad (12.59)$$

In addition to the previous set of constraints, one has to consider the obvious one that guarantees that $p(x)$ integrates to one, that is,

$$\int_{\mathcal{A}_x} p(x) dx = 1.$$

In the case of discrete variables, the involved integrations are replaced by summations. Solving the optimization task in (12.59), it turns out that (Problem 12.9)

$$\hat{p}(x) = C \exp \left(\sum_{i \in \mathcal{I}} \theta_i u_i(x) \right), \quad (12.60)$$

that is, the ME estimate is of an exponential form. The parameters θ_i , $i \in \mathcal{I}$, are the Lagrange multipliers used in the optimization task and their values are determined via the constraints and are given in terms of the available empirical expectations, $\hat{\mu}_i$, $i \in \mathcal{I}$. If no constraint is used other than the obvious (normalizing) one and $\mathcal{A}_x = [a, b] \subset \mathbb{R}$, then the resulting pdf is the uniform distribution, $p(x) = C$; indeed, this is the most random one, because it shows no preference to any specific interval of values. If two constraints are used, such that $u_1(x) = x$ and $u_2(x) = x^2$, the resulting pdf is the Gaussian one, because the exponent is of a quadratic form (appendix in [Section 12.9](#)). In other words, the Gaussian is the most random pdf, subject to two constraints related to the mean and the variance. Note that although we focused on real-valued random variables, everything is trivially extended to vector-valued ones. An interesting discussion concerning maximum entropy method and alternative views of the problem is provided in [\[44\]](#).

12.5 LATENT VARIABLES AND THE EM ALGORITHM

At the end of [Section 12.3](#), it was pointed out that the evidence function associated with the regression task in Eq. [\(12.3\)](#), assuming that $p(y|\theta)$ and $p(\theta)$ are Gaussians of the form given in [\(12.39\)](#), is also Gaussian parameterized via a set of parameters, ξ , where for this case $\xi = [\sigma_\eta^2, \sigma_\theta^2]$, and we can write $p(y; \xi)$. Maximizing the evidence with respect to ξ becomes a typical ML task. However, in general, such closed-form expressions for the evidence function are not possible, and the integration in [\(12.14\)](#) is intractable. The main source of difficulty is the fact that our regression model is described via two random variables, that is, y and θ , yet only one of them, y , can be directly observed. The other one, θ , cannot be observed, and this is the reason that the Bayesian philosophy tries to integrate it out of the joint pdf $p(y, \theta)$. If θ could be observed, then the unknown set of parameters ξ could be obtained by maximizing the likelihood $p(y, \theta; \xi)$, given a set of (joint) observations (y, θ) . Because it cannot be observed, the random variable θ is known as *latent* or *hidden* variable.

Although we introduced the notion of latent variables via our familiar regression task, latent variables occur very often in a number of problems in probability and statistics. In a number of cases, from a larger set of jointly distributed random variables, only some can be observed and the rest remain hidden. Moreover, it is often useful to *build* hidden variables into a model by design. These variables are meant to represent latent causes that influence the observed variables and their introduction may facilitate the analysis.

12.5.1 THE EXPECTATION-MAXIMIZATION ALGORITHM

The *expectation-maximization* (EM) algorithm is an elegant algorithmic tool to maximize the likelihood function for problems with latent variables. We will state the problem in a general formulation, and then we will apply it to different tasks, including regression.

Let \mathbf{x} be a random vector and let \mathcal{X} be the respective set of observations. Let $\mathcal{X}^l := \{\mathbf{x}_1^l, \dots, \mathbf{x}_N^l\}$ be the corresponding set of latent variables; these can be either of a discrete or of a continuous nature. Each observation in \mathcal{X} is associated with a latent vector \mathbf{x}^l in \mathcal{X}^l . We will refer to the set $\{\mathcal{X}, \mathcal{X}^l\}$

as the *complete* data set and to the set of observations \mathcal{X} as the *incomplete* one. Let, also, their joint distribution be parameterized in terms of a set of unknown parameters, ξ .⁶ We further assume that, although \mathcal{X}^l cannot be observed, the posterior distribution $p(\mathcal{X}^l|\mathcal{X}; \xi)$ ($P(\mathcal{X}^l|\mathcal{X}; \xi)$ for the discrete case) is fully specified, given the values in ξ and the observations in \mathcal{X} . This is a critical assumption for the EM algorithm. If the posterior pdf is not known, then one has to resort to variants of the EM, which attempt to approximate it. We will come to such schemes in [Section 13.2](#).

If the complete log-likelihood $p(\mathcal{X}, \mathcal{X}^l; \xi)$ were available, then the problem would be a typical ML one. However, because no observations for the latent variables are available, the EM algorithm considers the *expectation* of the complete log-likelihood with respect to the latent variables associated with \mathcal{X}^l ; this operation is possible, because the posterior distribution $p(\mathcal{X}^l|\mathcal{X}; \xi)$ is known, provided that ξ is known. To this end, the EM algorithm builds on an iterative philosophy, initialized by an arbitrary value $\xi^{(0)}$. Then it proceeds along the following steps (see [Problem 12.11](#) for a justification).

The EM Algorithm

1. Expectation E-step: at the $(j + 1)$ iteration, compute $p(\mathcal{X}^l|\mathcal{X}, \xi^{(j)})$ and

$$\mathcal{Q}(\xi, \xi^{(j)}) = \mathbb{E} \left[\ln p(\mathcal{X}, \mathcal{X}^l; \xi) \right], \quad (12.61)$$

where the expectation is taken with respect to $p(\mathcal{X}^l|\mathcal{X}; \xi^{(j)})$.

2. Maximization M-step: Determine $\xi^{(j+1)}$ so that

$$\xi^{(j+1)} = \arg \max_{\xi} \mathcal{Q}(\xi, \xi^{(j)}). \quad (12.62)$$

3. Check for convergence according to a criterion. If it is not satisfied go to step 1.

A possible convergence criterion is to check whether $\|\xi^{(j+1)} - \xi^{(j)}\| < \epsilon$, for some user-defined constant ϵ . The use of the EM algorithm presupposes that working with the joint pdf $p(\mathcal{X}, \mathcal{X}^l; \xi)$ is computationally tractable. This is, for example, the case when working within the exponential family of pdfs, where the E-step may require only the computation of a few statistics of the latent variables.

Remarks 12.3.

- The EM algorithm was proposed and given its name in the seminal 1977 paper by Arthur Dempster, Nan Laird, and Donald Rubin [12]. The paper generalized previously published results, as for example, [2, 40] and had a significant impact as a powerful tool in statistics. The complete convergence proof was given in [49]. See, for example, [31] for a related discussion.
- It can be shown that the EM algorithm converges to a (in general, local) maximum of $p(\mathcal{X}; \xi)$, which was our original goal. The likelihood never decreases. The convergence is slower than the quadratic convergence of Newton-type searching techniques, although near an optimal point a speed up may be possible. However, the convergence of the algorithm is smooth and its complexity more attractive to Newton-type schemes, with no matrix inversions involved. The keen reader may obtain more information in, for example, [15, 32, 35, 45].

⁶ In the context of this chapter, we keep the notation θ for parameters treated as random variables and ξ for deterministic ones.

- The EM algorithm can be modified to obtain the MAP estimate. To this end, the M-step is changed to

$$\boldsymbol{\xi}^{(j+1)} = \arg \max_{\boldsymbol{\xi}} \left\{ \mathcal{Q}(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)}) + \ln p(\boldsymbol{\xi}) \right\},$$

where $p(\boldsymbol{\xi})$ is the prior pdf associated with $\boldsymbol{\xi}$, if it is considered to be a random vector.

- The EM algorithm can be sensitive to the choice of the initial point $\boldsymbol{\xi}^{(0)}$. In practice, one can run the algorithm a number of times, starting from different initial points and keep the best of the results. Other initialization procedures have also been used, depending on the application.
- Missing data:* The EM algorithm can also be used to cope with cases where some of the values from the observed training data are missing. Missing values can be treated as hidden variables and maximization of the likelihood can be done by marginalizing over them. Such a procedure makes sense only if data are *missing at random*; that is, the cause of missing data is a random event and does not depend on the values of the unobserved samples.

12.5.2 THE EM ALGORITHM: A LOWER BOUND MAXIMIZATION VIEW

Let us consider the functional⁷

$$\boxed{\mathcal{F}(q, \boldsymbol{\xi}) := \int q(\mathcal{X}^l) \ln \frac{p(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\xi})}{q(\mathcal{X}^l)} d\mathcal{X}^l,} \quad (12.63)$$

where $q(\mathcal{X}^l)$ is any nonnegative function that integrates to one; that is, it is a pdf defined over the latent variables. The functional $\mathcal{F}(\cdot, \cdot)$, depends on $\boldsymbol{\xi}$ and on $q(\cdot)$, and its definition bears a strong similarity with the notion of free energy, used in statistical physics. Indeed, (12.63) can be written as,

$$\mathcal{F}(q, \boldsymbol{\xi}) = \int q(\mathcal{X}^l) \ln p(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\xi}) d\mathcal{X}^l + H,$$

where

$$H = - \int q(\mathcal{X}^l) \ln q(\mathcal{X}^l) d\mathcal{X}^l,$$

is the entropy associated with $q(\mathcal{X}^l)$. If one defines $-\ln p(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\xi})$ as the *energy* of the system, $(\mathcal{X}, \mathcal{X}^l)$, then $\mathcal{F}(q, \boldsymbol{\xi})$ represents the negative of the so-called *free energy* [36].

Elaborating on (12.63), we get

$$\begin{aligned} \mathcal{F}(q, \boldsymbol{\xi}) &= \int q(\mathcal{X}^l) \ln \frac{p(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi}) p(\mathcal{X}; \boldsymbol{\xi})}{q(\mathcal{X}^l)} d\mathcal{X}^l, \\ &= \int q(\mathcal{X}^l) \ln \frac{p(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi})}{q(\mathcal{X}^l)} d\mathcal{X}^l + \ln p(\mathcal{X}; \boldsymbol{\xi}), \end{aligned} \quad (12.64)$$

where the latter results because $p(\mathcal{X}; \boldsymbol{\xi})$ does not depend on $q(\mathcal{X}^l)$. The first term on the right-hand side is the negative of the so-called *Kullback-Leibler divergence* between $q(\mathcal{X}^l)$ and $p(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi})$, which we will denote as $\text{KL}(q \parallel p)$. Thus, finally we get

⁷ A functional is an operator that takes as input a function and returns a real value. It is a generalization of our familiar functions, where now the inputs are also functions.

$$\ln p(\mathcal{X}; \xi) = \mathcal{F}(q, \xi) + \text{KL}(q \parallel p). \quad (12.65)$$

Because $\text{KL}(q \parallel p) \geq 0$ (Problem 12.12) it turns out that

$$\boxed{\ln p(\mathcal{X}; \xi) \geq \mathcal{F}(q, \xi)}. \quad (12.66)$$

$\mathcal{F}(q, \xi)$ is a lower bound of the log-likelihood function, and the bound becomes tight if $\text{KL}(q \parallel p) = 0$, which is true, *if and only if*, $q(\mathcal{X}^l) = p(\mathcal{X}^l | \mathcal{X}; \xi)$. The previous findings pave the way of maximizing $\ln p(\mathcal{X}; \xi)$ by trying to maximize its lower bound. Note that maximization of $\mathcal{F}(\cdot, \cdot)$ involves two terms, namely q , ξ . We will adopt a procedure that belongs to a more general class of optimization algorithms known as *alternating optimization*. Such an approach naturally imposes an iterative procedure that starts from an arbitrary $\xi^{(0)}$ and the $(j+1)$ iteration comprises the following steps:

- Step 1: Holding $\xi^{(j)}$ fixed, optimize with respect to q . This step tightens the lower bound in (12.66). This is achieved if $\text{KL}(q \parallel p) = 0$, and it can only happen if

$$q^{(j+1)}(\mathcal{X}^l) = p(\mathcal{X}^l | \mathcal{X}; \xi^{(j)}), \quad (12.67)$$

that is, if we set $q(\mathcal{X}^l)$ equal to the posterior given \mathcal{X} and $\xi^{(j)}$; as (12.65) suggests, this makes the bound tight, that is,

$$\ln p(\mathcal{X}; \xi^{(j)}) = \mathcal{F}\left(p(\mathcal{X}^l | \mathcal{X}; \xi^{(j)}), \xi^{(j)}\right). \quad (12.68)$$

- Step 2: Fixing $q^{(j+1)}$, insert it in the place of q in (12.66), and because the bound holds for any q , maximize with respect to ξ , that is,

$$\xi^{(j+1)} = \arg \max_{\xi} \mathcal{F}\left(p\left(\mathcal{X}^l | \mathcal{X}; \xi^{(j)}\right), \xi\right).$$

It is now readily seen that we have re-derived the EM algorithm. Indeed, from the definition of $\mathcal{F}(\cdot, \cdot)$ in (12.63) we obtain that

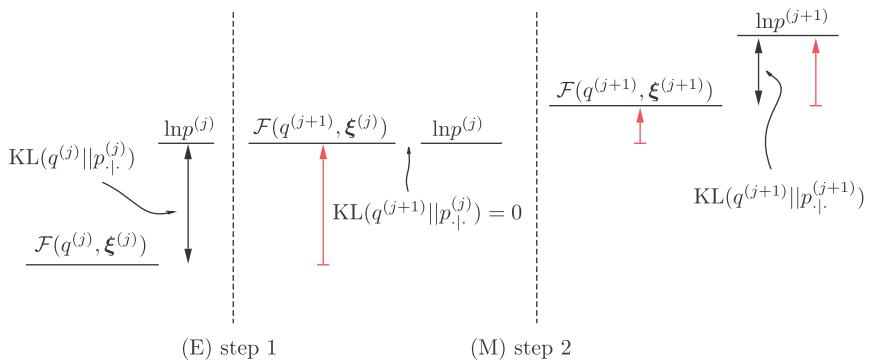
$$\mathcal{F}\left(p(\mathcal{X}^l | \mathcal{X}; \xi^{(j)}), \xi\right) = \mathcal{Q}(\xi, \xi^{(j)}) - \int p(\mathcal{X}^l | \mathcal{X}; \xi^{(j)}) \ln p(\mathcal{X}^l | \mathcal{X}; \xi^{(j)}) d\mathcal{X}^l, \quad (12.69)$$

where $\mathcal{Q}(\xi, \xi^{(j)})$ is the same as in (12.61) and the second term on the right-hand side is independent of ξ ; this latter term is equal to the entropy associated with $q^{(j+1)}(\mathcal{X}^l)$. The rederivation of the EM via this path makes it clear that the quantity that is maximized is the log-likelihood, $\ln p(\mathcal{X}; \xi)$, and that its value is guaranteed not to decrease after each combined iteration step. Figure 12.6 illustrates schematically the two EM steps comprising the $(j+1)$ th iteration.

Needless to say that the EM algorithm is not a panacea. We will soon seek variants to deal with cases where the posterior cannot be given in an analytic form. Moreover, there are still cases where the M-step can be computationally intractable. To this end, several variants have been proposed, see, for example, [33, 36].

Remarks 12.4.

- *Online versions of EM*: We have already pointed out that in many cases of large data applications, online versions are the preferable choice in practice. The EM algorithm is no exception, and a number of related versions have been proposed. In [36], an online EM algorithm is proposed based on the lower bound interpretation. In [6], stochastic approximation arguments have been employed. In [26], a comparative study of different techniques is reported.

**FIGURE 12.6**

The E-step adjusts $q^{(j)} := q^{(j)}(\mathcal{X}^l)$ so that its Kullback-Leibler (KL) distance from $p_{\cdot| \cdot}^{(j)} := p(\mathcal{X}^l | \mathcal{X}; \xi^{(j)})$ becomes zero. The M-step maximizes with respect to ξ .

- Often in practice, carrying out the expectation step may be intractable. Later on in the next chapter, we will see variational methods as a way to overcome this obstacle. An alternative path is to employ Monte Carlo sampling techniques ([Chapter 14](#)) to generate samples from the involved distributions and approximate the expectation with the computation of the respective sample mean see, for example, [[8, 11](#)].
- The EM algorithm can also be seen as a special case of a more general class of methods, which are based on optimizing a bound instead of the original cost. These techniques come under the name of *minorize-maximization, or majorize-minimization* (MM) methods; see, for example, [[21](#)].

12.6 LINEAR REGRESSION AND THE EM ALGORITHM

The Bayesian viewpoint to the regression task was considered in [Section 12.2.3](#) via the Gaussian model assumption for $p(\mathbf{y}|\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$, given in [\(12.9\)](#) and [\(12.8\)](#), which subsequently led to a Gaussian posterior for $p(\boldsymbol{\theta}|\mathbf{y})$, given in [\(12.16\)](#). In the current section, and for the sake of presentation simplicity, we will adopt the special case of diagonal covariance matrices, that is, $\Sigma_\eta = \sigma_\eta^2 I$, $\Sigma_\theta = \sigma_\theta^2 I$, and $\boldsymbol{\theta}_0 = \mathbf{0}$.

Our goal now becomes to consider σ_η^2 and σ_θ^2 as (non-random) parameters and to obtain their values by maximizing the corresponding evidence function in [\(12.15\)](#). To this end, we will use the EM algorithm. We will treat \mathbf{y} as the observed variables (corresponding to \mathcal{X}) and $\boldsymbol{\theta}$ as the set of latent variables (corresponding to \mathcal{X}^l). A prerequisite in order to apply the EM procedure is the knowledge of the posterior, which for this case is known, given the values of the parameters. As we have done many times in the book so far, we will work with the precision variables, and the parameter vector becomes

$$\xi = [\alpha, \beta]^T, \quad \alpha = \frac{1}{\sigma_\theta^2} \quad \text{and} \quad \beta = \frac{1}{\sigma_\eta^2}.$$

The EM algorithm is initialized with some arbitrary positive values $\alpha^{(0)}$ and $\beta^{(0)}$. Then the algorithm at the $(j+1)$ iteration step, where $\alpha^{(j)}$ and $\beta^{(j)}$ are assumed known, proceeds as follows:

- E-Step: Compute the posterior $p(\theta|y; \xi^{(j)})$, which according to (12.16) and for $\theta_0 = \mathbf{0}$ is fully specified if we compute its mean and covariance matrix, using (12.19, 12.20), that is,

$$\Sigma_{\theta|y}^{(j)} = (\alpha^{(j)} I + \beta^{(j)} \Phi^T \Phi)^{-1}, \quad (12.70)$$

$$\mu_{\theta|y}^{(j)} = \beta^{(j)} \Sigma_{\theta|y}^{(j)} \Phi^T y. \quad (12.71)$$

Compute the expected value of the log-likelihood associated with the complete data set; this is given by,

$$\ln p(y, \theta; \xi) := \ln p(y, \theta; \alpha, \beta) = \ln \left(p(y|\theta; \beta) p(\theta; \alpha) \right),$$

or

$$\begin{aligned} \ln p(y, \theta; \alpha, \beta) &= \frac{N}{2} \ln \beta + \frac{K}{2} \ln \alpha - \frac{\beta}{2} \|y - \Phi \theta\|^2 - \frac{\alpha}{2} \theta^T \theta \\ &\quad - \left(\frac{N}{2} + \frac{K}{2} \right) \ln(2\pi). \end{aligned} \quad (12.72)$$

Treating the latent parameters as random variables, the expected value of (12.72), with respect to Θ , is carried out via the Gaussian posterior defined by (12.70) and (12.71). To this end, the following steps are adopted.

1. To compute $\mathbb{E}[\theta^T \theta]$, recall the definition of the respective covariance matrix,

$$\Sigma_{\theta|y}^{(j)} = \mathbb{E} \left[(\theta - \mu_{\theta|y}^{(j)}) (\theta - \mu_{\theta|y}^{(j)})^T \right] \quad (12.73)$$

or

$$\mathbb{E}[\theta \theta^T] = \Sigma_{\theta|y}^{(j)} + \mu_{\theta|y}^{(j)} \mu_{\theta|y}^{(j)T}, \quad (12.74)$$

which results in

$$\begin{aligned} A &:= \mathbb{E}[\theta^T \theta] = \mathbb{E}[\text{trace}\{\theta \theta^T\}] \\ &= \text{trace}\{\mu_{\theta|y}^{(j)} \mu_{\theta|y}^{(j)T} + \Sigma_{\theta|y}^{(j)}\} \\ &= \|\mu_{\theta|y}^{(j)}\|^2 + \text{trace}\{\Sigma_{\theta|y}^{(j)}\}. \end{aligned} \quad (12.75)$$

2. To compute $\mathbb{E}[\|y - \Phi \theta\|^2]$, define $\psi := y - \Phi \theta$, and use the previous rationale to compute $\mathbb{E}[\psi^T \psi]$, which leads to (Problem 12.13)

$$B := \mathbb{E}[\|y - \Phi \theta\|^2] = \|y - \Phi \mu_{\theta|y}^{(j)}\|^2 + \text{trace}\{\Phi \Sigma_{\theta|y}^{(j)} \Phi^T\}. \quad (12.76)$$

Hence,

$$\mathcal{Q}(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}) = \frac{N}{2} \ln \beta + \frac{K}{2} \ln \alpha - \frac{\beta}{2} B - \frac{\alpha}{2} A - \left(\frac{N}{2} + \frac{K}{2} \right) \ln(2\pi). \quad (12.77)$$

- M-Step: Compute

$$\alpha^{(j+1)} : \frac{\partial}{\partial \alpha} \mathcal{Q}(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}) = 0,$$

$$\beta^{(j+1)} : \frac{\partial}{\partial \beta} \mathcal{Q}(\alpha, \beta; \alpha^{(j)}, \beta^{(j)}) = 0,$$

which trivially lead to

$$\alpha^{(j+1)} = \frac{K}{\|\boldsymbol{\mu}_{\theta|y}^{(j)}\|^2 + \text{trace}\{\Sigma_{\theta|y}^{(j)}\}}, \quad (12.78)$$

$$\beta^{(j+1)} = \frac{N}{\|y - \Phi\boldsymbol{\mu}_{\theta|y}^{(j)}\|^2 + \text{trace}\{\Phi\Sigma_{\theta|y}^{(j)}\Phi^T\}}. \quad (12.79)$$

Once the algorithm converges, the resulting values for α and β are used to completely specify the involved pdfs, which can be used either to obtain an estimate of $\hat{\theta}$, for example, $\hat{\theta} = \mathbb{E}[\Theta|y]$, or make predictions via (12.21).

Example 12.5. In this example, the generalized linear regression model of Example 12.1 is reconsidered. The goal is to use the EM algorithm of Section 12.6, as summarized by the recursions (12.70), (12.71), (12.78), and (12.79). The variance of the Gaussian noise used in the model to generate the data was set equal to $\sigma_\eta^2 = 0.05$. The number of training points was $N = 500$. For the EM algorithm, both α and β were initialized to one. The correct dimensionality for the unknown parameter vector was used. The recovered values after the convergence of the EM were $\alpha = 1.32$ corresponding to $\sigma_\theta^2 = 0.756$ and $\beta = 19.96$ corresponding to $\sigma_\eta^2 = 0.0501$. Note that the latter is very close to the true variance of the noise. Then, predictions of the output variable y were performed at 20 points, using (12.22) and the value of $\boldsymbol{\mu}_{\theta|y}$ recovered by the EM algorithm, via (12.71).

Figure 12.7a shows the predictions together with the associated error bars, computed from (12.23) using the values of σ_η^2 and σ_θ^2 obtained via the EM algorithm. Figure 12.7b shows the convergence curve for σ_η^2 as a function of the number of iterations of the EM algorithm.

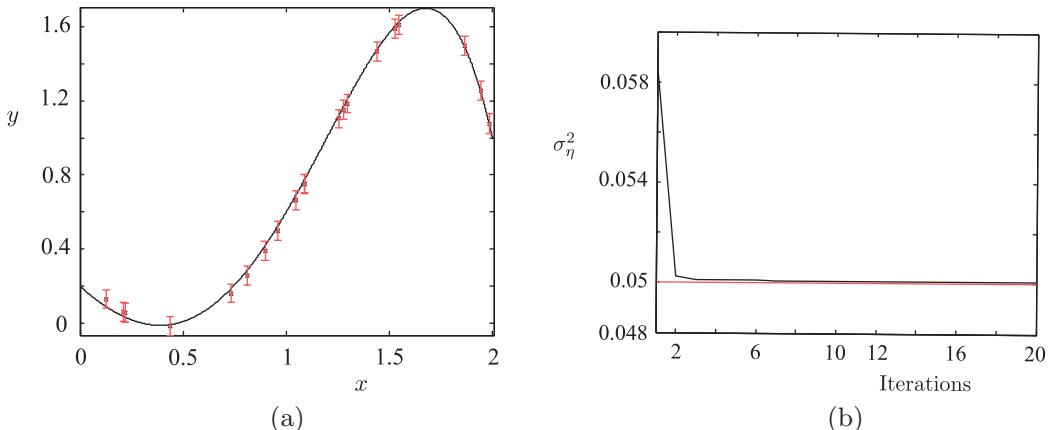


FIGURE 12.7

(a) The original graph from which the training points were sampled. In red, the respective predictions \hat{y} and associated error bars for twenty randomly chosen points are shown. (b) The convergence curve for σ_η^2 as a function of the iterations of the EM algorithm. The red line corresponds to the true value.

12.7 GAUSSIAN MIXTURE MODELS

So far, we have seen a number of pdfs that can be used to model the distribution of an unknown random vector $\mathbf{x} \in \mathbb{R}^l$. However, all these models restrict the pdf to a specific functional term. Mixture modeling provides the freedom to model the unknown pdf, $p(\mathbf{x})$, as a linear combination of different distributions, that is,

$$p(\mathbf{x}) = \sum_{k=1}^K P_k p(\mathbf{x}|k), \quad (12.80)$$

where P_k is the parameter weighting the specific contributing pdf, $p(\mathbf{x}|k)$. To guarantee that $p(\mathbf{x})$ is a pdf, the weighting parameters must be nonnegative and add to one ($\sum_{k=1}^K P_k = 1$). The physical interpretation of (12.80) is that we are given a set of K distributions, $p(\mathbf{x}|k)$, $k = 1, 2, \dots, K$. Each observation, \mathbf{x}_n , $n = 1, 2, \dots, N$, is drawn from one of these K distributions, but we are not told from which one. All we know is a set of parameters, P_k , $1, 2, \dots, K$, each one providing the probability that a sample has been drawn from the corresponding pdf, $p(\mathbf{x}|k)$. It can be shown that for a large enough number of *mixtures*, K , and appropriate choice of the involved parameters, one can approximate arbitrarily close any continuous pdf.

Mixture modeling is a typical task involving hidden variables; that is, the labels, k , of the pdf from which an obtained observation has originated. In practice, each $p(\mathbf{x}|k)$ is chosen from a known pdf family, parameterized via a set of parameters, and (12.80) can be rewritten as

$$p(\mathbf{x}) = \sum_{k=1}^K P_k p(\mathbf{x}|k; \boldsymbol{\xi}_k), \quad (12.81)$$

and the task is to estimate $(P_k, \boldsymbol{\xi}_k)$, $k = 1, 2, \dots, K$, based on a set of observations \mathbf{x}_n , $n = 1, 2, \dots, N$. The set of observations, $\mathcal{X} = \{\mathbf{x}_n, n = 1, \dots, N\}$, forms the incomplete set while the complete set $\{\mathcal{X}, \mathcal{K}\}$ comprises the samples (\mathbf{x}_n, k_n) , $n = 1, \dots, N$, with k_n being the label of the distribution (pdf) from which \mathbf{x}_n was drawn. Parameter estimation for such a problem naturally lends itself to be treated via the EM algorithm. We will demonstrate the procedure via the use of Gaussian mixtures.

Let

$$p(\mathbf{x}|k; \boldsymbol{\xi}_k) = p(\mathbf{x}|k; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),$$

where for simplicity we will assume that $\boldsymbol{\Sigma}_k = \sigma_k^2 I$, $k = 1, \dots, K$. We will further assume the observations to be i.i.d. For such a modeling, the following hold true:

- The log-likelihood of the *complete* data set is given by

$$\ln p(\mathcal{X}, \mathcal{K}; \boldsymbol{\Xi}, \mathbf{P}) = \sum_{n=1}^N \ln p(\mathbf{x}_n, k_n; \boldsymbol{\xi}_{k_n}) = \sum_{n=1}^N \ln \left(p(\mathbf{x}_n|k_n; \boldsymbol{\xi}_{k_n}) P_{k_n} \right). \quad (12.82)$$

We have used the notation,

$$\boldsymbol{\Xi} = [\boldsymbol{\xi}_1^T, \dots, \boldsymbol{\xi}_K^T]^T, \quad \mathbf{P} = [P_1, P_2, \dots, P_K]^T, \quad \text{and} \quad \boldsymbol{\xi}_k = [\boldsymbol{\mu}_k^T, \sigma_k^2]^T.$$

- The posterior probabilities of the hidden discrete variables are given by

$$P(k|\mathbf{x}; \boldsymbol{\Xi}, \mathbf{P}) = \frac{p(\mathbf{x}|k; \boldsymbol{\xi}_k) P_k}{p(\mathbf{x}; \boldsymbol{\Xi}, \mathbf{P})}, \quad (12.83)$$

where

$$p(\mathbf{x}; \boldsymbol{\Xi}, \mathbf{P}) = \sum_{k=1}^K P_k p(\mathbf{x}|k; \boldsymbol{\xi}_k). \quad (12.84)$$

We have now all the ingredients required by the EM algorithm. Starting from $\boldsymbol{\Xi}^{(0)}$ and $\mathbf{P}^{(0)}$, the $(j+1)$ iteration comprises the following steps:

- E-step: Using (12.83, 12.84) compute

$$P(k|\mathbf{x}_n; \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)}) = \frac{p(\mathbf{x}_n|k; \boldsymbol{\xi}_k^{(j)}) P_k^{(j)}}{\sum_{k=1}^K P_k^{(j)} p(\mathbf{x}_n|k; \boldsymbol{\xi}_k^{(j)})}, \quad n = 1, 2, \dots, N, \quad (12.85)$$

which in turn defines

$$\begin{aligned} \mathcal{Q}(\boldsymbol{\Xi}, \mathbf{P}; \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)}) &= \sum_{n=1}^N \mathbb{E} \left[\ln \left(p(\mathbf{x}_n|k_n; \boldsymbol{\xi}_{k_n}) P_{k_n} \right) \right] \\ &:= \sum_{n=1}^N \sum_{k=1}^K P(k|\mathbf{x}_n; \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)}) \left(\ln P_k - \frac{l}{2} \ln \sigma_k^2 \right. \\ &\quad \left. - \frac{1}{2\sigma_k^2} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2 \right) + C, \end{aligned} \quad (12.86)$$

where C includes all the terms corresponding to the normalization constant. Note that we have finally relaxed the notation from k_n to k , because we sum up over all k , which does not depend on n .

- M-step: Maximization of $\mathcal{Q}(\boldsymbol{\Xi}, \mathbf{P}; \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)})$ with respect to all the involved parameters results in the following set of recursions (Problem 12.14):

Set for notational convenience,

$$\gamma_{kn} := P(k|\mathbf{x}_n; \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)}).$$

Then,

$$\boldsymbol{\mu}_k^{(j+1)} = \frac{\sum_{n=1}^N \gamma_{kn} \mathbf{x}_n}{\sum_{n=1}^N \gamma_{kn}}, \quad (12.87)$$

$$\sigma_k^{2(j+1)} = \frac{\sum_{n=1}^N \gamma_{kn} \|\mathbf{x}_n - \boldsymbol{\mu}_k^{(j+1)}\|^2}{l \sum_{n=1}^N \gamma_{kn}}, \quad (12.88)$$

$$P_k^{(j+1)} = \frac{1}{N} \sum_{n=1}^N \gamma_{kn}. \quad (12.89)$$

Iterations continue until a convergence criterion is met. The extension to the case of a general covariance matrix is straightforward by replacing (12.88) by

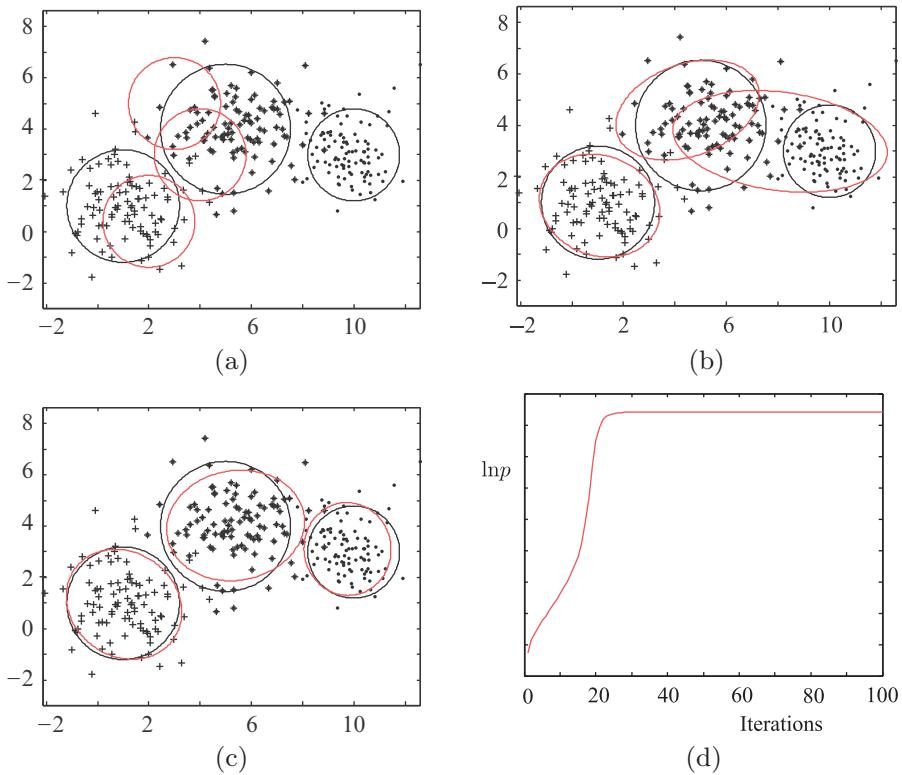
$$\boldsymbol{\Sigma}_k^{(j+1)} = \frac{\sum_{n=1}^N \gamma_{kn} (\mathbf{x}_n - \boldsymbol{\mu}_k^{(j+1)}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{(j+1)})^\top}{\sum_{n=1}^N \gamma_{kn}}.$$

Remarks 12.5.

- To get good initialization for the EM algorithm, sometimes a simpler clustering algorithm, for example, the k -means (Section 12.7.1, [43]), is run to provide an initial estimate of the means and shapes of clusters (covariance matrices), by associating each mixture with a cluster in the input space. Another simpler way is to select K points randomly from the data set. A more elaborate technique, which is commonly used, is to select them randomly but in such a way to make sure that the whole data set is represented in a balanced way, see, for example, [1].
- The number of mixtures, K , is usually determined by cross-validation (Chapter 3); see, also, [17].
- The mixing parameters, P_k , $k = 1, \dots, K$, should be initialized by keeping in mind that they are probabilities and they have to add to one.
- One of the problems, that may be encountered in practice in the Gaussian mixture task, is when one of the mixture components is centered at (or very close to) one of the data points, for example, $\mu_k^{(j+1)} = \mathbf{x}_n$, for some values of k and n . In such a case, the exponent term of the respective Gaussian becomes one and the contribution of this particular component in the log-likelihood is equal to $(2\pi\sigma_k^2)^{-l/2}$. If, in addition, σ_k is very small, this will lead the likelihood to a large value, although this is not indicative that the true model has been learned. Soon, we will see that the use of priors can alleviate such problems.
- *Identifiability:* A further issue associated with the EM algorithm, in the context of distribution mixtures, is that the obtained solution in the parameter space is not unique. For the case of K mixtures, for each solution (point in the parameter space) there are $K! - 1$ other points which give rise to the same distribution. For example, let us fit a model of two Gaussians in the one-dimensional space, which will result in estimates for the respective mean values, $\hat{\mu}_1$ and $\hat{\mu}_2$. However, in the corresponding parameter space, there is an uncertainty on whether these values define the point $\boldsymbol{\mu}_a = [\hat{\mu}_1, \hat{\mu}_2]^T$ or the point $\boldsymbol{\mu}_b = [\hat{\mu}_2, \hat{\mu}_1]$. Both of these points give rise to the same distribution. We say that the parameters in our model are not identifiable. A parameter (vector), which defines a family of distributions, $p(\mathbf{x}; \boldsymbol{\theta})$, is said to be *identifiable* if $p(\mathbf{x}; \boldsymbol{\theta}_1) \neq p(\mathbf{x}; \boldsymbol{\theta}_2)$ for $\boldsymbol{\theta}_1 \neq \boldsymbol{\theta}_2$, see, e.g., [7]. Although in our context, where our interest is in computing $p(\mathbf{x})$, unidentifiability does not cause any problems, this can be an issue in cases where the focus of interest lies on the parameters, see, for example, [38].
- *Mixtures of Student's-t distributions:* A significant shortcoming of mixtures based on normal distributions is their vulnerability to outliers. Recently, the replacement of normal distributions with the heavier-tailed Student's-t distributions (see Section 13.5) has been proposed as a way to mitigate these shortcomings and a related treatment of the resulting model under an expectation-maximization algorithmic framework has been conducted. Although the steps get a bit more involved, the ideas explored so far transfer nicely in this case too, see, for example, [9, 10, 37, 39].

Example 12.6. The goal of this example is to demonstrate the application of the EM algorithm in the context of the Gaussian mixture modeling. The data are generated according to three Gaussians in the two-dimensional space, with parameters,

$$\boldsymbol{\mu}_1 = [10, 3]^T, \quad \boldsymbol{\mu}_2 = [1, 1]^T, \quad \boldsymbol{\mu}_3 = [5, 4]^T,$$

**FIGURE 12.8**

The curves (ellipses) indicate the 80% probability regions. The gray curves correspond to the true Gaussian clusters, of [Example 12.6](#). The red curves correspond to (a) the initial values for the mean, covariance matrices and probabilities, (b) to the recovered by the EM algorithm mixtures after five iterations, and (c) after 30 iterations. (d) The log-likelihood as a function of the number of iterations.

and covariance matrices,

$$\Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}, \quad \Sigma_3 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},$$

respectively. The number of the generated points is 300 with 100 points per mixture. The points are shown in [Figure 12.8](#), together with the gray circles indicating the 80% probability regions, for each one of the clusters. The EM algorithm, comprising the steps [\(12.85\)](#) and [\(12.87\)-\(12.89\)](#) was run, with the following initial values,

$$\mu_1^{(0)} = [3, 5]^T, \quad \mu_2^{(0)} = [2, 0.4]^T, \quad \mu_3^{(0)} = [4, 3]^T,$$

and

$$\Sigma_1^{(0)} = \Sigma_2^{(0)} = \Sigma_3^{(0)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

The probabilities were initialized to their true values $P_1^{(0)} = P_2^{(0)} = P_3^{(0)} = 1/3$. The red curves in Figure 12.8 correspond to the mixtures recovered by the EM algorithm at (a) the initial estimates (12.8a), (b) after five iterations (12.8b) and (c) after convergence, (12.8c). Figure 12.8d shows the log-likelihood as a function of the number of iterations.

Figure 12.9 corresponds to a different setup. This time, the mean values were initialized at points very far from the true ones, that is,

$$\mu_1^{(0)} = [10, 13]^T, \quad \mu_2^{(0)} = [11, 12]^T, \quad \mu_3^{(0)} = [13, 11]^T,$$

while the covariances and probabilities were initialized as before. Observe that in this case, the EM algorithm fails to capture the true nature of the problem, having been trapped in a local minimum.

12.7.1 GAUSSIAN MIXTURE MODELING AND CLUSTERING

Clustering or unsupervised learning is an important part of machine learning, which is not treated in this book. Extensive coverage of clustering is given in, for example, [43]. However, the mixture modeling

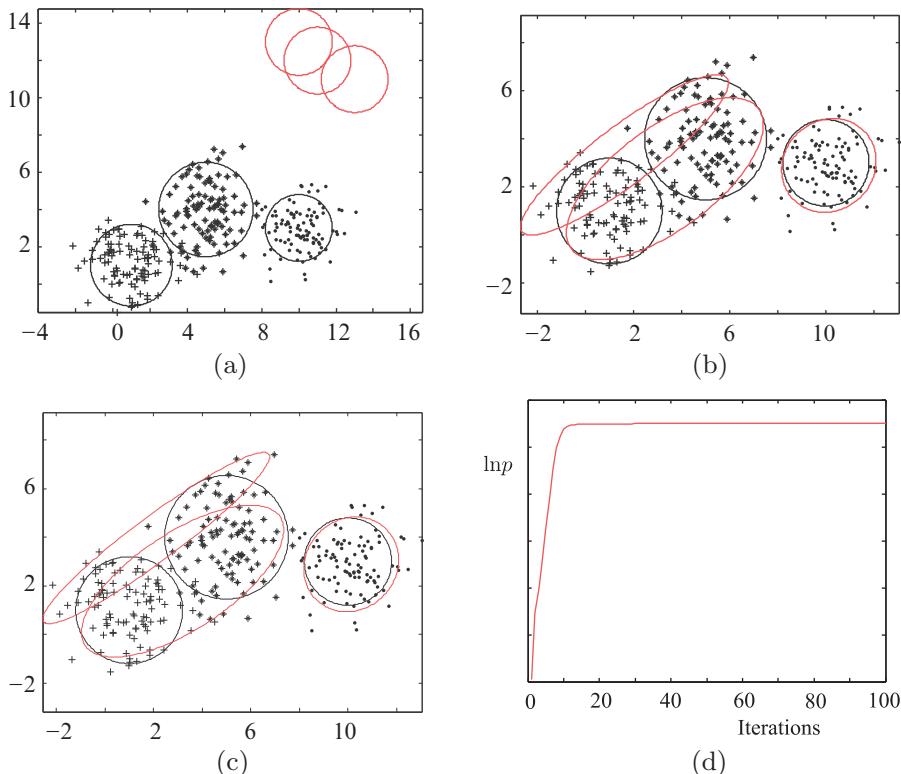


FIGURE 12.9

This is the counterpart of Figure 12.8, where now the initial values for the means are very far from the true ones. In this case, the EM fails to recover the true nature of the mixtures and has been trapped in a local minimum.

task via the EM offers us a good excuse to say a few words. Without going into formal definitions, the task of clustering is to assign a number of points, $\mathbf{x}_1, \dots, \mathbf{x}_N$, into K groups or clusters. Points that are assigned to the same cluster must be more “similar” than points which are assigned to different clusters. Some clustering algorithms need the number of clusters, K , to be provided by the user as an input variable. Other schemes treat it as a free parameter to be recovered from the data by the algorithm. The other major issue in clustering is to quantify “similarity.” Different definitions end up with different clusterings. A clustering is a specific allocation of the points to clusters. In general, assigning points to clusters according to an optimality criterion is an NP-hard task, see, for example, [43]. Thus, in general, any clustering algorithm provides a suboptimal solution.

Gaussian mixture modeling is among the popular clustering algorithms. The main assumption is that the points, which belong to the same cluster, are distributed according to the same Gaussian distribution (this is how similarity is defined in this case), of unknown mean and covariance matrix. Each mixture component defines a different cluster. Thus, the goal is to run the EM algorithm over the available data points to provide, after convergence, the posterior probabilities $P(k|\mathbf{x}_n)$, $k = 1, 2, \dots, K$, $n = 1, 2, \dots, N$, where each k corresponds to a cluster. Then, each point is assigned to cluster k according to the rule,

$$\text{assign } \mathbf{x}_n \text{ to cluster } k = \arg \min_i P(i|\mathbf{x}_n), \quad i = 1, 2, \dots, K.$$

The EM algorithm for clustering can be considered to be a refined version of a more primitive scheme, known as the *k-means* or *isodata* algorithm. In the EM algorithm, the posterior probability of each point, \mathbf{x}_n , with respect to each one of the clusters, k , is computed recursively. Moreover, the mean value μ_k , of the points associated with cluster k , is computed as a weighted average of *all* the training points (12.87). In contrast, in the *k-means* algorithm, at each iteration the posterior probability, gets a binary value in {1, 0}; for each point, \mathbf{x}_n , the Euclidean distance from all the currently available estimates of the mean values is computed, and the posterior probability is estimated according to the following rule,

$$P(k|\mathbf{x}_n) = \begin{cases} 1, & \text{if } \|\mathbf{x}_n - \mu_k\|^2 < \|\mathbf{x}_n - \mu_j\|^2, j \neq k, \\ 0, & \text{otherwise.} \end{cases}$$

The *k-means* algorithm is not concerned about covariance matrices. Despite its simplicity, it is not an exaggeration to say that it is the most well-known clustering algorithm, and a number of theoretical papers and improved versions have been proposed over the years, see, for example, [43]. Due to its popularity, we will take the liberty to state it in [Algorithm 12.1](#).

Algorithm 12.1 (The *k-means* or *isodata* clustering algorithm).

- Initialize
 - Select the number of clusters K .
 - Set μ_k , $k = 1, 2, \dots, K$, to arbitrary values.
- **For** $n = 1, 2, \dots, N$, **Do**
 - Determine the closest cluster mean, say, μ_k , to \mathbf{x}_n .
 - Set $b(n) = k$.
- **End For**
- **For** $k = 1, 2, \dots, K$, **Do**
 - Update μ_k , $k = 1, 2, \dots, K$, as the mean of all the points with $b(n) = k$, $n = 1, 2, \dots, N$.

- **End For**
- Until no change in μ_k , $k = 1, 2, \dots, K$, occurs between two successive iterations.

The k -means algorithm can also be derived as a limiting case of the EM-scheme, for example, [36]. Note that both the EM algorithm as well as the k -means one can only recover *compact clusters*. In other words, if the points are distributed in ring-shaped clusters, then this type of clustering algorithms is not appropriate.

[Figure 12.10a](#) shows the data points generated by two Gaussians; 200 points from each one. The points are shown by red and gray colors, depending on the Gaussian that generated them. Of course in clustering, the data points are given to the algorithm without the “color” (labeling). It is up to the algorithm to make the partition in clusters. For both, the EM and the k -means algorithms, the correct number of clusters ($K = 2$) was given. The k -means was initialized with zero mean values. [Figure 12.10b](#) shows the clusters formed by the k -means and in [12.10d](#) the clusters formed

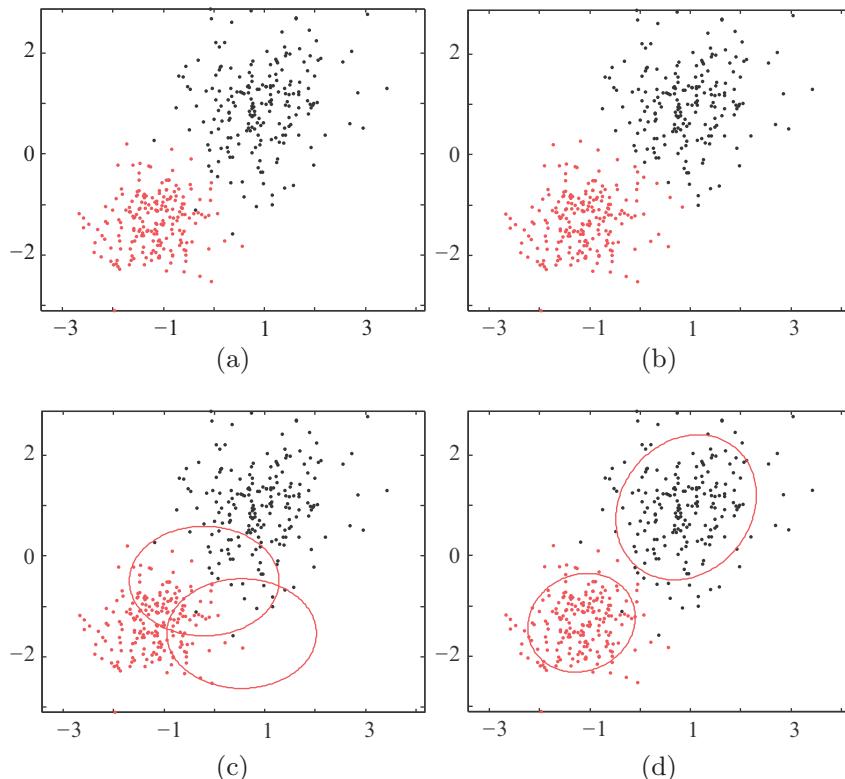


FIGURE 12.10

(a) The data points generated by two Gaussians (red and gray), (b) the recovered clusters by the k -means (red and gray), (c) The 80% probability curves for the initialization of the the EM algorithm, and (d) the final obtained by the EM algorithm Gaussians with the respective clusters.

by the EM algorithm. Figure 12.10c shows the Gaussians that were used for the initialization of the EM.

Figure 12.11 shows the respective sequence of figures, which corresponds to points obtained by the same Gaussians; however, now, there is an imbalance to the number of the points, where only 20 points spring from the first one and 200 points from the second. Observe that the k -means has a problem in recovering the true clustering structure; it attempts to make the two clusters more equally sized. A number of techniques and versions of the basic k -means scheme have been proposed to overcome its drawbacks; see, [43]. Finally, it must be stressed that both, the EM and the k -means algorithms, will always recover as many clusters as the user-defined input variable, K , dictates. In the case of the EM algorithm, this drawback is overcome when the variational EM algorithm is used, as will be discussed in Section 13.4.

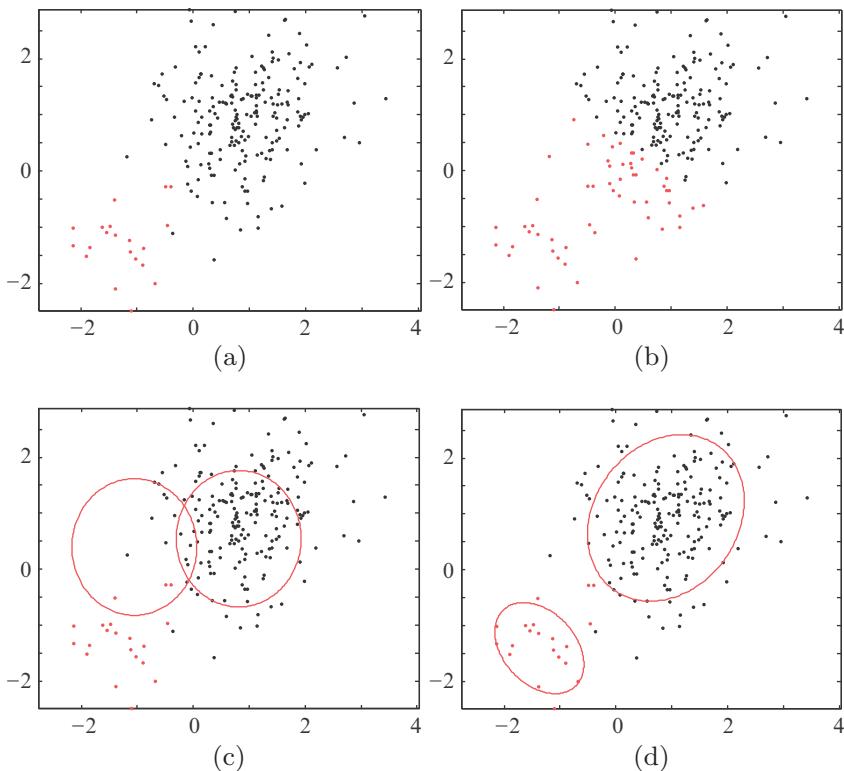


FIGURE 12.11

(a) The data points generated by two Gaussians (red and gray). One of the clusters consists of only 20 points and the other one of 200 points. (b) The recovered clusters by the k -means (red and gray). Observe that the algorithm has not identified the correct clusters, by assigning more points to the “smaller” one. (c) The 80% probability curves for the initialization of the the EM algorithm, and (d) the final Gaussians, obtained by the EM algorithm, with the respective clusters.

12.8 COMBINING LEARNING MODELS: A PROBABILISTIC POINT OF VIEW

The idea of combining different learners to boost the overall performance, by exploiting their individual characteristics was introduced in [Section 7.9](#). We now return to this task via probabilistic arguments. Our starting consideration is that the data are distributed in different regions of the input space. Thus, it seems reasonable to fit different learning models, one for each region. This idea reminds us of the decision trees treated in [Chapter 7](#). There, axis-aligned (linear) splits of the input space were performed. Here, the input space will be split via hyperplanes (generalizations to more general hypersurfaces are also possible) in a general position. Moreover, the main difference lies in the fact that in CARTS, the splits were of the hard-type decision rule. In the current setting, we adopt a more relaxed attitude and we are going to consider soft-type probabilistic splits, at the expense of some loss in interpretability.

The basic concept of the combining scheme of this section is illustrated in [Figure 12.12](#). It is common to refer to each one of the K learners as an *expert*. At the heart of our modeling approach lie the so-called *gating* functions, $g_k(\mathbf{x})$, $k = 1, 2, \dots, K$, which control the importance of each expert towards the final decision. These are optimally tuned during the training phase, together with the set of parameters, θ_k , $k = 1, 2, \dots, K$, which parameterize the experts, respectively. In the general case, the gating functions are functions of the input variables. We refer to this type of modeling as *mixture of experts*. In contrast, the special type of combination, where these are parameters and not functions, that is, $g_k(\mathbf{x}) = g_k$, will be referred to as *mixing of learners*. We will focus on the latter case and present the method in the context of the regression and classification tasks, using linear models.

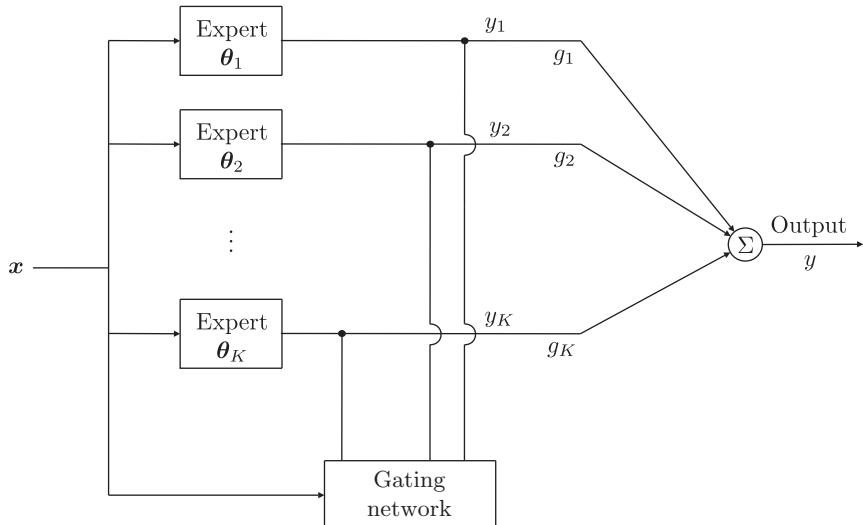


FIGURE 12.12

A block diagram of a mixture of experts. The output of each expert is weighted according to the outputs of the gating network. In the general case, these weights are considered functions of the input.

12.8.1 MIXING LINEAR REGRESSION MODELS

Our starting point is that each model is a linear regression model, $\boldsymbol{\theta}_k$, $k = 1, 2, \dots, K$, where the dimensionality of the input space has been assumed to increase by one, to account for the intercepts and the output variables are related to the input according to our familiar equation,

$$\mathbf{y}_k = \boldsymbol{\theta}_k^T \mathbf{x} + \eta, \quad (12.90)$$

where, η is a Gaussian noise source, with variance σ_η^2 and it is assumed to be common for all models; extension to more general cases can be obtained in a straightforward way. For generalized linear models, \mathbf{x} can simply be replaced by the nonlinear mapping, $\phi(\mathbf{x})$. We assume that the gating parameters are interpreted as probabilities, and they will be denoted as $g_k = P_k$.

Under the previous assumptions, the following mixing model is adopted,

$$p(\mathbf{y}; \boldsymbol{\Xi}, \mathbf{P}) = \sum_{k=1}^K P_k \mathcal{N}(\mathbf{y} | \boldsymbol{\theta}_k^T \mathbf{x}, \sigma_\eta^2), \quad (12.91)$$

where

$$\boldsymbol{\Xi} := [\boldsymbol{\theta}_1^T, \dots, \boldsymbol{\theta}_K^T, \sigma_\eta^2]^T, \quad \mathbf{P} := [P_1, \dots, P_K]^T, \quad (12.92)$$

are the vectors of the unknown parameters, to be estimated during the training phase using the set of training points, $(\mathbf{y}_n, \mathbf{x}_n)$, $n = 1, 2, \dots, N$. Because each model is designed to be “in charge” of one region in space, the corresponding parameters should be trained using input samples that originate from the respective region; note, however, that the regions are not known and have to be learned during training, as well. This is in analogy with the task of Gaussian mixture modeling; recall that during training, each observation was associated with a specific mixture component via the use of a hidden variable. In the current setting, each input sample will be associated with a specific learner. Thus, our current task is a close relative of the one treated in [Section 12.7](#) and we could follow similar steps to derive our results. However, for the sake of variety, a slightly different route will be taken. This will also prove useful later on and at the same time fits slightly better with the jargon used for the current formulation. Instead of the indices, k_n , used in Gaussian mixture modeling, we will introduce a new set of *hidden* variables, $z_{nk} \in \{0, 1\}$, $k = 1, 2, \dots, K$, $n = 1, 2, \dots, N$. If $z_{nk} = 1$, then sample \mathbf{x}_n is processed by expert k . At the same time, for each n , z_{nk} becomes equal to one only for a single value of k and zero for the rest. We are now ready to write down the likelihood for the *complete* training data set⁸, (\mathbf{y}_n, z_{nk}) , that is,

$$p(\mathbf{y}, Z; \boldsymbol{\Xi}, \mathbf{P}) = \prod_{n=1}^N \prod_{k=1}^K \left(P_k \mathcal{N} \left(\mathbf{y}_n | \boldsymbol{\theta}_k^T \mathbf{x}_n, \sigma_\eta^2 \right) \right)^{z_{nk}}, \quad (12.93)$$

where \mathbf{y} is the vector of the output observations and Z the matrix of the respective hidden variables. The log-likelihood is readily obtained as,

$$\ln p(\mathbf{y}, Z; \boldsymbol{\Xi}, \mathbf{P}) = \sum_{n=1}^N \sum_{k=1}^K z_{nk} \ln \left(P_k \mathcal{N} \left(\mathbf{y}_n | \boldsymbol{\theta}_k^T \mathbf{x}_n, \sigma_\eta^2 \right) \right). \quad (12.94)$$

⁸ Strictly speaking, the data set depends also on \mathbf{x}_n ; to simplify notation, we only give \mathbf{y}_n , because this is the one that is treated as a random variable.

We can now state the steps for the EM algorithm. Starting from some initial conditions, $\Xi^{(0)}$, $\mathbf{P}^{(0)}$, the $j+1$ iteration is given by:

- E-Step:

$$\begin{aligned}\mathcal{Q}(\Xi, \mathbf{P}; \Xi^{(j)}, \mathbf{P}^{(j)}) &= \mathbb{E}_Z [\ln p(\mathbf{y}, \mathbf{Z}; \Xi, \mathbf{P})] \\ &= \sum_{n=1}^N \sum_{k=1}^K \mathbb{E}[z_{nk}] \ln \left(P_k \mathcal{N}(y_n | \boldsymbol{\theta}_k^T \mathbf{x}_n, \sigma_\eta^2) \right).\end{aligned}$$

However,

$$\begin{aligned}\mathbb{E}[z_{nk}] &= P(k|y_n; \Xi^{(j)}, \mathbf{P}^{(j)}), \\ &= \frac{P_k^{(j)} \mathcal{N}(y_n | \boldsymbol{\theta}_k^{(j)T} \mathbf{x}_n, \sigma_\eta^2)}{\sum_{i=1}^K P_i^{(j)} \mathcal{N}(y_n | \boldsymbol{\theta}_i^{(j)T} \mathbf{x}_n, \sigma_\eta^2)},\end{aligned}\tag{12.95}$$

or

$$\begin{aligned}\mathcal{Q}(\Xi, \mathbf{P}; \Xi^{(j)}, \mathbf{P}^{(j)}) &= \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \left(\ln P_k - \frac{1}{2} \ln \sigma_\eta^2 - \frac{1}{2\sigma_\eta^2} (y_n - \boldsymbol{\theta}_k^T \mathbf{x}_n)^2 \right) + C,\end{aligned}\tag{12.96}$$

where C is a constant not affecting the optimization and

$$\gamma_{nk} := P(k|y_n; \Xi^{(j)}, \mathbf{P}^{(j)}).$$

As expected, (12.96) looks like (12.86).

- M-Step: The step comprises the computation of the unknown parameters via three different optimization problems.

Gating parameters: Following exactly similar steps as for (12.89), we obtain

$$P_k^{(j+1)} = \frac{1}{N} \sum_{n=1}^N \gamma_{nk}.$$

(12.97)

Learners' parameters: For each $k = 1, 2, \dots, K$, we have

$$\mathcal{Q}(\Xi, \mathbf{P}; \Xi^{(j)}, \mathbf{P}^{(j)}) = - \sum_{n=1}^N \frac{\gamma_{nk}}{2\sigma_\eta^2} (y_n - \boldsymbol{\theta}_k^T \mathbf{x}_n)^2 + C_1,\tag{12.98}$$

where C_1 includes all terms that do not depend on $\boldsymbol{\theta}_k$. Taking the gradient and equating to zero we readily obtain,

$$\sum_{n=1}^N \gamma_{nk} \mathbf{x}_n (y_n - \mathbf{x}_n^T \boldsymbol{\theta}_k) = \mathbf{0},$$

or, employing the input data matrix, $X^T := [\mathbf{x}_1, \dots, \mathbf{x}_N]$,

$$X^T \Gamma_k (\mathbf{y} - X \boldsymbol{\theta}_k) = \mathbf{0},$$

with

$$\Gamma_k := \text{diag}\{\gamma_{1k}, \dots, \gamma_{Nk}\},$$

and finally

$$\boldsymbol{\theta}_k^{(j+1)} = (X^T \Gamma_k X)^{-1} X^T \Gamma_k \mathbf{y}, \quad k = 1, 2, \dots, K. \quad (12.99)$$

Equation (12.99) is the solution to a weighted LS problem, similar in form as the one met in [Section 7.6](#), while dealing with the logistic regression. Note that the weighting matrix involves the posterior probabilities associated with the k th expert.

Noise variance: We have that,

$$\begin{aligned} \mathcal{Q}(\boldsymbol{\Xi}, \mathbf{P}; \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)}) &= \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \left(-\frac{1}{2} \ln \sigma_\eta^2 - \right. \\ &\quad \left. \frac{1}{2\sigma_\eta^2} (y_n - \boldsymbol{\theta}_k^{T(j+1)} \mathbf{x}_n)^2 \right) + C_2, \end{aligned} \quad (12.100)$$

whose optimization with respect to σ_η^2 , leads to

$$(\sigma_\eta^{(j+1)})^2 = \frac{1}{N} \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} (y_n - \boldsymbol{\theta}_k^{T(j+1)} \mathbf{x}_n)^2. \quad (12.101)$$

Mixture of experts

In mixture of experts, [25], the gating parameters are expressed in a parametric form, as functions of the input variables, \mathbf{x} . A common choice is to assume that

$$g_k(\mathbf{x}) := P_k(\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T \mathbf{x})}{\sum_{i=1}^K \exp(\mathbf{w}_i^T \mathbf{x})}. \quad (12.102)$$

Referring to [Figure 12.12](#), the gating weights are the outputs of the gating network, which is also excited by the same inputs as the experts. In the neural networks context, as we will see in [Chapter 18](#), we can consider the gating network as a neural network, with activation function given by (12.102), which is known as the *softmax* activation, [5]. Note that (12.102) is of exactly the same form as (7.40), used in the multiclass logistic regression. Under such a setting, P_k in (12.96) is replaced in $P_k(\mathbf{x})$, and the respective M-step becomes equivalent with optimizing with respect to \mathbf{w}_k , $k = 1, 2, \dots, K$, the following

$$\mathcal{Q}(\boldsymbol{\Xi}, \mathbf{P}; \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)}) = \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \ln P_k(\mathbf{x}) + C_3. \quad (12.103)$$

Observe that (12.103) is of the same form as (7.42), used for the multiclass logistic regression, and optimization follows similar steps; see also, for example, [20].

A mixture of experts has been used in a number of applications with a typical one being that of inverse problems, where from the output, one has to deduce the input. However, in many cases, this

is a one-to-many task and the mixture of experts is useful to model the choice among these “many” options. For example, in [4], mixture of experts are used for tracking people in video recordings, where the mapping from the image to pose is not unique, due to occlusion.

Hierarchical mixture of experts

A direct generalization of the mixture of experts concept is to add more levels of gating functions in a hierarchical fashion, giving rise to what is known as a *hierarchical mixture of experts* (HME). The idea is illustrated in the block diagram of [Figure 12.13](#). This architecture resembles that of trees, having the experts as leaves, the gating networks as nonterminal nodes, and the output (summing) node as the root one. A hierarchical mixture of experts divides the space into a nested set of regions, with the information combined among the experts under the control of the hierarchically placed gating networks. This hierarchy conforms with the more general idea of *conquer and divide* strategies.

Compared to decision trees, an HME evolves around soft decision rules, in contrast to the hard ones that are employed in CARTs. A hard decision, usually, leads to a loss of information. Once a decision is taken, it cannot change later on. In contrast, soft decision rules provide the luxury to the network to preserve information until a final decision is taken. For example, according to a hard decision rule, if a sample is located close to a decision surface, it will be labeled according to the label on which side it lies. However, in a soft decision rule, the information, related to the position of the point with respect to the decision surface, will be retained until the stage at which the final decision must be made, by taking into consideration more information that becomes available as the processing develops.

Note that training a mixture of experts can also be carried via a different path, by optimizing a cost function without it being necessary to employ probabilistic arguments; see, for example, [20].

12.8.2 MIXING LOGISTIC REGRESSION MODELS

Following [Section 12.8.1](#), the combination rationale can also be applied to classification tasks. To this end, we employ the two-class logistic regression model for each one of the experts and the combination rule, given the input value \mathbf{x} , is now written as,

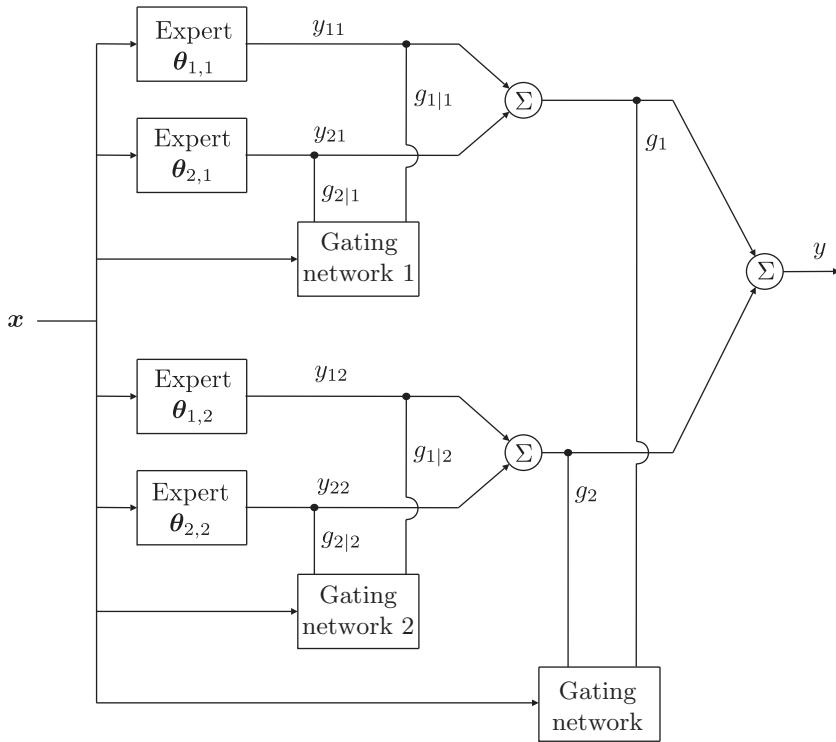
$$P(y; \Xi, \mathbf{P}) = \sum_{k=1}^K P_k s_k^y (1 - s_k)^{1-y}, \quad (12.104)$$

where the definition of logistic regression from [Section 7.6](#) has been used, with the labels $y \in \{0, 1\}$ corresponding to the two classes, ω_1 and ω_2 respectively, and

$$s_k := \sigma(\boldsymbol{\theta}_k^T \mathbf{x}), \quad (12.105)$$

denotes the output of the k th expert. As in [Section 12.8.1](#), Ξ is the set of the unknown parameters and \mathbf{P} the corresponding set of the gating network. Mobilizing similar arguments as for the case of linear regression, we can easily state that the likelihood of the complete data set is given by

$$P(\mathbf{y}, \mathbf{Z}; \Xi, \mathbf{P}) = \prod_{n=1}^N \prod_{k=1}^K \left(P_k s_{nk}^{y_n} (1 - s_{nk})^{1-y_n} \right)^{z_{nk}}, \quad (12.106)$$

**FIGURE 12.13**

A block diagram of a hierarchical mixture of experts with two levels of hierarchy.

where $s_{nk} := \sigma(\boldsymbol{\theta}_k^T \mathbf{x}_n)$ and \mathbf{y} the set of labels, y_n , $n = 1, 2, \dots, N$, of the training samples. Following the standard arguments of the EM algorithm applied on the respective log-likelihood function, it is readily shown that the E-step at the j th iteration is given by,

$$Q(\boldsymbol{\Xi}, \mathbf{P}; \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)}) = \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \left(\ln P_k + y_n \ln s_{nk} + (1 - y_n) \ln(1 - s_{nk}) \right), \quad (12.107)$$

where,

$$\gamma_{nk} = \mathbb{E}[z_{nk}] = P(k|y_n, \boldsymbol{\Xi}^{(j)}, \mathbf{P}^{(j)}) = \frac{P_k^{(j)} s_{nk}^{y_n} (1 - s_{nk})^{1-y_n}}{\sum_{i=1}^K P_i^{(j)} s_{ni}^{y_n} (1 - s_{ni})^{1-y_n}}. \quad (12.108)$$

Note that in (12.108), the notation $s_{nk}^{(j)}, s_{ni}^{(j)}$ should have been used, but we tried to unclutter it slightly.

In the M-step, minimization with respect to P_k is of the same form as it was for the regression task, and it leads to

$$P_k^{(j+1)} = \frac{1}{N} \sum_{n=1}^N \gamma_{nk}. \quad (12.109)$$

To obtain the parameters for the experts, one has to resort to an iterative scheme. Observe that the only differences of (12.107) with (7.31) are (a) the presence of the term involving P_k , (b) the summation over k and (c) the existence of the multiplicative factors γ_{nk} . The first two make no difference in the optimization with respect to a single θ_k and the latter is just a constant. Hence the optimization is similar to the one used for the two-class logistic regression in Section 7.6, with the gradient and the Hessian matrices the same except for the multiplicative factors (and the sign because, there, the negative log-likelihood was considered). The extension to multiclass case is straightforward and follows similar steps.

Example 12.7. This example demonstrates the application of a mixture of two linear regression models to a synthetic data set. The input and the output are scalars, x_n, y_n . Figure 12.14a shows the setup. The data in the input space reside in different parts of the space and in each region the input-output relation is of a different form. The goal is to estimate the two linear functions, $\theta_{1,k}x + \theta_{0,k}$, $k \in \{1, 2\}$. The EM algorithm of Subsection 12.8.1 was initialized with the true value of the noise variance σ_η^2 .

Figures 12.14a, 12.14b and 12.14c show the resulting linear models after the 1st, the 7th, and finally the 15th iterations, respectively. Figure 12.14d, shows the resulting posteriors $P(k|y_n, x_n)$ (measured by the length of the bar) associated with each learner, as a function of x_n . After convergence, they are of a bimodal nature, depending on where each input sample resides in the input space. In this way, significant probability mass is assigned even to regions where data points do not exist. A smoother and more accurate, from a generalization point of view, estimate results if we let the gating parameters to be functions of the input variables themselves.

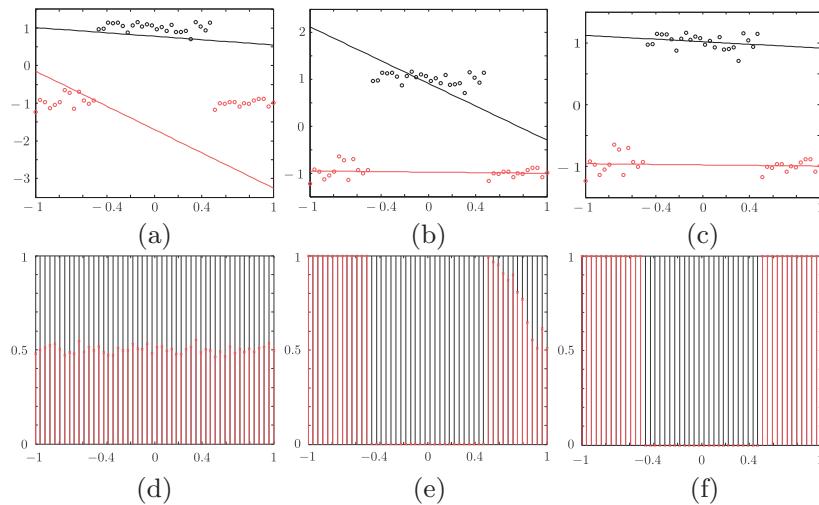


FIGURE 12.14

The two fitted lines as estimated by the EM algorithm after (a) the 1st, (b) the 7th, and (c) the 15th iterations. Figures (d)-(f) show the corresponding posterior probabilities, for each one of the training points x_n . The length of each segment is equal to the value of the respective probability.

PROBLEMS

- 12.1** Show that if

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \Sigma_z),$$

and

$$p(\mathbf{t}|\mathbf{z}) = \mathcal{N}(\mathbf{t}|A\mathbf{z}, \Sigma_{t|z}),$$

then

$$\mathbb{E}[\mathbf{z}|\mathbf{t}] = (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A)^{-1} (A^T \Sigma_{t|z}^{-1} \mathbf{t} + \Sigma_z^{-1} \boldsymbol{\mu}_z).$$

- 12.2** Prove that the binomial and beta distributions are conjugate pairs with respect to the mean value.

- 12.3** Show that the normalizing constant C in the Dirichlet pdf

$$\text{Dir}(\mathbf{x}|\boldsymbol{a}) = C \prod_{k=1}^K x_k^{a_k-1}, \quad \sum_{k=1}^K x_k = 1,$$

is given by

$$C = \frac{\Gamma(a_1 + a_2 + \dots + a_K)}{\Gamma(a_1)\Gamma(a_2)\dots\Gamma(a_K)}.$$

Hint. Use the property $\Gamma(a+1) = a\Gamma(a)$.

- (a) Use induction. Because the proposition is true for $k=2$ (beta distribution), assume that it is true for $k=K-1$, and prove that it will be true for $k=K$.
(b) Note that due to the constraint $\sum_{k=1}^K x_k = 1$, only $K-1$ of the variables are independent. So, basically the Dirichlet pdf implies that

$$p(x_1, x_2, \dots, x_{K-1}) = C \prod_{k=1}^{K-1} x_k^{a_k-1} \left(1 - \sum_{k=1}^{K-1} x_k\right)^{a_{K-1}-1}.$$

- 12.4** Show that $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$ for known Σ is of an exponential form and that its conjugate prior is also Gaussian.

- 12.5** Show that the conjugate prior of the multivariate Gaussian with respect to the precision matrix, Q , is a Wishart distribution.

- 12.6** Show that the conjugate prior of the univariate Gaussian $\mathcal{N}(x|\mu, \sigma^2)$ with respect to the mean and the precision $\beta = \frac{1}{\sigma^2}$, is the Gaussian-gamma product

$$p(\mu, \beta; \lambda, \mathbf{v}) = \mathcal{N}\left(\mu \middle| \frac{v_2}{\lambda}, (\lambda\beta)^{-1}\right) \text{Gamma}\left(\beta \middle| \frac{\lambda+1}{2}, \frac{v_1}{2} - \frac{v_2^2}{2\lambda}\right),$$

where $\mathbf{v} := [v_1, v_2]^T$.

- 12.7** Show that the multivariate Gaussian $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, Q^{-1})$ has as a conjugate prior, with respect to the mean and the precision matrix, Q , the Gaussian-Wishart product.

- 12.8** Show that the distribution

$$P(x|\mu) = \mu^x(1-\mu)^{1-x}, \quad x \in \{0, 1\},$$

is of an exponential form and derive its conjugate prior with respect to μ .

- 12.9** Show that estimating an unknown pdf by maximizing the respective entropy, subject to a set of empirical expectations, results in a pdf that belongs to the exponential family.
- 12.10** Let $\mathbf{x} \in \mathbb{R}^l$ be a random vector following the normal $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \Sigma)$. Consider $\mathbf{x}_n, n = 1, 2, \dots, N$, to be i.i.d. observations. If the prior for $\boldsymbol{\mu}$ follows $\mathcal{N}(\boldsymbol{\mu}|\boldsymbol{\mu}_0, \Sigma_0)$, show that the posterior $p(\boldsymbol{\mu}|\mathbf{x}_1, \dots, \mathbf{x}_N)$ is normal $\mathcal{N}(\boldsymbol{\mu}|\tilde{\boldsymbol{\mu}}, \tilde{\Sigma})$ with

$$\tilde{\Sigma}^{-1} = \Sigma_0^{-1} + N\Sigma^{-1},$$

and

$$\tilde{\boldsymbol{\mu}} = \tilde{\Sigma}(\Sigma_0^{-1}\boldsymbol{\mu}_0 + N\Sigma^{-1}\bar{\mathbf{x}}),$$

where $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$.

- 12.11** If \mathcal{X} is the set of observed variables and \mathcal{X}^l the set of the corresponding latent ones, show that

$$\frac{\partial \ln p(\mathcal{X}; \boldsymbol{\xi})}{\partial \boldsymbol{\xi}} = \mathbb{E} \left[\frac{\partial \ln p(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\xi})}{\partial \boldsymbol{\xi}} \right],$$

where $\mathbb{E}[\cdot]$ is with respect to $p(\mathcal{X}^l|\mathcal{X}; \boldsymbol{\xi})$ and $\boldsymbol{\xi}$ is an unknown vector parameter. Note that if one fixes the value of $\boldsymbol{\xi}$ in $p(\mathcal{X}^l|\mathcal{X}; \boldsymbol{\xi})$, then one has obtained the M-step of the EM algorithm.

- 12.12** Show that the Kullback-Leibler divergence $\text{KL}(p \parallel q)$ is a nonnegative quantity.

Hint. Recall that $\ln(\cdot)$ is a concave function and use Jensen's inequality, that is,

$$f \left(\int g(\mathbf{x})p(\mathbf{x})d\mathbf{x} \right) \leq \int f(g(\mathbf{x}))p(\mathbf{x})d\mathbf{x},$$

where $p(\mathbf{x})$ is a pdf and f is a convex function.

- 12.13** Let $\mathbf{y} \in \mathbb{R}^N$, $\boldsymbol{\theta} \in \mathbb{R}^l$ and Φ a matrix of appropriate dimensions. Derive the expected value of $\|\mathbf{y} - \Phi\boldsymbol{\theta}\|^2$ with respect to $\boldsymbol{\theta}$, given $\mathbb{E}[\boldsymbol{\theta}]$ and the corresponding covariance matrix Σ_θ .

- 12.14** Derive recursions (12.87)–(12.89).

MATLAB Exercises

- 12.15** Sample $N = 20$ equally spaced points x_n in the interval $[0, 2]$. Create the output samples, y_n , according to the nonlinear model of [Example 12.1](#), where the noise variance is set equal to $\sigma_\eta^2 = 0.05$.

- (a) Let the parameters of the Gaussian prior be $\boldsymbol{\theta}_0 = [0.2, -1, 0.9, 0.7, -0.2]^T$ and $\Sigma_\theta = 0.1I$. Compute the covariance matrix and the mean of the posterior Gaussian distribution using Eq. (12.19) and Eq. (12.20), respectively. Then, select randomly $K = 20$ points x_k in the interval $[0, 2]$. Compute the predictions for the mean values, μ_y , and the associated variances, σ_y^2 , utilizing Eq. (12.22) and Eq. (12.23), respectively. Plot the graph of the true function together with the predicted mean values, μ_y , and use MATLAB's “errorbar” function to show the confidence intervals on these predictions. Repeat the experiment again, using $N = 500$ points, and try different values of σ_η^2 , to notice the change on the estimated confidence intervals.

- (b) Repeat the previous experiment using a randomly chosen value for θ_0 and different values on the parameters, for example $\sigma_\eta^2 = 0.05$, or $\sigma_\eta^2 = 0.15$, $\sigma_\theta^2 = 0.1$, or $\sigma_\theta^2 = 2$, and $N = 500$ or $N = 20$.
- (c) Repeat the experiment once more using the alternative model as in (c) of [Example 12.1](#).

- 12.16** Consider [Example 12.1](#) as before. Sample $N = 500$ equally spaced points x_n in the interval $[0, 2]$. Create the output samples, y_n , according to the nonlinear model of the example, where the noise variance is set equal to $\sigma_\eta^2 = 0.05$. Implement the linear regression EM algorithm of [Section 12.6](#). Assume the correct number of parameters. Then repeat [Example 12.5](#). After the convergence of the EM, sample ten points, x_k , randomly, in the same interval as before and compute the predictive means μ_y and variances σ_y^2 . Plot the true signal curve, the predictive means μ_y , and the respective confidence intervals using Matlab's "errorbar" function. Repeat the EM run, using different initial values and an incorrect number of parameters. Comment on the results.
- 12.17** Generate 100 data points from each of the three two-dimensional Gaussian distributions of [Example 12.6](#). Plot the data points along with the confidence ellipsoids for each Gaussian with coverage probability 80%. Implement the Gaussian mixture model via the EM algorithm, whose steps are described in Eqs. (12.85)–(12.89). Moreover, compute the log-likelihood function in every iteration of the EM algorithm using Eq. (12.82).
- (a) In separate figures (always containing the data), plot the ellipsoids of the Gaussian distributions estimated by the EM algorithm during iterations $j = 1, j = 5$, and $j = 30$, and the log-likelihood function versus the number of iterations.
 - (b) Repeat the same experiment after bringing the cluster means closer together. Compare the results.
- 12.18** Generate 100 data points from each of the following two-dimensional Gaussian distributions with parameters

$$\mu_1^T = [0.9, 1.02]^T, \quad \mu_2^T = [-1.2, -1.3]^T$$

and

$$\Sigma_1 = \begin{bmatrix} 0.5 & 0.081 \\ 0.081 & 0.7 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 0.4 & 0.02 \\ 0.02 & 0.3 \end{bmatrix}.$$

Plot the data points using different colors for the two Gaussian distributions. Implement the k -means algorithm, presented in [Algorithm 12.1](#).

- (a) Run the k -means algorithm for $K = 2$ and plot the results. Run, also, the Gaussian mixtures EM of the previous exercise and plot the 80% probability confidence ellipsoids to compare the results.
- (b) Now, sample $N_1 = 100$ and $N_2 = 20$ points from each distribution and repeat the experiment to reproduce the results of [Figure 12.11d](#).
- (c) Try different configurations and play with K different than the true number of clusters. Comment on the results.
- (d) Play with different initialization points and also try points which are too far from the true mean values of the clusters. Comment on the results.

- 12.19** Generate 50 equidistant input data points in the interval $[-1, 1]$. Assume two linear regression models, the first with scale 0.005 and intercept -1 and the second with scale 0.018 and

intercept 1. Generate observations from these two models by using the first model for the input points in the interval $[-0.5, 0.5]$, and the second model for the inputs in the interval $[-1, -0.5] \cup [0.5, 1]$. Also, add Gaussian noise of zero-mean and variance 0.01. Next, implement the EM algorithm developed in [Section 12.8.1](#). Initialize the noise precision β to its true value. For iterations 1, 5, and 30, plot the data points and the estimated linear functions $\theta_{1,k}x + \theta_{0,k}$, $k \in \{1, 2\}$ of the models, to reproduce the results of [Figure 12.14](#).

12.9 APPENDIX TO CHAPTER 12

In this appendix, a number of results concerning the Gaussian pdf are derived. The reader is advised to work on these derivations to get familiar with the tools that are heavily used in Bayesian inference. Because the derived formulas are applicable to different parts of the book and to different variables, we denote the involved random vectors as \mathbf{z} and \mathbf{t} and one can substitute notation accordingly, depending on the notational needs for each case.

12.9.1 PDFs WITH EXPONENT OF QUADRATIC FORM

Let

$$p(\mathbf{z}) = \exp(F(\mathbf{z})), \quad (12.110)$$

where

$$F(\mathbf{z}) = -\frac{1}{2}\mathbf{z}^T Q \mathbf{z} + \mathbf{z}^T \mathbf{p} + C, \quad (12.111)$$

where C is a constant and $Q = Q^T$ and invertible. We rewrite (12.111) as

$$\begin{aligned} F(\mathbf{z}) &= -\frac{1}{2}\mathbf{z}^T Q \mathbf{z} + \mathbf{z}^T Q Q^{-1} \mathbf{p} + C \\ &= -\frac{1}{2}(\mathbf{z} - Q^{-1} \mathbf{p})^T Q (\mathbf{z} - Q^{-1} \mathbf{p}) + \frac{1}{2}\mathbf{p}^T Q^{-1} \mathbf{p} + C. \end{aligned} \quad (12.112)$$

From (12.112), (12.110) we get

$$p(\mathbf{z}) = \exp\left(\frac{1}{2}\mathbf{p}^T Q^{-1} \mathbf{p} + C\right) \exp\left(-\frac{1}{2}(\mathbf{z} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{z} - \boldsymbol{\mu})\right),$$

where

$$\boldsymbol{\mu} = Q^{-1} \mathbf{p}, \quad (12.113)$$

and

$$\Sigma = Q^{-1}. \quad (12.114)$$

Because $p(\mathbf{z})$ has to integrate to one then, necessarily,

$$\exp\left(\frac{1}{2}\mathbf{p}^T Q^{-1} \mathbf{p} + C\right) = \frac{1}{(2\pi)^{l/2} |\Sigma|^{1/2}},$$

where l is the dimensionality of the space where \mathbf{z} lies.

12.9.2 THE CONDITIONAL FROM THE JOINT GAUSSIAN PDF

Let $\mathbf{z} \in \mathbb{R}^{l_1}$ and $\mathbf{t} \in \mathbb{R}^{l_2}$ be two jointly Gaussian vectors, with $l = l_1 + l_2$. Let

$$\boldsymbol{\phi} = \begin{bmatrix} \mathbf{z} \\ \mathbf{t} \end{bmatrix}, \quad \mathbb{E}[\boldsymbol{\phi}] = \begin{bmatrix} \boldsymbol{\mu}_z \\ \boldsymbol{\mu}_t \end{bmatrix} := \boldsymbol{\mu}_{\boldsymbol{\phi}}.$$

Then

$$p(\mathbf{z}, \mathbf{t}) = p(\boldsymbol{\phi}) = \frac{1}{(2\pi)^{l/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (\boldsymbol{\phi} - \boldsymbol{\mu}_{\boldsymbol{\phi}})^T \Sigma_{\boldsymbol{\phi}}^{-1} (\boldsymbol{\phi} - \boldsymbol{\mu}_{\boldsymbol{\phi}})\right), \quad (12.115)$$

where

$$\Sigma_{\boldsymbol{\phi}} = \mathbb{E}\left[(\boldsymbol{\phi} - \boldsymbol{\mu}_{\boldsymbol{\phi}})(\boldsymbol{\phi} - \boldsymbol{\mu}_{\boldsymbol{\phi}})^T\right] = \begin{bmatrix} \Sigma_z & \Sigma_{zt} \\ \Sigma_{tz} & \Sigma_t \end{bmatrix}, \quad (12.116)$$

with

$$\Sigma_z = \mathbb{E}\left[(\mathbf{z} - \boldsymbol{\mu}_z)(\mathbf{z} - \boldsymbol{\mu}_z)^T\right], \quad \Sigma_t = \mathbb{E}\left[(\mathbf{t} - \boldsymbol{\mu}_t)(\mathbf{t} - \boldsymbol{\mu}_t)^T\right], \quad (12.117)$$

and

$$\Sigma_{zt} = \mathbb{E}\left[(\mathbf{z} - \boldsymbol{\mu}_z)(\mathbf{t} - \boldsymbol{\mu}_t)^T\right] = \Sigma_{tz}^T. \quad (12.118)$$

We will prove that $p(\mathbf{z}|\mathbf{t})$ is also Gaussian. In a similar way, one proves the same for $p(\mathbf{t}|\mathbf{z})$. To this end, we will show that (12.115) is quadratic with respect to \mathbf{z} . Indeed,

$$\boldsymbol{Q}_{\boldsymbol{\phi}} := \Sigma_{\boldsymbol{\phi}}^{-1} = \begin{bmatrix} \Sigma_z & \Sigma_{zt} \\ \Sigma_{tz} & \Sigma_t \end{bmatrix}^{-1} := \begin{bmatrix} \boldsymbol{Q}_z & \boldsymbol{Q}_{zt} \\ \boldsymbol{Q}_{tz} & \boldsymbol{Q}_t \end{bmatrix}, \quad (12.119)$$

where \boldsymbol{Q}_z is $l_1 \times l_1$, \boldsymbol{Q}_t is $l_2 \times l_2$ and $\boldsymbol{Q}_{zt} = \boldsymbol{Q}_{tz}^T$ are $l_1 \times l_2$ matrices. The exponent in (12.115) becomes

$$\text{EXP} = -\frac{1}{2} [\mathbf{z}^T - \boldsymbol{\mu}_z^T, \mathbf{t}^T - \boldsymbol{\mu}_t^T] \begin{bmatrix} \boldsymbol{Q}_z & \boldsymbol{Q}_{zt} \\ \boldsymbol{Q}_{tz} & \boldsymbol{Q}_t \end{bmatrix} \begin{bmatrix} \mathbf{z} - \boldsymbol{\mu}_z \\ \mathbf{t} - \boldsymbol{\mu}_t \end{bmatrix}, \quad (12.120)$$

which after some trivial algebra becomes

$$\text{EXP} = -\frac{1}{2} \mathbf{z}^T \boldsymbol{Q}_z \mathbf{z} + \mathbf{z}^T \boldsymbol{Q}_z \boldsymbol{\mu}_z - \mathbf{z}^T \boldsymbol{Q}_{zt} (\mathbf{t} - \boldsymbol{\mu}_t) + C(\mathbf{t}), \quad (12.121)$$

where

$$C(\mathbf{t}) = -\frac{1}{2} (\mathbf{t} - \boldsymbol{\mu}_t)^T \boldsymbol{Q}_t (\mathbf{t} - \boldsymbol{\mu}_t) + \boldsymbol{\mu}_z^T \boldsymbol{Q}_{zt} (\mathbf{t} - \boldsymbol{\mu}_t) - \frac{1}{2} \boldsymbol{\mu}_z^T \boldsymbol{Q}_z \boldsymbol{\mu}_z, \quad (12.122)$$

which is considered constant, because in the conditional $p(\mathbf{z}|\mathbf{t})$ we fix the values of \mathbf{t} . It is readily seen that the exponent is quadratic with respect to \mathbf{z} and of the form given in (12.111),

$$\text{EXP} = -\frac{1}{2} \mathbf{z}^T \boldsymbol{Q}_z \mathbf{z} + \mathbf{z}^T (\boldsymbol{Q}_z \boldsymbol{\mu}_z - \boldsymbol{Q}_{zt} (\mathbf{t} - \boldsymbol{\mu}_t)) + C(\mathbf{t}). \quad (12.123)$$

Hence, combining with (12.113, 12.114) we get for the conditional mean

$$\boldsymbol{\mu}_{z|t} := \mathbb{E}[\mathbf{z}|\mathbf{t}] = \boldsymbol{\mu}_z - \boldsymbol{Q}_z^{-1} \boldsymbol{Q}_{zt} (\mathbf{t} - \boldsymbol{\mu}_t) \quad (12.124)$$

and the conditional covariance

$$\Sigma_{z|t} = Q_z^{-1}. \quad (12.125)$$

It suffices to compute Q_z and Q_{zt} in terms of the known Σ_z , Σ_t , Σ_{zt} . From (12.119) and using the matrix inversion lemmas (Appendix A.1) we get

$$Q_z = \Sigma_z^{-1} + \Sigma_z^{-1} \Sigma_{zt} \left(\Sigma_t - \Sigma_{tz} \Sigma_z^{-1} \Sigma_{zt} \right)^{-1} \Sigma_{tz} \Sigma_z^{-1} \quad (12.126)$$

$$= (\Sigma_z - \Sigma_{zt} \Sigma_t^{-1} \Sigma_{tz})^{-1} \quad (12.127)$$

$$Q_{zt} = -\Sigma_z^{-1} \Sigma_{zt} \left(\Sigma_t - \Sigma_{tz} \Sigma_z^{-1} \Sigma_{zt} \right)^{-1} \quad (12.128)$$

$$= -\left(\Sigma_z - \Sigma_{zt} \Sigma_t^{-1} \Sigma_{tz} \right)^{-1} \Sigma_{zt} \Sigma_t^{-1}. \quad (12.129)$$

Thus, from (12.125) and (12.127) we obtain

$$\Sigma_{z|t} = Q_z^{-1} = \Sigma_z - \Sigma_{zt} \Sigma_t^{-1} \Sigma_{tz}, \quad (12.130)$$

and combining (12.127) and (12.129) with (12.124) we obtain

$$\boldsymbol{\mu}_{z|t} = \boldsymbol{\mu}_z + \left(\Sigma_z - \Sigma_{zt} \Sigma_t^{-1} \Sigma_{tz} \right) \left(\Sigma_z - \Sigma_{zt} \Sigma_t^{-1} \Sigma_{tz} \right)^{-1} \Sigma_{zt} \Sigma_t^{-1} (\mathbf{t} - \boldsymbol{\mu}_t),$$

or

$$\boldsymbol{\mu}_{z|t} = \boldsymbol{\mu}_z + \Sigma_{zt} \Sigma_t^{-1} (\mathbf{t} - \boldsymbol{\mu}_t). \quad (12.131)$$

12.9.3 THE MARGINAL FROM THE JOINT GAUSSIAN PDF

Our next goal is to compute the marginal pdf of either of the two involved jointly Gaussian variables, for example,

$$p(\mathbf{t}) = \int p(\mathbf{z}, \mathbf{t}) d\mathbf{z},$$

and show that it is also Gaussian.

Similar arguments will follow for the computation of $p(\mathbf{z})$. The exponent of the joint pdf from (12.122), (12.123), and (12.124) becomes

$$\begin{aligned} \text{EXP} &= -\frac{1}{2} (\mathbf{z} - \boldsymbol{\mu}_{z|t})^T Q_z (\mathbf{z} - \boldsymbol{\mu}_{z|t}) + \frac{1}{2} \boldsymbol{\mu}_{z|t}^T Q_z \boldsymbol{\mu}_{z|t} \\ &\quad - \frac{1}{2} (\mathbf{t} - \boldsymbol{\mu}_t)^T Q_t (\mathbf{t} - \boldsymbol{\mu}_t) + \boldsymbol{\mu}_z^T Q_{zt} (\mathbf{t} - \boldsymbol{\mu}_t) - \frac{1}{2} \boldsymbol{\mu}_z^T Q_z \boldsymbol{\mu}_z \\ &= -\frac{1}{2} (\mathbf{z} - \boldsymbol{\mu}_{z|t})^T Q_z (\mathbf{z} - \boldsymbol{\mu}_{z|t}) + F(\mathbf{t}). \end{aligned} \quad (12.132)$$

Observe that both terms in (12.132) are functions of \mathbf{t} . However, combining (12.115), (12.120), (12.132) we get

$$p(\mathbf{t}) = \int p(\mathbf{z}, \mathbf{t}) d\mathbf{z} \propto \exp(F(\mathbf{t})), \quad (12.133)$$

because

$$\int \exp\left(-\frac{1}{2}(z - \mu_{z|t})^T Q_z (z - \mu_{z|t})\right) dz,$$

being the integral of the exponent of a Gaussian will depend on $|Q_z|$ only and it is independent of $\mu_{z|t}$, which is a function of t . Hence, it suffices to check in (12.133) if $F(t)$ is quadratic in t . From the respective definition in (12.132) we obtain

$$F(t) = -\frac{1}{2}(t - \mu_t)^T Q_t (t - \mu_t) + \mu_z^T Q_{zt} (t - \mu_t) + \frac{1}{2} \mu_{z|t}^T Q_z \mu_{z|t} - \frac{1}{2} \mu_z^T Q_z \mu_z, \quad (12.134)$$

and using (12.124) we can readily check that $p(t)$ is of quadratic form. However, we need not manipulate further (12.134) to find the respective mean and covariance matrix. Because the original joint pdf was specified, then from (12.117) μ_t and Σ_t are readily available. No doubt, these are the same values that would result by manipulating (12.134), and the reader can verify it as an exercise.

12.9.4 THE POSTERIOR FROM GAUSSIAN PRIOR AND CONDITIONAL PDFs

Let

$$p(z) = \mathcal{N}(z|\mu_z, \Sigma_z), \quad (12.135)$$

and

$$p(t|z) = \mathcal{N}(t|Az, \Sigma_{t|z}). \quad (12.136)$$

The goal is to show that $p(z|t)$ and $p(t)$ are also Gaussians.

Path 1: We will first show that \mathbf{z} and \mathbf{t} are jointly Gaussian. We have that

$$p(z, t) = p(t|z)p(z), \quad (12.137)$$

which from (12.135), (12.136) turns out to be

$$p(z, t) \propto \exp(F(z, t)),$$

where

$$\begin{aligned} F(z, t) &= -\frac{1}{2}(t - Az)^T \Sigma_{t|z}^{-1} (t - Az) \\ &\quad - \frac{1}{2}(z - \mu_z)^T \Sigma_z^{-1} (z - \mu_z). \end{aligned} \quad (12.138)$$

Then, after some simple algebraic manipulations we have,

$$\begin{aligned} F(z, t) &= -\frac{1}{2} \mathbf{t}^T \Sigma_{t|z}^{-1} \mathbf{t} - \frac{1}{2} \mathbf{z}^T (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A) \mathbf{z} \\ &\quad + \mathbf{t}^T \Sigma_{t|z}^{-1} A \mathbf{z} + \mathbf{z}^T \Sigma_z^{-1} \mu_z - \frac{1}{2} \mu_z^T \Sigma_z^{-1} \mu_z, \end{aligned}$$

or

$$F(z, t) = -\frac{1}{2} [\mathbf{z}^T, \mathbf{t}^T] \begin{bmatrix} \Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A & -A^T \Sigma_{t|z}^{-1} \\ -\Sigma_{t|z}^{-1} A & \Sigma_{t|z}^{-1} \end{bmatrix} \begin{bmatrix} z \\ t \end{bmatrix}$$

$$+ [z^T, t^T] \begin{bmatrix} \Sigma_z^{-1} \mu_z \\ 0 \end{bmatrix} + C, \quad (12.139)$$

which is obviously in the quadratic form of (12.111) for the joint variables

$$\phi = \begin{bmatrix} z \\ t \end{bmatrix}.$$

Then we already know that $p(z|t)$ will also be Gaussian, with mean and covariance matrix resulting by combining (12.124), (12.125),

$$\mu_{z|t} = \mu_z + (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A)^{-1} A^T \Sigma_{t|z}^{-1} (t - \mu_t) \quad (12.140)$$

$$\Sigma_{z|t} = (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A)^{-1}. \quad (12.141)$$

It suffices to compute μ_t . From (12.139) and (12.113) we have

$$\mathbb{E}[\Phi] = \begin{bmatrix} \mu_z \\ \mu_t \end{bmatrix} = \begin{bmatrix} \Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A & -A^T \Sigma_{t|z}^{-1} \\ -\Sigma_{t|z}^{-1} A & \Sigma_{t|z}^{-1} \end{bmatrix}^{-1} \begin{bmatrix} \Sigma_z^{-1} \mu_z \\ 0 \end{bmatrix},$$

or taking into consideration the matrix inversion lemma with respect to the low partition and some algebraic manipulations, we obtain

$$\begin{bmatrix} \mu_z \\ \mu_t \end{bmatrix} = \begin{bmatrix} \Sigma_z & \Sigma_z A^T \\ A \Sigma_z & \Sigma_{t|z} + A \Sigma_z A^T \end{bmatrix} \begin{bmatrix} \Sigma_z^{-1} \mu_z \\ 0 \end{bmatrix}, \quad (12.142)$$

or

$$\mu_t = A \mu_z \quad (12.143)$$

$$\mu_{z|t} = \mu_z + (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A)^{-1} A^T \Sigma_{t|z}^{-1} (t - A \mu_z)$$

$$\Sigma_{z|t} = (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A)^{-1}. \quad (12.144)$$

Using the matrix inversion lemma from Appendix A.1,

$$(A^{-1} + B^T C^{-1} B)^{-1} B^T C^{-1} = AB^T (BAB^T + C)^{-1}$$

we get

$$\mu_{z|t} = \mu_z + \Sigma_z A^T (\Sigma_{t|z} + A \Sigma_z A^T)^{-1} (t - A \mu_z). \quad (12.145)$$

Also, by applying Woodbury's identity (Appendix A.1) in (12.144) we obtain an alternative expression for $\Sigma_{z|t}$,

$$\Sigma_{z|t} = \Sigma_z - \Sigma_z A^T (\Sigma_{t|z} + A \Sigma_z A^T)^{-1} A \Sigma_z. \quad (12.146)$$

It now remains to derive the marginal $p(t)$. Because \mathbf{z} , \mathbf{t} are jointly Gaussian, $p(t)$ is also Gaussian with mean

$$\mu_t = A \mu_z, \quad (12.147)$$

and using $\Sigma_\phi := Q_\phi^{-1}$ in (12.142)

$$\Sigma_t = \Sigma_{t|z} + A \Sigma_z A^T. \quad (12.148)$$

Finally, one can show via the use of the matrix inversion lemmas that $\mu_{z|t}$ in (12.143) is equal to (Problem 12.1)

$$\mu_{z|t} = \left(\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A \right)^{-1} \left(A^T \Sigma_{t|z}^{-1} t + \Sigma_z^{-1} \mu_z \right). \quad (12.149)$$

Path 2: There is another more direct path to obtain $p(z|t)$, if one wants to bypass the joint distribution. From (12.135, 12.136) we obtain that

$$\begin{aligned} p(z|t) &\propto \exp \left(-\frac{1}{2} z^T \Sigma_z^{-1} z + z^T \Sigma_z^{-1} \mu_z \right) \times \\ &\quad \exp \left(-\frac{1}{2} z^T A^T \Sigma_{t|z}^{-1} A z + z^T A^T \Sigma_{t|z}^{-1} t \right), \end{aligned}$$

or

$$\begin{aligned} p(z|t) &\propto \exp \left(-\frac{1}{2} z^T (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A) z \right. \\ &\quad \left. + z^T (\Sigma_z^{-1} \mu_z + A^T \Sigma_{t|z}^{-1} t) \right) \\ &:= \exp \left(-\frac{1}{2} z^T Q_{z|t} z + z^T p_{z|t} \right), \end{aligned}$$

which is of quadratic form and using (12.113, 12.114) leads to the conclusion that

$$\Sigma_{z|t} = Q_{z|t}^{-1} = \left(\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A \right)^{-1}$$

and

$$\begin{aligned} \mu_{z|t} &= (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A)^{-1} p_{z|t} \\ &= (\Sigma_z^{-1} + A^T \Sigma_{t|z}^{-1} A)^{-1} (A^T \Sigma_{t|z}^{-1} t + \Sigma_z^{-1} \mu_z). \end{aligned}$$

Remarks 12.6.

- Let

$$Q_z = \Sigma_z^{-1}, \quad p_z = Q_z \mu_z, \quad Q_{t|z} = \Sigma_{t|z}^{-1}, \quad p_{t|z} = A^T Q_{t|z} t.$$

Then observe that the posterior $p(z|t)$ is normal with

$$Q_{z|t} = Q_z + A^T Q_{t|z} A, \quad (12.150)$$

and

$$p_{z|t} = p_z + p_{t|z}. \quad (12.151)$$

Observe in (12.150) and (12.151) that if we express the involved normal pdfs in terms of their precision matrices and the respective p vectors, as in (12.111), then these parameters are added to

give the respective parameters of the posterior. This property complies with what we have said in Section 12.4 concerning the conjugate priors of the exponential family.

REFERENCES

- [1] D. Arthur, S. Vassilvitskii, *k*-means++: the advantages of careful seeding, in: Proceedings 18th ACM-SIAM Symposium on Discrete algorithms, SODA, 2007, pp. 1027-1035.
- [2] L.E. Baum, T. Petrie, G. Soules, N. Weiss, A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains, *Ann. Math. Stat.* 41 (1970) 164-171.
- [3] M.J. Beal, Variational algorithms for approximate Bayesian inference, Ph.D. Thesis, University College London, 2003.
- [4] L. Bo, C. Sminchisescu, A. Kanaujia, D. Metaxas, Fast algorithms for large scale conditional 3D prediction, in: Proceedings International Conference to Computer Vision and Pattern Recognition, CVPR, Anchorage, AK, 2008.
- [5] J.S. Bridle, Probabilistic interpretation of feedforward classification network outputs with relationship to statistical pattern recognition, in: F. Fougeron-Soulie, J. Heurault (Eds.), *Nuero-Computing: Algorithms, Architectures and Applications*, Springer Verlag, 1990.
- [6] O. Cappe, E. Mouline, Online EM algorithm for latent data models, *J. R. Stat. Soc. B* 71(3) (2009) 593-613.
- [7] G. Casella, R.L. Berger, *Statistical Inference*, second ed., Duxbury, 2002.
- [8] G. Celeux, J. Diebolt, The SEM algorithm: A probabilistic teacher derived from the EM algorithm for the mixture problem, *Comput. Stat. Quart.* 2 (1985) 73-82.
- [9] S.P. Chatzis, D.I. Kosmopoulos, T.A. Varvarigou, Signal modeling and classification using a robust latent space model based on *t*-distributions, *IEEE Trans. Signal Process.* 56(3) (2008) 949-963.
- [10] S.P. Chatzis, D. Kosmopoulos, T.A. Varvarigou, Robust sequential data modeling using an outlier tolerant hidden Markov model, *IEEE Trans. Pattern Anal. Machine Intell.* 31(9) (2009) 1657-1669.
- [11] B. Delyon, M. Lavielle, E. Moulines, Convergence of a stochastic approximation version of the EM algorithm, *Ann. Stat.* 27(1) (1999) 94-128.
- [12] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, *J. R. Stat. Soc. B* 39(1) (1977) 1-38.
- [13] R.P. Feynman, *Statistical Mechanics: A Set of Lectures*, Addison-Wesley, Reading, MA, 1998.
- [14] S.F. Gull, Bayesian inductive inference and maximum entropy, in: G.J. Erickson, C.R. Smith (Eds.), *Maximum Entropy and Bayesian Methods in Science and Engineering*, Kluwer, 1988.
- [15] M.R. Gupta, Y. Chen, Theory and Use of the EM Algorithm, *Found. Trends Signal Process.* 4(3) (2010) 223-299.
- [16] L.K. Hansen, C.E. Rasmussen, Pruning from adaptive regularization, *Neural Comput.* 6 (1993) 1223-1232.
- [17] L.K. Hansen, J. Larsen, Unsupervised learning and generalization, in: *IEEE International Conference on Neural Networks*, 1996, pp. 25-30.
- [18] L.K. Hansen, Bayesian averaging is well-tempered, in: S. Solla (Ed.), *Proceedings Neural Information Processing*, NIPS, MIT Press, 2000, pp. 265-271.
- [19] D. Haussler, M. Kearns, R. Schapire, Bounds on the sample complexity of Bayesian learning using information theory and the VC dimension, *Machine Learn.* 14 (1994) 83-113.
- [20] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1999.
- [21] D.R. Hunter, K. Lange, A tutorial on MM algorithms, *Amer. Statist.* 58 (2004) 30-37.
- [22] E.T. Jaynes, On the rationale of the maximum entropy methods, *Proc. IEEE* 70(9) (1982) 939-952.
- [23] E.T. Jaynes, *Bayesian Methods-an introductory tutorial*, in: J.H. Justice (Ed.), *Maximum Entropy and Bayesian Methods in Science and Engineering*, Cambridge University Press, 1986.

- [24] H. Jeffreys, *Theory of Probability*, Oxford University Press, 1992.
- [25] M.I. Jordan, R.A. Jacobs, Hierarchical mixture of experts and the EM algorithm, *Neural Comput.* 6 (1994) 181-214.
- [26] P. Liang, D. Klein, Online EM for unsupervised models, in: Proceeding of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, NAACL, 2009, pp. 611-619.
- [27] T.J. Loredo, From Laplace to supernova SN 1987A: Bayesian inference in astrophysics, in: P. Fougere (Ed.), *Maximum entropy and Bayesian methods*, Kluwer, 1990, pp. 81-143.
- [28] D.J.C. McKay, Bayesian interpolation, *Neural Comput.* 4(3) (1992) 417-447.
- [29] D.J.C. MacKay, The evidence framework applied to classification networks, *Neural Comput.* 4 (1992) 720-736.
- [30] D.J.C. McKay, Probable networks and plausible predictions – a review of practical Bayesian methods for supervised neural networks. *Network Comput. Neural Syst.* 6 (1995) 469-505.
- [31] X.L. Meng, D. Van Dyk, The EM algorithman old folk-song sung to a fast new tune, *J. R. Stat. Soc. B* 59(3) (1997) 511-567.
- [32] G.J. McLachlan, K.E. Basford, *Mixture Models. Inference and Applications to Clustering*, Marcel Dekker, 1988.
- [33] X.L. Meng, D.B. Rubin, Maximum likelihood estimation via the ECM algorithm: a generalization framework, *Biometrika* 80 (1993) 267-278.
- [34] J.E. Moody, Note on generalization, regularization, and architecture selection in nonlinear learning systems, in: Proceedings, IEEE Workshop on Neural Networks for Signal Processing, Princeton, NJ, USA, 1991, pp. 1-10.
- [35] T. Moon, The expectation maximization algorithm, *Signal Process. Mag.* 13(6) (1996) 47-60.
- [36] R.M. Neal, G.E. Hinton, A new view of the EM algorithm that justifies incremental, sparse and other variants, in: M.J. Jordan (Ed.), *Learning in Graphical Models*, Kluwer Academic Publishers, 1998, pp. 355-369.
- [37] S. Shoham, Robust clustering by deterministic agglomeration EM of mixtures of multivariate t distributions, *Pattern Recognit.* 35(5) (2002) 1127-1142.
- [38] M. Stephens, Dealing with label-switching in mixture models, *J. R. Statist. Soc. B* 62 (2000) 795-809.
- [39] M. Svensen, C.M. Bishop, Robust Bayesian mixture modeling, *Neurocomputing* 64 (2005) 235-252.
- [40] R. Sundberg, Maximum likelihood theory for incomplete data from an exponential family, *Scand. J. Statist.* 1(2) (1974) 49-58.
- [41] G. Schwarz, Estimating the dimension of a model, *Ann. Stat.* 6 (1978) 461-464.
- [42] J. Shao, *Mathematical Statistics: Exercises and Solutions*, Springer, 2005.
- [43] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, fourth ed., Academic Press, 2009.
- [44] Y. Tikochinsky, N.Z. Tishby, R.D. Levin, Alternative approach to maximum-entropy inference, *Phys. Rev. A* 30(5) (1985) 2638-2644.
- [45] D.M. Titterington, A.F.M. Smith, U.E. Makov, *Statistical Analysis of Finite Mixture Distributions*, John Wiley & Sons, 1985.
- [46] A. Vehtari, J. Lampinen, Bayesian model assessment and comparison using cross-validation predictive densities, *Neural Comput.* 14 (10) (2002) 2439-2468.
- [47] C.S. Wallace, P.R. Freeman, Estimation and inference by compact coding, *J. R. Stat. Soc. B* 493 (1987) 240-265.
- [48] R.L. Wolpert, Exponential families, Technical Report, University of Duke, www.stat.duke.edu/courses/Spring11/sta114/lec/expofam.pdf, 2011.
- [49] C. Wu, On the convergence properties of the EM algorithm, *Ann. Stat.* 11(1) (1983) 95-103.

BAYESIAN LEARNING: APPROXIMATE INFERENCE AND NONPARAMETRIC MODELS

13

CHAPTER OUTLINE

13.1	Introduction	650
13.2	Variational Approximation in Bayesian Learning	650
	<i>The Mean Field Approximation</i>	651
13.2.1	The Case of the Exponential Family of Probability Distributions	654
13.3	A Variational Bayesian Approach to Linear Regression	655
	<i>Computation of the Lower Bound</i>	660
13.4	A Variational Bayesian Approach to Gaussian Mixture Modeling	661
13.5	When Bayesian Inference Meets Sparsity.....	665
13.6	Sparse Bayesian Learning (SBL)	667
13.6.1	The Spike and Slab Method	670
13.7	The Relevance Vector Machine Framework	671
13.7.1	Adopting the Logistic Regression Model for Classification	672
13.8	Convex Duality and Variational Bounds	676
13.9	Sparsity-Aware Regression: A Variational Bound Bayesian Path.....	681
13.10	Sparsity-Aware Learning: Some Concluding Remarks	685
	<i>Parameter Identifiability and Sparse Bayesian Modeling</i>	688
13.11	Expectation Propagation	689
	<i>Minimizing the KL Divergence</i>	690
	<i>The Expectation Propagation Algorithm</i>	691
13.12	Nonparametric Bayesian Modeling.....	693
13.12.1	The Chinese Restaurant Process	694
13.12.2	Inference	694
13.12.3	Dirichlet Processes.....	694
13.12.4	The Stick-Breaking Construction of a DP	695
13.13	Gaussian Processes	697
13.13.1	Covariance Functions and Kernels	698
13.13.2	Regression	700
	<i>Dealing with Hyperparameters</i>	701
	<i>Computational Considerations</i>	702
13.13.3	Classification	702

13.14 A Case Study: Hyperspectral Image Unmixing	703
13.14.1 Hierarchical Bayesian Modeling	705
13.14.2 Experimental Results	706
Problems.....	709
<i>MATLAB Exercises.....</i>	711
References.....	712

13.1 INTRODUCTION

This chapter is the second one dedicated to Bayesian learning. The emphasis here, compared to [Chapter 12](#), is on more advanced topics, dealing with approximate inference methods. Such methods are employed when the involved integrations are no longer computationally tractable. Two paths for approximate inference, known as variational techniques, are discussed. One is based on the mean field approximation and the lower bound interpretation of the EM, and the other on convex duality and variational bounds. Regression and mixture modeling are discussed in this framework. Emphasis is given to sparse Bayesian modeling techniques and hierarchical Bayesian models. The relevance vector machine framework is presented. Expectation propagation is also discussed as an alternative to variational methods for approximate inference. At the end of the chapter, Bayesian learning in the context of nonparametric models is discussed, including Gaussian processes. Finally, a case study concerning hyperspectral imaging is presented.

13.2 VARIATIONAL APPROXIMATION IN BAYESIAN LEARNING

Recall that in order to apply the EM algorithm, the functional form of the posterior of the latent variables, given the observations, must be known. However, the analytic computation of the posterior is not always tractable. In such cases, the EM algorithm, in its standard form as discussed in the previous chapter, is not applicable. In this section, we will describe an alternative path that builds upon the EM interpretation given in [Section 12.5.2](#).

Once more, we will adopt a general notation, which can then be adapted to the needs of specific problems. Let \mathcal{X} be the set of observed variables and \mathcal{X}^l the respective set of latent ones, as they were defined in [Section 12.5](#). Furthermore, in this section, we will explicitly bring into the game the set of parameters, $\theta \in \mathbb{R}^K$, which are treated as random variables in the Bayesian context, accompanied by a prior pdf. Note that although we could consider the parameters as latent variables, we do not. Here, we reserve the term “latent” for hidden variables whose number depends on the number of observations, N . In contrast, a random parameter vector, θ , though a hidden random vector, has a fixed dimension. The reason we do that is to allow us to employ a prior pdf only for the parameters, in order to serve the needs of our examples. The functional in Eq. (12.63) is now redefined as

$$\mathcal{F}(q, \xi) = \int q(\mathcal{X}^l, \theta) \ln \frac{p(\mathcal{X}, \mathcal{X}^l, \theta; \xi)}{q(\mathcal{X}^l, \theta)} d\mathcal{X}^l d\theta, \quad (13.1)$$

where ξ is the set of deterministic (hyper)parameters. Let us, for the time being, get rid of ξ for the sake of notational relaxation. Then the counterpart of Eq. (12.64) becomes

$$\mathcal{F}(q) = \ln p(\mathcal{X}) + \int q(\mathcal{X}^l, \boldsymbol{\theta}) \ln \frac{p(\mathcal{X}^l, \boldsymbol{\theta} | \mathcal{X})}{q(\mathcal{X}^l, \boldsymbol{\theta})} d\mathcal{X}^l d\boldsymbol{\theta}. \quad (13.2)$$

The difference with Eq. (12.64) lies in the fact that $p(\mathcal{X}^l, \boldsymbol{\theta} | \mathcal{X})$ is not known, so maximizing Eq. (13.2) with respect to q by setting to zero the KL divergence $\text{KL}(q || p(\mathcal{X}^l, \boldsymbol{\theta} | \mathcal{X}))$ is no longer possible.

Optimizing a functional with respect to a function is known in mathematics as *calculus of variations*. The simplest example of this problem is to compute the geodesic that connects two points on a surface. Two names whose contributions are considered significant breakthroughs that consolidated this field are the Swiss German mathematician Leonhard Euler (1707-1783) and the Italian-born mathematician and astronomer Joseph-Louis Lagrange (1736-1813). It is interesting to note that Lagrange succeeded Euler as director of mathematics in the Prussian Academy of Sciences in Berlin.

In order to deal with the current problem, we will constrain $q(\mathcal{X}^l, \boldsymbol{\theta})$ to lie within a family of functions. Note that in this case, if the unknown $p(\mathcal{X}^l, \boldsymbol{\theta} | \mathcal{X})$ does not belong to this specific family of functions, the KL divergence *cannot* become zero, and the lower bound, $\mathcal{F}(q)$, of the marginal log-likelihood *cannot* be made tight. This is the reason the method is called a variational approximation.

The mean field approximation

This type of approximation results by constraining $q(\mathcal{X}^l, \boldsymbol{\theta})$ to be factorized, that is,

$$q(\mathcal{X}^l, \boldsymbol{\theta}) = q_{\mathcal{X}^l}(\mathcal{X}^l)q_{\boldsymbol{\theta}}(\boldsymbol{\theta}). \quad (13.3)$$

This factorization can be, and usually is, extended to

$q(\mathcal{X}^l, \boldsymbol{\theta}) = q_{x_1^l}(x_1^l) \dots q_{x_N^l}(x_N^l)q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) : \quad \text{Mean Field Approximation.}$

(13.4)

Also, the hidden variables can be factorized in groups. Similarly, the parameter factor can be further factorized if the parameters can be grouped in different groups, as is often the case. To simplify our notation, without sacrificing generality, we will work with Eq. (13.3). This type of approximation has been inspired from the field of statistical physics and is known as *mean field approximation* (e.g., [14, 40, 53]).

Having adopted Eq. (13.3) and recalling that $p(\mathcal{X}, \mathcal{X}^l, \boldsymbol{\theta}) = p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta})p(\boldsymbol{\theta})$, Eq. (13.1) becomes (Problem 13.1),

$$\begin{aligned} \mathcal{F}(q_{\mathcal{X}^l}, q_{\boldsymbol{\theta}}) &= \int q_{\mathcal{X}^l}(\mathcal{X}^l) \left(\int q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \ln p(\mathcal{X}, \mathcal{X}^l, \boldsymbol{\theta}) d\boldsymbol{\theta} \right) d\mathcal{X}^l \\ &\quad - \int q_{\mathcal{X}^l}(\mathcal{X}^l) \ln q_{\mathcal{X}^l}(\mathcal{X}^l) d\mathcal{X}^l - \int q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \ln q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta}, \end{aligned} \quad (13.5)$$

or equivalently

$$\begin{aligned} \mathcal{F}(q_{\mathcal{X}^l}, q_{\boldsymbol{\theta}}) &= \int q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \left(\int q_{\mathcal{X}^l}(\mathcal{X}^l) \ln \left(p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta})p(\boldsymbol{\theta}) \right) d\mathcal{X}^l \right) d\boldsymbol{\theta} \\ &\quad - \int q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) \ln q_{\boldsymbol{\theta}}(\boldsymbol{\theta}) d\boldsymbol{\theta} - \int q_{\mathcal{X}^l}(\mathcal{X}^l) \ln q_{\mathcal{X}^l}(\mathcal{X}^l) d\mathcal{X}^l. \end{aligned} \quad (13.6)$$

Having expressed the lower bound, $\mathcal{F}(q_{\mathcal{X}^l}, q_{\boldsymbol{\theta}})$, as in Eqs. (13.5) and (13.6), maximization with respect to $q(\mathcal{X}^l, \boldsymbol{\theta})$ (as required by the E-step of the EM algorithm) will take place by splitting the process in

order to maximize first with respect to $q_{\mathcal{X}^l}$ and then with respect to q_θ . Bringing back into the scene the (deterministic) parameter vector, ξ , and initializing the algorithm from arbitrary values for $\xi^{(0)}$ as well as for the involved statistics related to q_θ (this will become clear while dealing with the examples), the $(j+1)$ iteration comprises the following steps:

E-Step 1a: Holding $\xi^{(j)}$ and $q_\theta^{(j)}$ fixed, optimize Eq. (13.5) with respect to $q_{\mathcal{X}^l}$, that is,

$$\begin{aligned} q_{\mathcal{X}^l}^{(j+1)}(\mathcal{X}^l) &= \max_{q_{\mathcal{X}^l}} \mathcal{F}(q_{\mathcal{X}^l}(\mathcal{X}^l), q_\theta^{(j)}(\boldsymbol{\theta})) \\ &= \max_{q_{\mathcal{X}^l}} \int q_{\mathcal{X}^l}(\mathcal{X}^l) \ln \frac{\tilde{p}(\mathcal{X}, \mathcal{X}^l; \xi^{(j)})}{q_{\mathcal{X}^l}(\mathcal{X}^l)} d\mathcal{X}^l + \text{constant}, \end{aligned} \quad (13.7)$$

where “constant” contains all the terms that do not depend on \mathcal{X}^l and we have defined

$$\begin{aligned} \int q_\theta^{(j)}(\boldsymbol{\theta}) \ln p(\mathcal{X}, \mathcal{X}^l, \boldsymbol{\theta}; \xi^{(j)}) d\boldsymbol{\theta} &= \mathbb{E}_{q_\theta^{(j)}} [\ln p(\mathcal{X}, \mathcal{X}^l, \boldsymbol{\theta}; \xi^{(j)})] \\ &:= \ln \tilde{p}(\mathcal{X}, \mathcal{X}^l; \xi^{(j)}). \end{aligned} \quad (13.8)$$

The negative Kullback-Leibler divergence in Eq. (13.7) is maximized if we set

$$q_{\mathcal{X}^l}^{(j+1)}(\mathcal{X}^l) \propto \tilde{p}(\mathcal{X}, \mathcal{X}^l; \xi^{(j)}). \quad (13.9)$$

Elaborating on Eqs. (13.8) and (13.9) and denoting all quantities that do not depend on \mathcal{X}^l as constants, we get

$$\ln q_{\mathcal{X}^l}^{(j+1)}(\mathcal{X}^l) = \mathbb{E}_{q_\theta^{(j)}} [\ln p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta}; \xi^{(j)})] + \text{constant}. \quad (13.10)$$

Hence, we can now write,

$$q_{\mathcal{X}^l}^{(j+1)}(\mathcal{X}^l) = \frac{\exp(\mathbb{E}_{q_\theta^{(j)}} [\ln p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta}; \xi^{(j)})])}{\int \exp(\mathbb{E}_{q_\theta^{(j)}} [\ln p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta}; \xi^{(j)})]) d\mathcal{X}^l}, \quad (13.11)$$

where the proportionality constant has necessarily been absorbed in the normalizing factor.

E-Step 1b: Freezing $\xi^{(j)}$ and $q_{\mathcal{X}^l}^{(j+1)}$ and following similar steps as before (repeat the steps as an exercise), starting from the formulation in Eq. (13.6) and maximizing with respect to q_θ , we obtain

$$q_\theta^{(j+1)}(\boldsymbol{\theta}) = \frac{p(\boldsymbol{\theta}; \xi^{(j)}) \exp(\mathbb{E}_{q_{\mathcal{X}^l}^{(j+1)}} [\ln p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta}; \xi^{(j)})])}{\int p(\boldsymbol{\theta}; \xi^{(j)}) \exp(\mathbb{E}_{q_{\mathcal{X}^l}^{(j+1)}} [\ln p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta}; \xi^{(j)})]) d\boldsymbol{\theta}}. \quad (13.12)$$

Steps 1a and 1b comprise the E-step of the variational Bayesian EM.

M-Step 2: Freezing $q_\theta^{(j+1)}$ and $q_{\mathcal{X}^l}^{(j+1)}$, maximize the lower bound with respect to ξ , that is,

$$\xi^{(j+1)} = \arg \max_{\xi} \mathcal{F}(q_\theta^{(j+1)}, q_{\mathcal{X}^l}^{(j+1)}; \xi). \quad (13.13)$$

The counterpart of the EM illustration of Figure 12.6 is given in Figure 13.1. There are two observations to be made. Step 1 is now split into two parts, and more important, the KL divergence does *not* (in

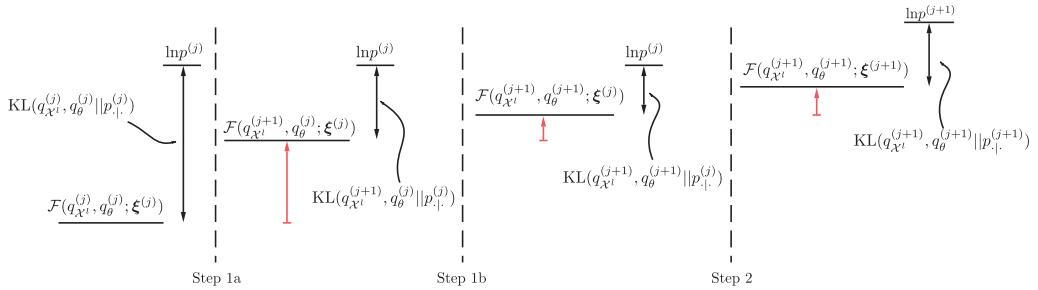

FIGURE 13.1

Illustration of the stepwise increase of $\ln p^{(j)}$ at the $(j+1)$ iteration of the Variational Bayesian EM algorithm. Observe that $\ln p^{(j+1)} > \ln p^{(j)}$, where we have used the notation $p^{(j)} = p(\mathcal{X}; \boldsymbol{\xi}^{(j)})$ and $p_{|\cdot|}^{(j)} := p(\mathcal{X}^l, \boldsymbol{\theta} | \mathcal{X}; \boldsymbol{\xi}^{(j)})$.

general) go to zero; hence the bound does not become tight. This comprises the E-step of the variational Bayesian EM.

If there are more than two factors in $q(\mathcal{X}^l, \boldsymbol{\theta})$, as in Eq. (13.4), then there are more than two substeps in step 1, and each time we estimate one of the factors by averaging $\ln p(\mathcal{X}, \mathcal{X}^l, \boldsymbol{\theta}; \boldsymbol{\xi})$ with respect to the rest. Let q be factorized in M factors

$$q(\mathcal{X}^l) = q_1(\mathcal{X}_1^l) \dots q_M(\mathcal{X}_M^l),$$

where for notational uniformity we have not differentiated between parameters and latent variables and the dependence on $\boldsymbol{\xi}$ has been suppressed. Then the general form of update becomes

$$\boxed{\ln q_m(\mathcal{X}_m^l) = \mathbb{E} \left[\ln p(\mathcal{X}, \mathcal{X}_1^l, \dots, \mathcal{X}_M^l) \right] + \text{constant}}, \quad (13.14)$$

where the expectation is with respect to $\prod_{r=1, r \neq m}^M q_r(\mathcal{X}_r^l)$.

Remarks 13.1.

- Note that $q(\mathcal{X}^l, \boldsymbol{\theta})$ is an estimate of the posterior $p(\mathcal{X}^l, \boldsymbol{\theta} | \mathcal{X})$ and each one of the factors is the respective posterior estimate given the observations \mathcal{X} , for example, $q_\theta(\boldsymbol{\theta}) \simeq p(\boldsymbol{\theta} | \mathcal{X})$.
- Once $q(\mathcal{X}^l, \boldsymbol{\theta})$ is factorized, *no additional assumptions on the functional form* of $q_{\mathcal{X}^l}$ and q_θ are made.
- Note that factorization of a pdf implies independence. Thus, if this is not the case for the data at hand, the recovered approximations may not be faithful representations of the underlying data structure. Hence, choosing a specific factorization has to be carried out with care. In practice, one may have to use a number of alternatives and keep the best one. However, computational complexity is the other face of the coin, which one must consider in a trade-off game. In general, the factorized variational approach tends to provide approximations to the posterior pdf that are more compact than the true ones (e.g., [40]).
- Recall our discussion in Section 12.3 related to model selection and Occam's rule. This was a kick-off point for our efforts to maximize the evidence with respect to different models, in order to achieve the complexity-accuracy tradeoff. However, having resorted to approximate solutions (even if we forget convergence to local maxima, which somehow can be bypassed by using different initializations) we do not maximize the evidence but a lower bound of it; the latter is not,

in general, tight. How tight it is depends on the Kullback-Leibler divergence, which, unfortunately, cannot be trivially computed. Hence, if the lower bound is used for model selection, it has to be treated with caution [6].

- The variational approximation to Bayesian inference was first proposed in [21] and later on was used in a number of areas ranging from machine learning to decoding (e.g., [6, 22, 27–29, 31]).
- *Online Versions:* An online version of the variational Bayes algorithm was first proposed in [63]. There, the exponential family has been employed to show that parameter updating via the variational Bayes philosophy is equivalent to a natural gradient descent method (see Section 8.12, for the natural gradient) with step-size equal to one. This equivalence is further discussed in [24], where, similarly, a stochastic approximation algorithm is proposed in order to process chunks of data in parallel. An online variational Bayes algorithm for parameter estimation in the context of sparse linear regression modeling has also been proposed in [73].

13.2.1 THE CASE OF THE EXPONENTIAL FAMILY OF PROBABILITY DISTRIBUTIONS

Looking carefully at Eqs. (13.11) and (13.12), it becomes clear that the practical application of the variational Bayesian EM depends on the computational tractability of the expected values of the $\ln p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta}; \xi)$. Let us now see the form that the iterative steps take when one adopts the pdf models from the exponential family.

Let us assume that the points in the complete data set $(\mathbf{x}_n, \mathbf{x}_n^l), n = 1, 2, \dots, N$, are i.i.d. Then,

$$p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta}) = \prod_{n=1}^N p(\mathbf{x}_n, \mathbf{x}_n^l | \boldsymbol{\theta}). \quad (13.15)$$

We further assume $p(\mathbf{x}_n, \mathbf{x}_n^l | \boldsymbol{\theta})$ to lie within the exponential family (Section 12.4), that is,

$$p(\mathbf{x}_n, \mathbf{x}_n^l | \boldsymbol{\theta}) = g(\boldsymbol{\theta}) f(\mathbf{x}_n, \mathbf{x}_n^l) \exp(\boldsymbol{\phi}^T(\boldsymbol{\theta}) \mathbf{u}(\mathbf{x}_n, \mathbf{x}_n^l)). \quad (13.16)$$

We also adopt a prior for $\boldsymbol{\theta}$ to be of the respective conjugate form, that is,

$$p(\boldsymbol{\theta} | \lambda, \mathbf{v}) = h(\lambda, \mathbf{v}) (g(\boldsymbol{\theta}))^\lambda \exp(\boldsymbol{\phi}^T(\boldsymbol{\theta}) \mathbf{v}). \quad (13.17)$$

The parameters λ, \mathbf{v} constitute ξ , which will be considered fixed, because our current emphasis is to follow up the specific functional forms that $q_{\mathcal{X}^l}$ and $q_{\boldsymbol{\theta}}$ get as iterations progress. So we relax the notational dependence on these parameters.

E-Step 1a: We have from Eq. (13.11) that

$$\begin{aligned} q_{\mathcal{X}^l}^{(j+1)}(\mathcal{X}^l) &\propto \exp\left(\mathbb{E}_{q_{\boldsymbol{\theta}}^{(j)}} [\ln p(\mathcal{X}, \mathcal{X}^l | \boldsymbol{\theta})]\right) \\ &= \exp\left(\mathbb{E}_{q_{\boldsymbol{\theta}}^{(j)}} \left[\sum_{n=1}^N \ln p(\mathbf{x}_n, \mathbf{x}_n^l | \boldsymbol{\theta}) \right]\right) \\ &= \prod_{n=1}^N \exp\left(\mathbb{E}_{q_{\boldsymbol{\theta}}^{(j)}} [\ln p(\mathbf{x}_n, \mathbf{x}_n^l | \boldsymbol{\theta})]\right), \end{aligned}$$

which then suggests that

$$q_{\mathbf{x}_n^l}^{(j+1)}(\mathbf{x}_n^l) \propto \exp\left(\mathbb{E}_{q_{\boldsymbol{\theta}}^{(j)}} [\ln p(\mathbf{x}_n, \mathbf{x}_n^l | \boldsymbol{\theta})]\right),$$

and combined with Eq. (13.16) results in

$$q_{\mathbf{x}_n^l}^{(j+1)}(\mathbf{x}_n^l) = \tilde{g}f(\mathbf{x}_n, \mathbf{x}_n^l) \exp\left(\tilde{\boldsymbol{\phi}}^T \mathbf{u}(\mathbf{x}_n, \mathbf{x}_n^l)\right),$$

where \tilde{g} is the respective normalization constant and

$$\tilde{\boldsymbol{\phi}}^T = \mathbb{E}_{q_\theta^{(j)}}[\boldsymbol{\phi}^T(\boldsymbol{\theta})]. \quad (13.18)$$

This is very interesting indeed. Although no functional form was assumed for $q_{\mathcal{X}^l}$, it turns out to be a member of the exponential family!

E-Step 1b: In a similar way, from (13.12), (13.15) and (13.16), we obtain

$$\begin{aligned} q_\theta^{(j+1)}(\boldsymbol{\theta}) &\propto p(\boldsymbol{\theta}) \exp\left(N \ln g(\boldsymbol{\theta}) + \sum_{n=1}^N \mathbb{E}_{q_{\mathbf{x}_n^l}^{(j+1)}} \left[\ln(f(\mathbf{x}_n, \mathbf{x}_n^l)) \right] \right. \\ &\quad \left. + \boldsymbol{\phi}^T(\boldsymbol{\theta}) \sum_{n=1}^N \mathbb{E}_{q_{\mathbf{x}_n^l}^{(j+1)}} \left[\mathbf{u}(\mathbf{x}_n, \mathbf{x}_n^l) \right] \right), \end{aligned}$$

which combined with Eq. (13.17) results in

$$q_\theta^{(j+1)}(\boldsymbol{\theta}) \propto (g(\boldsymbol{\theta}))^{\lambda+N} \exp\left(\boldsymbol{\phi}^T(\boldsymbol{\theta}) \left(v + \sum_{n=1}^N \mathbb{E}_{q_{\mathbf{x}_n^l}^{(j+1)}} \left[\mathbf{u}(\mathbf{x}_n, \mathbf{x}_n^l) \right]\right)\right). \quad (13.19)$$

Thus, the approximation $q_\theta^{(j+1)}(\boldsymbol{\theta})$ of the posterior $p(\boldsymbol{\theta}|\mathcal{X})$ is of the same form as the conjugate prior with

$$\tilde{\lambda} = \lambda + N, \quad \tilde{v} = v + \sum_{n=1}^N \mathbb{E}_{q_{\mathbf{x}_n^l}^{(j+1)}} \left[\mathbf{u}(\mathbf{x}_n, \mathbf{x}_n^l) \right]. \quad (13.20)$$

Note that Eq. (13.20) is of the same form as Eq. (12.45). We only have to average out the hidden variables. This is a very elegant result, because nothing has been assumed about the functional form of q_θ . In other words, once we adopt the functional form for the pdfs of the complete set as well as that of the prior of the parameters to be of the exponential type, then subsequent iterations become a “family business.”

13.3 A VARIATIONAL BAYESIAN APPROACH TO LINEAR REGRESSION

Once more, let us consider our familiar regression task

$$\mathbf{y} = \Phi\boldsymbol{\theta} + \boldsymbol{\eta}, \quad \mathbf{y} \in \mathbb{R}^N, \quad \boldsymbol{\theta} \in \mathbb{R}^K.$$

In Section 12.6, we treated the case where $\boldsymbol{\eta}$ was Gaussian and the prior $p(\boldsymbol{\theta})$ was also Gaussian. We used the EM in order to optimize the evidence $p(\mathbf{y})$ with respect to the parameters that define the two adopted Gaussian pdfs; note that for this case, one could bypass the EM and resort to analytical computations in order to obtain the evidence and subsequently use an optimization technique to estimate the unknown parameters.

In this section, we will adopt assumptions that do not allow for tractable analytic computations of the posterior, $p(\boldsymbol{\theta}|\mathbf{y})$, which is a prerequisite both for the standard EM as well as for the analytic computations of the evidence $p(\mathbf{y})$. This approach is far from a pedagogic toy and has strong practical flavor. We will develop the task in some detail, and the reader is advised to go through the computations,

because they are typical of what will be encountered in practice, once the variational Bayesian approach is chosen for addressing a task.

Assume that

$$p(\mathbf{y}|\boldsymbol{\theta}, \beta) = \mathcal{N}(\Phi\boldsymbol{\theta}, \beta^{-1}\mathbf{I}). \quad (13.21)$$

That is, the noise is Gaussian and for simplicity we have considered it to be white, $\Sigma_\eta = \sigma_\eta^2 \mathbf{I}$, and $\beta = \frac{1}{\sigma_\eta^2}$. In contrast to what we did in [Section 12.6](#), now we will be more democratic and give the freedom to each one of the parameter components, θ_k , to have a different variance, $\sigma_k^2 := \frac{1}{\alpha_k}$, $k = 0, 1, \dots, K - 1$. Moreover, we go one step further. The values of β and α_k , $k = 0, \dots, K - 1$, will not be treated as deterministic variables. We will also treat them as random ones, (β, α_k) , which will be assigned prior pdfs; these prior pdfs are in turn controlled by another set of hyperparameters. More specifically, our model, in addition to Eq. (13.21) comprises [7]

$$p(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \prod_{k=0}^{K-1} \mathcal{N}(\theta_k|0, \alpha_k^{-1}), \quad (13.22)$$

$$p(\boldsymbol{\alpha}) = \prod_{k=0}^{K-1} \text{Gamma}(\alpha_k|a, b), \quad (13.23)$$

and

$$p(\beta) = \text{Gamma}(\beta|c, d). \quad (13.24)$$

Note that the previous choice of the priors indicates our will to “play” the game within the exponential family terrain. The prior $p(\boldsymbol{\alpha})$ is the conjugate pair of Eq. (13.22) (see [Chapter 12](#)). Also, Eq. (13.24) would be the conjugate of Eq. (13.21), if we had considered $\boldsymbol{\theta}$ fixed. [Figure 13.2](#) provides a graphical

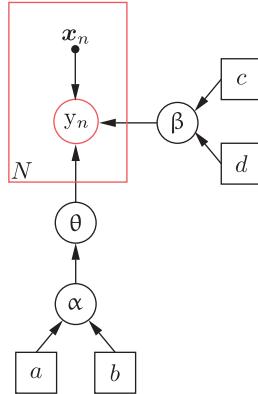


FIGURE 13.2

A graphical illustration of the dependencies among the various variables involved in the model of linear regression. The red circle indicates the random variable that is observed, gray circles indicate hidden random variables, and squares correspond to deterministic parameters. The direction of each arrow indicates the direction of the dependence between the connected variables. The red box indicates that the above dependencies hold for all, N , time instants.

representation of the dependencies among the various variables involved in our model. Arrows indicate conditional dependencies. Graphical models will be considered in a formal way in Chapter 15. Note that such a model forms various levels of *hierarchy* in the dependency among the involved parameters.

This concept of hierarchy is at the heart of what we call *hierarchical Bayesian modeling*. Each one of the involved pdfs is expressed in terms of certain parameters. Because the values of these parameters are unknown, they are also treated as random variables whose priors are expressed in terms of a new set of hyperparameters. Each one of them is in turn treated as a random variable associated with a new prior, known as *hyperprior*. This rationale can be extended in order to construct different levels of hierarchy. Often, at the higher level of hierarchy, the corresponding (unknown) hyperparameters are assigned values by the user, based on experience; for example, the overall model can be relatively insensitive to their specific values, which makes the corresponding choice a fairly easy job.

Our current task comprises hidden variables in the form of parameters grouped in θ , α , and β and it involves no other latent variables. The set of observations is now given by y . Also, observe that the posterior $p(\theta, \alpha, \beta | y)$ is not analytically tractable. We will resort to the variational Bayesian EM to obtain an estimate of the previous posterior pdf.

Using the mean field approximation, we assume that the approximation to the posterior (the dependence on y has been suppressed for notational convenience) factorizes as

$$q(\theta, \alpha, \beta) = q_\theta(\theta)q_\alpha(\alpha)q_\beta(\beta), \quad (13.25)$$

where we have relaxed our notation, for simplicity, from the explicit dependence on a , b , c , and d . We will bring them back into the game whenever needed. The variational EM consists of three substeps, one for each factor in Eq. (13.25). Starting from some initial guesses, for $\mathbb{E}[\beta]$, $\mathbb{E}[\alpha_k]$, $k = 0, \dots, K - 1$, (it will become clear soon why we need to start with those¹) we get:

E-Step 1a: From the general update form of Eq. (13.14) we have,

$$\ln q_\theta^{(j+1)}(\theta) = \mathbb{E}_{q_\alpha^{(j)} q_\beta^{(j)}} [\ln p(y, \theta, \alpha, \beta)] + \text{constant}, \quad (13.26)$$

where now

$$\begin{aligned} \ln p(y, \theta, \alpha, \beta) &= \ln(p(y|\theta, \alpha, \beta)p(\theta, \alpha, \beta)) \\ &= \ln(p(y|\theta, \beta)p(\theta|\alpha)p(\alpha)p(\beta)), \end{aligned} \quad (13.27)$$

where the independence of y on α , given the values θ , has been taken into account. Using Eqs. (13.21), (13.24) and some trivial algebra we get that

$$\begin{aligned} \ln p(y, \theta, \alpha, \beta) &= \ln \frac{\beta^{N/2}}{(2\pi)^{N/2}} - \frac{\beta}{2} \|y - \Phi\theta\|^2 - \frac{1}{2} \sum_{k=0}^{K-1} a_k \theta_k^2 \\ &\quad + \sum_{k=0}^{K-1} \ln \sqrt{\frac{\alpha_k}{2\pi}} + \ln p(\alpha) + \ln p(\beta) \end{aligned}$$

or

$$\ln p(y, \theta, \alpha, \beta) = -\frac{\beta}{2} \|y - \Phi\theta\|^2 - \frac{1}{2} \sum_{k=0}^{K-1} a_k \theta_k^2 + \text{constant}, \quad (13.28)$$

¹ If a, b, c, d were not fixed, then one would need initialization for these parameters, too.

where constant includes all terms that do not depend on θ , because in this step our goal is to estimate a function of θ . Expanding Eq. (13.28) and taking expectations w.r. to β and α , considering $q_\beta^{(j)}(\beta)$ and $q_\alpha^{(j)}(\alpha)$ known, we get

$$\begin{aligned}\ln q_\theta^{(j+1)}(\theta) &= \mathbb{E}_{q_\beta^{(j)} q_\alpha^{(j)}} [\ln p(y, \theta, \alpha, \beta)] + \text{constant} = -\frac{1}{2} \mathbb{E}[\beta] \theta^T \Phi^T \Phi \theta \\ &\quad - \frac{1}{2} \mathbb{E}[\beta] y^T y + \mathbb{E}[\beta] \theta^T \Phi^T y - \frac{1}{2} \theta^T A \theta + \text{constant},\end{aligned}\quad (13.29)$$

where by definition

$$A := \text{diag}\{\mathbb{E}[\alpha_0], \dots, \mathbb{E}[\alpha_{K-1}]\},$$

and we have used for notational simplifications

$$\mathbb{E}[\beta] := E_{q_\beta^{(j)}}[\beta] \quad \text{and} \quad \mathbb{E}[\alpha_k] := E_{q_\alpha^{(j)}}[\alpha_k], \quad k = 0, 1, 2, \dots, K-1. \quad (13.30)$$

It is readily noticed that the right-hand side of Eq. (13.29) is of a quadratic form with respect to θ , hence $q_\theta^{(j+1)}(\theta)$ is Gaussian; in order to completely specify it, it suffices to compute the respective mean and covariance (precision) matrix.

Reshuffling the terms in Eq. (13.29), we get

$$\ln q_\theta^{(j+1)}(\theta) = -\frac{1}{2} \theta^T (A + \mathbb{E}[\beta] \Phi^T \Phi) \theta + \mathbb{E}[\beta] \theta^T \Phi^T y + \text{constant},$$

which according to Eqs. (12.111), (12.113) and (12.114) of Section 12.9 of the previous chapter results in

$$q_\theta^{(j+1)}(\theta) = \mathcal{N}(\theta | \mu_\theta^{(j+1)}, \Sigma_\theta^{(j+1)}), \quad (13.31)$$

$$\Sigma_\theta^{(j+1)} = (A + \mathbb{E}[\beta] \Phi^T \Phi)^{-1}, \quad (13.32)$$

and

$$\mu_\theta^{(j+1)} = \mathbb{E}[\beta] \Sigma_\theta^{(j+1)} \Phi^T y. \quad (13.33)$$

During the first iteration step, $\mathbb{E}[\beta]$ and $\mathbb{E}[\alpha_k]$ are provided by their initial values. For the subsequent iterations, they have to be obtained together with $q_\beta^{(j)}$ and $q_\alpha^{(j)}$. Note that the approximation to the posterior $p(\theta|y)$ turns out to be Gaussian, although we did not assume it to be so. This is a consequence of the particular form of the adopted pdfs, which spring from the exponential family.

E-Step 1b:

$$\ln q_\alpha^{(j+1)}(\alpha) = \mathbb{E}_{q_\theta^{(j+1)} q_\beta^{(j)}} [\ln p(y, \theta, \alpha, \beta)] + \text{constant} \quad (13.34)$$

$$= \mathbb{E}_{q_\theta^{(j+1)} q_\beta^{(j)}} [\ln p(\theta|\alpha) + \ln p(\alpha)] + \text{constant}, \quad (13.35)$$

where the constant contains all terms that do not depend on α . Because no term in the bracket in the right-hand side of Eq. (13.35) depends on β , we have

$$\ln q_\alpha^{(j+1)}(\alpha) = E_{q_\theta^{(j+1)}} \left[\frac{1}{2} \sum_{k=0}^{K-1} \ln \alpha_k - \frac{1}{2} \sum_{k=0}^{K-1} \alpha_k \theta_k^2 \right] + \ln p(\alpha) + \text{constant}. \quad (13.36)$$

Taking into account Eq. (13.23) and after some algebra (Problem 13.2), we obtain

$$q_{\alpha}^{(j+1)}(\boldsymbol{\alpha}) = \prod_{k=0}^{K-1} \text{Gamma}(\alpha_k | \tilde{a}, \tilde{b}_k), \quad (13.37)$$

where

$$\tilde{a} = a + \frac{1}{2} \quad (13.38)$$

$$\tilde{b}_k = b + \frac{1}{2} \mathbb{E}_{q_{\theta}^{(j+1)}}[\theta_k^2], \quad k = 0, \dots, K-1. \quad (13.39)$$

In order to compute $\mathbb{E}[\theta_k^2]$, recall Eqs. (12.73) and (12.74) and apply them into our setting to give

$$\mathbb{E}_{q_{\theta}^{(j+1)}}[\boldsymbol{\theta}\boldsymbol{\theta}^T] = \Sigma_{\theta}^{(j+1)} + \boldsymbol{\mu}_{\theta}^{(j+1)}\boldsymbol{\mu}_{\theta}^{(j+1)T},$$

or

$$\mathbb{E}[\theta_k^2] = [E_{q_{\theta}^{(j+1)}}[\boldsymbol{\theta}\boldsymbol{\theta}^T]]_{kk} = [\Sigma_{\theta}^{(j+1)} + \boldsymbol{\mu}_{\theta}^{(j+1)}\boldsymbol{\mu}_{\theta}^{(j+1)T}]_{kk}, \quad k = 0, 1, \dots, K-1, \quad (13.40)$$

where $[A]_{kk}$ denotes the (k, k) element of a matrix, A . To complete the computations, we have to compute $\mathbb{E}[\alpha_k], k = 0, 1, \dots, K-1$, to be used during the next iteration in Eq. (13.32). However, because each α_k follows a gamma distribution, we know that (Section 2.3.2)

$$\mathbb{E}_{q_{\alpha}^{(j+1)}}[\alpha_k] = \frac{\tilde{a}}{\tilde{b}_k}. \quad (13.41)$$

E-Step 1c:

$$\begin{aligned} \ln q_{\beta}^{(j+1)}(\beta) &= \mathbb{E}_{q_{\theta}^{(j+1)} q_{\alpha}^{(j+1)}} [\ln p(\mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\alpha}, \beta)] + \text{constant} \\ &= \mathbb{E}_{q_{\theta}^{(j+1)} q_{\alpha}^{(j+1)}} [\ln p(\mathbf{y}|\boldsymbol{\theta}, \beta) + \ln p(\beta)] + \text{constant}. \end{aligned}$$

This is of the same form as Eq. (13.35), and following similar steps (Problem 13.3), it can be shown that

$$q_{\beta}^{(j+1)}(\beta) = \text{Gamma}(\beta | \tilde{c}, \tilde{d}), \quad (13.42)$$

$$\tilde{c} = c + \frac{N}{2}, \quad (13.43)$$

$$\tilde{d} = d + \frac{1}{2} \mathbb{E}_{q_{\theta}^{(j+1)}}[\|\mathbf{y} - \Phi\boldsymbol{\theta}\|^2]. \quad (13.44)$$

To compute the expectation in Eq. (13.44), recall Eq. (12.76), which for our needs becomes

$$\mathbb{E}_{q_{\theta}^{(j+1)}}[\|\mathbf{y} - \Phi\boldsymbol{\theta}\|^2] = \|\mathbf{y} - \Phi\boldsymbol{\mu}_{\theta}^{(j+1)}\|^2 + \text{trace} \left\{ \Phi \Sigma_{\theta}^{(j+1)} \Phi^T \right\}. \quad (13.45)$$

Finally, we have that

$$\mathbb{E}_{q_{\beta}^{(j+1)}}[\beta] = \frac{\tilde{c}}{\tilde{d}}, \quad (13.46)$$

which completes all the computations associated with the E-step of the variational EM. Note that $q_{\alpha}^{(j+1)}(\boldsymbol{\alpha}) \simeq p(\boldsymbol{\alpha}|\mathbf{y})$ and $q_{\beta}^{(j+1)}(\beta) \simeq p(\beta|\mathbf{y})$ retain the gamma functional form of the corresponding priors that were originally adopted, without forcing them to.

In principle, one can add an extra M-step in the algorithm to maximize the bound with respect to the unknown parameters a , b , c , and d . However, in practice, for computational simplicity these parameters are fixed to very small values, that is, $a = b = c = d = 10^{-6}$, which correspond to *uninformative* gamma prior distributions, in the sense of giving no preference to any specific range of values. Note that for such small values, the gamma distribution falls as $\frac{1}{x}$. Indeed, for $a, b \simeq 0$

$$\text{Gamma}(x|a, b) \simeq \frac{1}{x}, \quad x > 0.$$

Because every positive x can be expressed as

$$x = \exp(z), \quad z = \ln x, \quad z \in \mathbb{R},$$

then it can be easily checked out ([Problem 13.4](#)) that the pdf that describes z is uniform. This is a typical procedure in practice; that is, one allows enough levels of hierarchy and fixes the hyperparameters in the highest level to define uninformative hyperpriors.

In summary, the variational Bayesian EM steps are given in [Algorithm 13.1](#).

Algorithm 13.1 (Variational EM for linear regression).

- Initialization
 - Select initial values for $\mathbb{E}[\beta]$, $\mathbb{E}_{q_\alpha}[\alpha_k]$, $k = 0, 1, \dots, K - 1$.
- **For**, $j = 1, 2, \dots$, **Do**
 - $\mathbf{A} = \text{diag}\{\mathbb{E}_{q_\alpha}[\alpha_0], \mathbb{E}_{q_\alpha}[\alpha_1], \dots, \mathbb{E}_{q_\alpha}[\alpha_{K-1}]\}$.
 - Compute Σ_θ from Eq. [\(13.32\)](#) and μ_θ from Eq. [\(13.33\)](#).
 - Compute \tilde{a} from Eq. [\(13.38\)](#).
 - Compute \tilde{b}_k , $k = 0, 1, \dots, K - 1$, from Eqs. [\(13.40\)](#) and [\(13.39\)](#).
 - Compute $\mathbb{E}_{q_\alpha}[\alpha_k]$, $k = 0, 1, \dots, K - 1$, from Eq. [\(13.41\)](#).
 - Compute \tilde{c} from Eq. [\(13.43\)](#) and \tilde{d} from Eqs. [\(13.45\)](#) and [\(13.44\)](#).
 - Compute $\mathbb{E}_{q_\beta}[\beta]$ from Eq. [\(13.46\)](#).
 - If convergence criterion is met, Stop.
- **End For**

Once the algorithm has converged, predictions can be made on the basis of the predictive distribution given in Eqs. [\(12.21\)](#)–[\(12.23\)](#), by replacing $\Sigma_{\theta|y}$, $\mu_{\theta|y}$ and σ_η^2 by the converged values of Σ_θ , μ_θ , and $\mathbb{E}[\beta]$, respectively. Note, however, that this is only an approximation, because the Gaussian form for the posterior of the parameters is a result of the mean field approximation and also we have used the mean value, $\mathbb{E}[\beta]$, in place of the noise variance. The latter can be justified that as the number of training samples increases, the distribution of β sharply peaks around its mean value, [\[7\]](#).

Computation of the lower bound

Once the algorithm has converged, the quantities $q_\theta(\theta)$, $q_\alpha(\alpha)$, $q_\beta(\beta)$ are available and the lower bound $\mathcal{F}(q_\theta, q_\alpha, q_\beta)$ can be computed. The computation of this lower bound can also be done at every iteration to check how much it changes from iteration to iteration, and then this can be used as a convergence criterion. Let $\tilde{q}_\theta, \tilde{q}_\alpha, \tilde{q}_\beta$, be the approximate posteriors after convergence, defined by the parameters $\tilde{\Sigma}_\theta, \tilde{\mu}_\theta, \tilde{a}, \tilde{b}_k, k = 0, 1, 2, \dots, K - 1, \tilde{c}$, and \tilde{d} . The lower bound is then given as

$$\begin{aligned}\mathcal{F}(\tilde{q}_\theta, \tilde{q}_\alpha, \tilde{q}_\beta) &= \mathbb{E}_{\tilde{q}_\theta \tilde{q}_\alpha \tilde{q}_\beta} [\ln p(\mathbf{y}, \theta, \alpha, \beta)] - \mathbb{E}_{\tilde{q}_\theta} [\ln \tilde{q}_\theta(\theta)] \\ &\quad - \mathbb{E}_{\tilde{q}_\alpha} [\ln \tilde{q}_\alpha(\alpha)] - \mathbb{E}_{\tilde{q}_\beta} [\ln \tilde{q}_\beta(\beta)],\end{aligned}\quad (13.47)$$

Performing the expectations can be a bit tedious, but it is straightforward (Problem 13.5).

13.4 A VARIATIONAL BAYESIAN APPROACH TO GAUSSIAN MIXTURE MODELING

Dealing with the Gaussian mixture modeling in Section 12.7, it was pointed out in the remarks that the standard EM approach may lead to singularities. One way to bypass this drawback is to enforce priors on the involved parameters and resort to a variational Bayesian philosophy to estimate the quantities of interest. The task was first treated in [3] and later in [11]. We will present the latter approach and comment on the underlying differences between the two later on.

Given a set of observations, $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, the respective pdf model is

$$p(\mathbf{x}) = \sum_{k=1}^K P_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, Q_k^{-1}), \quad \mathbf{x} \in \mathbb{R}^l.$$

The task is to estimate the unknown parameters $(P_k, \boldsymbol{\mu}_k, Q_k)$, $k = 1, 2, \dots, K$. We already know that this is a typical task with latent variables, and the complete set comprises (\mathbf{x}_n, k_n) , $n = 1, 2, \dots, N$, with k_n being the index of the respective mixture, $k_n = 1, 2, \dots, K$. In section 12.7, the information about each one of the latent variables, k_n , entered into the problem via the posterior $P(k_n | \mathbf{x}_n)$ for every time instant n , the summation over all possible values of k_n was performed, hence one could drop out the time index. However, in the current context, a different path has to be followed and we have to consider the latent variables together with their corresponding time index. To this end, and following [11], an auxiliary latent random vector is introduced, $\mathbf{z}_n \in \mathbb{R}^K$, for each observation, $n = 1, 2, \dots, N$. Its components take binary values, such as

$$z_{nk} \in \{0, 1\}, \quad \text{and} \quad \sum_{k=1}^K z_{nk} = 1, \quad (13.48)$$

and they are used as indicators of the respective mixture from which the observation at time n , \mathbf{x}_n , was drawn; that is, if $z_{nk} = 1$ it indicates that \mathbf{x}_n was drawn from the k th distribution. Obviously,

$$P(z_{nk} = 1) = P_k,$$

and for any $\mathbf{z}_n \in \mathbb{R}^K$ that satisfies Eq. (13.48),

$$P(\mathbf{z}_n) = \prod_{k=1}^K P_k^{z_{nk}}. \quad (13.49)$$

Hence, the probability of occurrence of the set $\mathcal{Z} = \{\mathbf{z}_1, \dots, \mathbf{z}_N\}$ is

$$P(\mathcal{Z}) = \prod_{n=1}^N \prod_{k=1}^K P_k^{z_{nk}}, \quad (13.50)$$

and in this way, we have described the random nature of the N latent variables using a multinomial probability distribution.

In the sequel, we will treat the mean values as well as the precision matrices as random quantities adopting the following prior pdfs,

$$p(\boldsymbol{\mu}_k) = \mathcal{N}(\boldsymbol{\mu}_k | 0, \beta^{-1}I)$$

and

$$p(Q_k) = \mathcal{W}(Q_k | W_0, v_0),$$

for fixed v_0 , W_0 , and β . That is, the adopted priors are Gaussian for the mean values and Wishart pdfs for the precision matrices, respectively. We will treat $\mathbf{P} = [P_1, \dots, P_K]^T$ as deterministic parameters whose optimized value is obtained in the M-step.

Following the philosophy of the variational Bayesian EM, we adopt

$$q(\mathcal{Z}, \boldsymbol{\mu}_{1:K}, Q_{1:K}) = q_{\mathcal{Z}}(\mathcal{Z})q_{\boldsymbol{\mu}}(\boldsymbol{\mu}_{1:K})q_Q(Q_{1:K}),$$

where $\boldsymbol{\mu}_{1:K}$ and $Q_{1:K}$ indicate the collections $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K\}$ and $\{Q_1, \dots, Q_K\}$, respectively.

Furthermore, observe that the conditional pdf of the observations can now be written as

$$p(\mathcal{X} | \mathcal{Z}, \boldsymbol{\mu}_{1:K}, Q_{1:K}) = \prod_{n=1}^N \prod_{k=1}^K \left(\mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, Q_k^{-1}) \right)^{z_{nk}}.$$

[Figure 13.3](#) shows the corresponding graphical model.

Computational Steps of The Variational EM for Gaussian Mixture Modeling

Initialization: (a) $\mathbf{P}^{(0)}$, (b) $\mathbb{E}_{q_Q^{(0)}}[Q_k]$, (c) $\mathbb{E}_{q_Q^{(0)}}[\ln |Q_k|]$, (d) $\mathbb{E}_{q_{\boldsymbol{\mu}}^{(0)}}[\boldsymbol{\mu}_k] := \tilde{\boldsymbol{\mu}}_k^{(0)}$, and (e) $\mathbb{E}_{q_{\boldsymbol{\mu}}^{(0)}}[\boldsymbol{\mu}_k \boldsymbol{\mu}_k^T] := \tilde{\Sigma}_k^{(0)} + \tilde{\boldsymbol{\mu}}_k^{(0)} \tilde{\boldsymbol{\mu}}_k^{(0)T}$, $k = 1, 2, \dots, K$, where $|\cdot|$ denotes the corresponding determinant.

The $(j+1)$ iteration consists of the following computations ([Problem 13.6](#)):

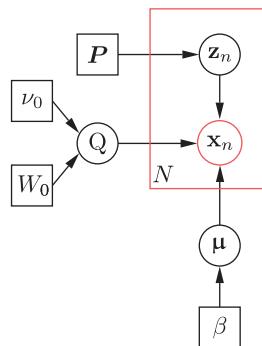


FIGURE 13.3

The graphical model associated with the Gaussian mixture modeling of [Section 13.4](#).

E-Step 1a:

$$\begin{aligned}\pi_{n_k} &= P_k^{(j)} \exp \left(\frac{1}{2} \mathbb{E}_{q_Q^{(j)}} [\ln |Q_k|] - \frac{1}{2} \text{trace} \{ \mathbb{E}_{q_Q^{(j)}} [Q_k] (\mathbf{x}_n \mathbf{x}_n^T - \mathbf{x}_n \mathbb{E}_{q_\mu^{(j)}} [\boldsymbol{\mu}_k^T] - \mathbb{E}_{q_\mu^{(j)}} [\boldsymbol{\mu}_k] \mathbf{x}_n^T + \mathbb{E}_{q_\mu^{(j)}} [\boldsymbol{\mu}_k \boldsymbol{\mu}_k^T]) \} \right) \\ \rho_{n_k} &= \frac{\pi_{n_k}}{\sum_{k=1}^K \pi_{n_k}} \\ q_z^{(j+1)}(\mathcal{Z}) &= \prod_{n=1}^N \prod_{k=1}^K \rho_{n_k}^{z_{n_k}}.\end{aligned}$$

E-Step 1b:

$$\begin{aligned}\tilde{Q}_k &= \beta I + \mathbb{E}_{q_Q^{(j)}} [Q_k] \sum_{n=1}^N \rho_{n_k} \\ \tilde{\boldsymbol{\mu}}_k &= \tilde{Q}_k^{-1} \mathbb{E}_{q_Q^{(j)}} [Q_k] \sum_{n=1}^N \rho_{n_k} \mathbf{x}_n \\ q_\mu^{(j+1)}(\boldsymbol{\mu}_{1:K}) &= \prod_{k=1}^K (\boldsymbol{\mu}_k | \tilde{\boldsymbol{\mu}}_k, \tilde{Q}_k^{-1}),\end{aligned}$$

and adopting Eq. (12.74) to the current needs,

$$\mathbb{E}_{q_\mu^{(j+1)}} [\boldsymbol{\mu}_k \boldsymbol{\mu}_k^T] = \tilde{\Sigma}_k + \tilde{\boldsymbol{\mu}}_k \tilde{\boldsymbol{\mu}}_k^T = \tilde{Q}_k^{-1} + \tilde{\boldsymbol{\mu}}_k \tilde{\boldsymbol{\mu}}_k^T.$$

E-Step 1c:

$$\begin{aligned}\tilde{v}_k &= v + \sum_{n=1}^N \rho_{n_k} \\ \tilde{W}_k^{-1} &= \tilde{W}_0^{-1} + \sum_{n=1}^N \rho_{n_k} \left(\mathbf{x}_n \mathbf{x}_n^T - \tilde{\boldsymbol{\mu}}_k \mathbf{x}_n^T - \mathbf{x}_n \tilde{\boldsymbol{\mu}}_k^T + \mathbb{E}_{q_\mu^{(j+1)}} [\boldsymbol{\mu}_k \boldsymbol{\mu}_k^T] \right) \\ q_Q^{(j+1)}(Q_{1:K}) &= \prod_{k=1}^K \mathcal{W}(Q_k | \tilde{v}_k, \tilde{W}_k) \\ \mathbb{E}_{q_Q^{(j+1)}} [Q_k] &= \tilde{v}_k \tilde{W}_k \\ \mathbb{E}_{q_Q^{(j+1)}} [\ln |Q_k|] &= \sum_{i=1}^l \psi \left(\frac{\tilde{v}_{k+1-i}}{2} \right) + l \ln 2 + \ln |\tilde{W}_k|\end{aligned}$$

where $\psi(\cdot)$ is the *digamma* function, defined as

$$\psi(a) := \frac{d \ln \Gamma(a)}{da},$$

and the gamma function has been defined in Eq. (2.91).

M-Step 2:

$$P_k^{(j+1)} = \frac{1}{N} \sum_{n=1}^N \rho_{nk}.$$

The previous steps have concluded the algorithm. Observe that the iterations retain the functional form of the pdfs that were adopted for the respective priors; this is a consequence of their exponential family origin. In [11], it is suggested that this procedure can also be used to determine the number of mixtures, instead of adopting a cross-validation technique, as pointed out in the remarks of [Section 12.7](#). By adopting a large enough value for K , the probabilities P_k associated with the irrelevant components will be driven to zero during the M-step. Note that such a modeling is possible in the Bayesian framework, because it automatically achieves a trade-off between model complexity and data fitting. In [3], the probabilities P_k , $k = 1, 2, \dots, K$, were considered as random variables and a Dirichlet prior was also imposed on them ([Problem 13.7](#)). However, such priors need to be selected with some care, otherwise it may affect the sparsification potential of the algorithm (e.g., [8]).

Example 13.1. The purpose of this example is to demonstrate the power of the variational Bayesian method for mixture modeling compared to the more classical EM algorithm, which was discussed in [Section 12.7](#). Five clusters of data were generated using a corresponding number of Gaussians, as shown in [Figure 13.4](#). The parameters used for each one of these Gaussians were

$$\boldsymbol{\mu}_1 = [-2.5, 2.5]^T, \quad \boldsymbol{\mu}_2 = [-4.0, -2.0]^T, \quad \boldsymbol{\mu}_3 = [2.0, -1.0]^T,$$

$$\boldsymbol{\mu}_4 = [0.1, 0.2]^T, \quad \boldsymbol{\mu}_5 = [3.0, 3.0]^T,$$

and

$$\Sigma_1 = \begin{bmatrix} 0.5 & 0.081 \\ 0.081 & 0.7 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 0.4 & 0.02 \\ 0.002 & 0.3 \end{bmatrix}, \quad \Sigma_3 = \begin{bmatrix} 0.6 & 0.531 \\ 0.531 & 0.9 \end{bmatrix},$$

$$\Sigma_4 = \begin{bmatrix} 0.5 & 0.22 \\ 0.22 & 0.8 \end{bmatrix}, \quad \Sigma_5 = \begin{bmatrix} 0.88 & 0.2 \\ 0.2 & 0.22 \end{bmatrix}.$$

Prior to running the algorithms, we assumed that we do not know the exact number of mixtures, so a number of $K = 25$ clusters was used, that is, a much larger number than the true one.

For the EM algorithm, the initial mean values were generated randomly, using a Gaussian $\mathcal{N}(\boldsymbol{\mu} | \mathbf{0}, \mathbf{I})$ and the respective initial covariance matrices, $\Sigma_k^{(0)}$, $k = 1, 2, \dots, 25$, with random elements, making sure that it is positive definite. One way to achieve this is to generate a matrix Φ with random elements from a $\mathcal{N}(0, 1)$ and then form $\Phi^T \Phi$. Another possibility is to start with a diagonal matrix, for example, the identity one \mathbf{I} .

For the Variational EM algorithm, the following initial values were used: the mean values, $\tilde{\boldsymbol{\mu}}_k^{(0)}$ and the initial covariance matrices, $\tilde{\Sigma}_k^{(0)}$, $k = 1, 2, \dots, 25$, were generated as before. Also, $\mathbb{E}_{q_Q^{(0)}}[Q_k] = I$, $\mathbb{E}_{q_Q^{(0)}}[\ln |Q_k|] = 1$. In both cases, the initial probabilities were set to be equal.

Observe that the variational EM identifies the five clusters associated with the data; the rest of the mixtures correspond to zero probability weights. In contrast, the EM algorithm tries to identify all 25 mixtures and the result is not satisfactory.

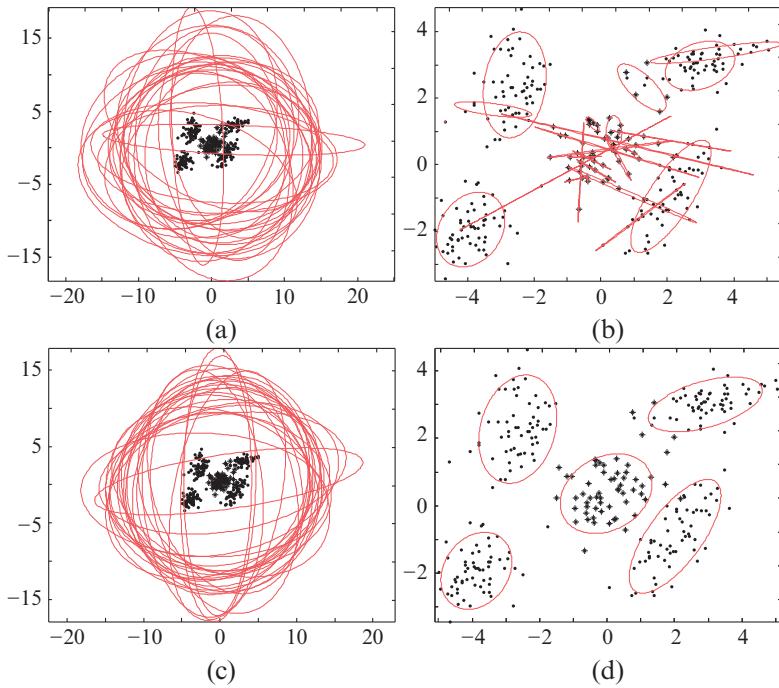
**FIGURE 13.4**

Figure for [Example 13.1](#): (a) The initial (25) Gaussians for the EM algorithm. (b) The final clusters obtained after convergence by the EM algorithm. (c) The initial (25) Gaussians for the variational EM. (d) The final Gaussians obtained by the variational EM, after convergence. All the curves correspond to the 80% probability regions. Observe that the variational EM identifies the five clusters associated with the data; the rest of the mixtures correspond to zero probability weights.

13.5 WHEN BAYESIAN INFERENCE MEETS SPARSITY

The Bayesian approach to sparsity-aware learning will soon become our major concern. However, we will use this subsection to “warm” us up. The close relationship between the use of a prior pdf and the regularization of a cost function has already been discussed in [Section 12.2.2](#). There, the adoption of a Gaussian prior together with a Gaussian noise for the regression task led to the equivalence of MAP with the ridge regression. It will not take a minute to show that the use of a Gaussian model for the noise together with a Laplacian prior for each one of the weights, that is,

$$p(\theta_k) = \frac{\lambda}{2} \exp(-\lambda|\theta_k|),$$

renders MAP equivalent to the ℓ_1 norm regularization of the LS cost. For a Bayesian, however, who is not interested in cost functions, the secret that lies within the Laplacian prior is hidden in the heavy tails of this distribution. This is in contrast to a Gaussian pdf, which has very light tails. In other words, the probability that an observation of a Gaussian random variable can take values far from its mean

decreases very fast. For example, the probability of observing variables that deviate from the mean by more than 2σ , 3σ , 4σ , and 5σ are 0.046, 0.003, 6×10^{-5} , and 6×10^{-7} , respectively. That is, if we provide a Gaussian prior, we basically inform the learning process to look for values “around” the mean; values away from the mean are heavily penalized. However, in sparsity-aware learning this would be the wrong information to pass over to our learning mechanism. Assuming the mean of the prior to be zero, although we expect most of the components of our parameters to be zero, still we want a few of them to be large. Hence, our prior information should be selected so as to assign small (but not too small) probabilities to large values. Hence, to a Bayesian, sparsity-aware learning becomes synonymous with imposing heavy-tail priors. Let us now turn back to our current task, and see how this brief introduction is related to our model. Our prior pdf, $p(\theta)$, according to the model Eqs. (13.22)–(13.23) is obtained by marginalizing out the hyperparameters α (Problem 13.10), that is,

$$\begin{aligned} p(\theta; a, b) &= \int p(\theta|\alpha)p(\alpha) d\alpha \\ &= \int \prod_{k=0}^{K-1} \mathcal{N}(\theta_k|0, \alpha_k^{-1}) \text{Gamma}(\alpha_k|a, b) d\alpha \\ &= \prod_{k=0}^{K-1} \text{st}\left(\theta_k|0, \frac{a}{b}, 2a\right), \end{aligned} \quad (13.51)$$

where $\text{st}(x|\mu, \lambda, \nu)$ is the student’s-t pdf, defined by

$$\text{st}(x|\mu, \lambda, \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})} \left(\frac{\lambda}{\pi\nu}\right)^{1/2} \frac{1}{\left(1 + \frac{\lambda(x-\mu)^2}{\nu}\right)^{\frac{\nu+1}{2}}}. \quad (13.52)$$

The parameter ν is known as the number of degrees of freedom. Figure 13.5 shows the graph of student’s-t pdfs for different values of ν . For $\nu \rightarrow \infty$, the student’s-t distribution tends to a Gaussian of the same mean and precision λ . Observe the heavy-tail feature of student’s-t pdf, especially for low

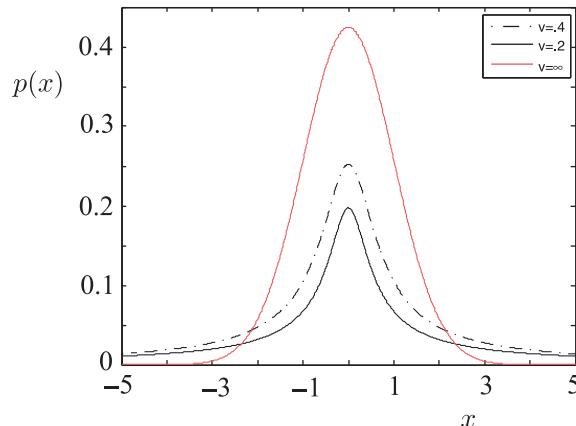


FIGURE 13.5

Observe that for low values of the degrees of freedom, ν , student’s-t pdf has very high tails. In contrast, the Gaussian pdf is a low-tailed pdf.

values of v . Recall that in our case, where we have used uninformative hyperpriors, the hyperparameter, a , was given a small value. Thus, our treatment in this section favors sparse solutions for the regression model. It will push as many of the coefficients, θ_k , as possible toward zero. That is, it prunes the less relevant basis functions, $\phi_k(\mathbf{x})$, by setting the corresponding coefficients to zero. This is also the reason for using different hyperparameters, α_k , for each one of the parameters, $\theta_k, k = 0, 1, \dots, K - 1$, which provide more freedom to the learning procedure to adjust each one of the parameters individually. In the earlier days, this approach was coined *Automatic Relevance Determination* (ARD) [41, 46, 48]. An interesting discussion relating adaptive regularization and pruning is provided in [20].

Figure 13.6a provides a clear demonstration of the sparsity-imposing properties of the student's-t distribution. In the two-dimensional space, and as we move away from zero, probability mass is skewed toward the coordinate axes; that is, the pdf peaks around sparse solutions and *sparsity is now enforced probabilistically*. In contrast, the Gaussian does not give much chance to large values; see Figure 13.6b.

13.6 SPARSE BAYESIAN LEARNING (SBL)

In Section 13.3, the prior for each one of the unknown parameters, $\theta_k, k = 0, 1, \dots, K - 1$, were given the liberty to have their own variances, $\sigma_k^2 := \frac{1}{\alpha_k}$. In turn, these variances were treated as hidden random variables and a prior was assigned to each of them in terms of a number of hyperparameters.

In [75, 81], the model was slightly modified. The concept of using different variances for the priors was retained, but the variances were treated as deterministic parameters and not as random ones.² In this context, the task becomes a generalization of the one treated in Section 12.6, and it is built upon the following assumptions:

$$p(\mathbf{y}|\boldsymbol{\theta}; \beta) = \mathcal{N}(\mathbf{y}|\Phi\boldsymbol{\theta}, \beta^{-1}\mathbf{I}), \quad (13.53)$$

$$p(\boldsymbol{\theta}; \boldsymbol{\alpha}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \mathbf{A}^{-1}), \quad (13.54)$$

where

$$\mathbf{A} := \text{diag}\{\alpha_0, \dots, \alpha_{K-1}\}. \quad (13.55)$$

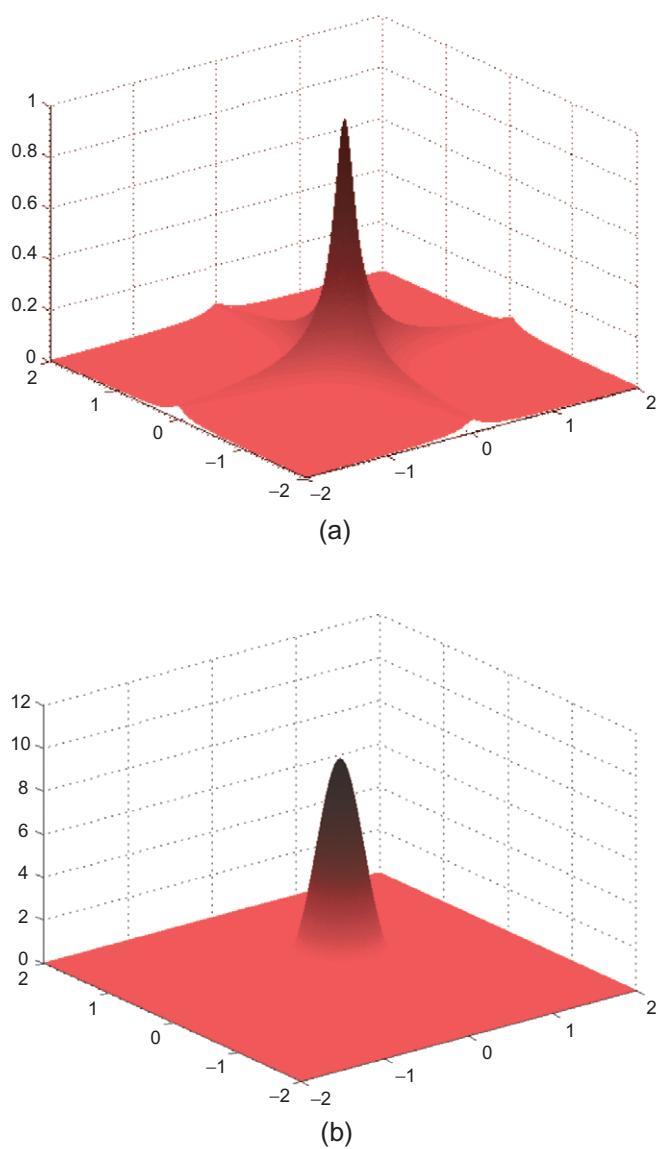
The precision β is also treated as an unknown deterministic parameter. Our goal is (a) to obtain estimates for β and $\alpha_k, k = 0, 1, \dots, K - 1$, and (b) to compute the predictive distribution, $p(\mathbf{y}|\mathbf{x}, \mathbf{y})$, where \mathbf{y} is the vector of observations.³ To this end, one could adopt the EM algorithm and follow similar steps as in Section 12.6; the only difference is that there, a common variance was shared by all the involved prior pdfs. The method is usually referred to as *sparse Bayesian learning* (SBL) and complies with the ARD rationale discussed in Section 13.5.

In this section, we will adopt a different path, exploiting the Gaussian nature of the involved pdfs. A Type II maximum likelihood method will be employed, which was introduced in Remarks 12.2. Type II likelihood is defined as the marginal one, after integrating out the parameters, $\boldsymbol{\theta}$. Following the discussion in Section 12.2 and for our current needs, Eq. (12.15) is written as

$$p(\mathbf{y}; \boldsymbol{\alpha}, \beta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \beta^{-1}\mathbf{I} + \Phi\mathbf{A}^{-1}\Phi^T). \quad (13.56)$$

² A slightly different yet equivalent view, employing uniform priors and using the respective modes instead of marginalizing out the variances, is followed in [75].

³ Notational dependence on the input training data, \mathcal{X} , has been suppressed.

**FIGURE 13.6**

(a) The student's-t peaks sharply around zero and falls slowly along the axes; hence, sparse solutions are favored. (b) The Gaussian peaks around zero and decays very fast, along all directions.

Also, for the sake of completeness, Eqs. (12.16), (12.17) and (12.10) take the form

$$p(\boldsymbol{\theta}|\mathbf{y}; \boldsymbol{\alpha}, \beta) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma}; \boldsymbol{\alpha}, \beta), \quad (13.57)$$

with

$$\boldsymbol{\mu} = \beta \boldsymbol{\Sigma} \boldsymbol{\Phi}^T \mathbf{y}, \quad \boldsymbol{\Sigma} = \left(\mathbf{A} + \beta \boldsymbol{\Phi}^T \boldsymbol{\Phi} \right)^{-1}. \quad (13.58)$$

The objective now becomes to maximize with respect to α_k , $k = 0, \dots, K - 1$, and β the cost function,

$$\begin{aligned} L(\boldsymbol{\alpha}, \beta) &:= \ln p(\mathbf{y}; \boldsymbol{\alpha}, \beta) \\ &= -\frac{N}{2} \ln(2\pi) - \frac{1}{2} |\beta^{-1} \mathbf{I} + \boldsymbol{\Phi} \boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^T| \\ &\quad - \frac{1}{2} \mathbf{y}^T \left(\beta^{-1} \mathbf{I} + \boldsymbol{\Phi} \boldsymbol{\Sigma}^{-1} \boldsymbol{\Phi}^T \right)^{-1} \mathbf{y}. \end{aligned} \quad (13.59)$$

Maximizing the above cost cannot be carried out analytically, and the following iterative scheme is derived (Problem 13.11, the proof is a bit tedious):

$$\gamma_k = 1 - \alpha_k^{(\text{old})} \Sigma_{kk}^{(\text{old})}, \quad (13.60)$$

$$\alpha_k^{(\text{new})} = \frac{\gamma_k}{(\mu_k^{(\text{old})})^2}, \quad k = 0, 1, \dots, K - 1, \quad (13.61)$$

$$\beta^{(\text{new})} = \frac{N - \sum_{k=0}^{K-1} \gamma_k}{\|\mathbf{y} - \boldsymbol{\Phi} \boldsymbol{\mu}^{(\text{new})}\|^2}. \quad (13.62)$$

The iterative scheme is initialized by an arbitrary set of values and it is repeated until a convergence criterion is met. Σ_{kk} is the respective diagonal element of the matrix $\boldsymbol{\Sigma}$. Note that both $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ depend on the values of β and α_k . The main complexity per iteration step is due to the matrix inversion involved in the respective definition in (13.58), which amounts to $O(K^3)$ operations. Moreover, because a matrix inversion is involved, one must take care of near singularities, due to numerical errors. This can be the case in practice, because some of the values of α_k may become very large. Thus, care must be taken so that once such values occur, one removes the corresponding columns in $\boldsymbol{\Phi}$ and sets the respective values of θ_k to zero. As a matter of fact, this is how sparsity is enforced by the method. Parameters with mean value equal to zero and a variance that becomes very small (precision very large) are set to zero. This behavior has empirically been observed in practice.

The alternative path to deal with the method is via the EM algorithm. This leads to an equivalent set of recursions [75], but practical experience has shown that the previously given set of updates converge faster.

Extensions of the SBL framework in the context of block sparsity and for the case of multiple measurement vectors (MMV), when elements in each nonzero row of the solution matrix are temporally correlated, are reported in [85, 86]. Moreover, in the latter one, a theoretical analysis is provided, which shows that the SBL cost function has the very desirable property that its global minimum coincides with the sparsest solution to the MMV problem.

Example 13.2. The goal of this example is to demonstrate the comparative performance, via a simulation example, of (a) the variational Bayesian method, (b) the maximum likelihood/LS (Eq. (12.6)),

and (c) the EM algorithm of [Section 12.6](#) in the context of linear regression and in particular in the sparse modeling framework. The SBL method gave results very similar to the variational approach, and it is not discussed any further. To this end, we generated the training data according to the following scenario.

The interval in the real axis $[-10, 10]$ was sampled at $N = 100$ equidistant points, x_n , $n = 1, 2, \dots, 100$. The training data comprise the pairs (y_n, x_n) , $n = 1, 2, \dots, N$, where

$$y_n = \exp\left(-\frac{1}{2} \frac{(x_n + 5.8)^2}{0.1}\right) + \exp\left(-\frac{1}{2} \frac{(x_n - 2.6)^2}{0.1}\right) + \eta_n,$$

where η_n are i.i.d. zero mean Gaussian noise samples, of variance $\sigma_\eta^2 = 0.015$. To fit the data, the following model was adopted,

$$y = \sum_{k=1}^N \theta_k \exp\left(-\frac{1}{2} \frac{(x - x_k)^2}{0.1}\right).$$

Thus, the matrix Φ has the following elements:

$$[\Phi]_{nk} = \exp\left(-\frac{1}{2} \frac{(x_n - x_k)^2}{0.1}\right), \quad n = 1, 2, \dots, N, \quad k = 1, 2, \dots, N.$$

Note that we have as many parameters as the number of training points. This is in line with the relevance vector machine rationale, which will be discussed in [Section 13.7](#). [Figure 13.7](#) illustrates the results. The red full-line curve corresponds to the true function that generates the data. The gray full-curve corresponds to the model, having plugged in as estimated values $\hat{\theta}_k$ the respective posterior mean values from Eq. (13.33). The dotted red curve corresponds to the ML solution and the dotted gray curve to the EM, where the estimates correspond to the mean of the respective posterior (Eq. (12.71)). The performance advantages of the variational approach are obvious, which almost coincide with the true one. Observe how the variational Bayesian approach managed to cope with the overfitting and pushed most of the parameters to zero values.

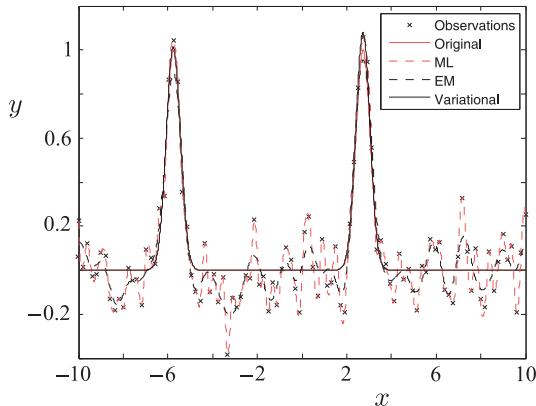
13.6.1 THE SPIKE AND SLAB METHOD

This is an old technique for imposing sparsity [37, 44]. Let us consider our familiar regression model,

$$y = \boldsymbol{\theta}^T \boldsymbol{\phi}(x) + \eta = \sum_{k=0}^{K-1} \theta_k \phi_k(x) + \eta. \quad (13.63)$$

A new set of auxiliary binary *indicator* variables are introduced, $s_k \in \{0, 1\}$, $k = 0, 1, \dots, K - 1$. Let also the prior imposed on $\boldsymbol{\theta}$ be a Gaussian, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} | \mathbf{0}, \sigma^2 I)$. As the name suggests, the indicator variables control the presence or not of a parameter in the summation in Eq. (13.63). For example, if $s_k = 1$ the corresponding parameter, θ_k , is present and if $s_k = 0$ then θ_k is removed; this is the way sparsity is imposed onto the model. To this end, a joint Bernoulli prior distribution ([Chapter 2](#)) is adopted for the indicator variables, to push as many of them as possible to zero, that is,

$$P(s) = \prod_{k=0}^{K-1} p^{s_k} (1 - p)^{1-s_k}, \quad (13.64)$$

**FIGURE 13.7**

The figure corresponds to the setup of [Example 13.2](#). Observe that the fitting curve obtained via the variational method is almost identical to the true one.

where the parameter $0 \leq p \leq 1$ specifies a prior level of sparsity. This turns out to be equivalent with adopting the following prior on the parameters:

$$p(\boldsymbol{\theta}) = \prod_{k=0}^{K-1} \left(s_k \mathcal{N}(\theta_k | 0, \sigma^2) + (1 - s_k) \delta(\theta_k) \right) : \quad \text{Spike and Slab Prior.} \quad (13.65)$$

The latter is known as the *spike and slab* prior. The name comes from the fact that if $s_k = 0$ then a “spike” is imposed at the zero and the values $s_k = 1$ imposes a “slab,” because a Gaussian is a broad one (for large enough σ^2). The corresponding posterior is not Gaussian and its computation can be done by mobilizing approximate inference techniques, such as variational or Monte Carlo (see e.g., [25] and the references therein).

Variants of the basic spike and slab scheme do also exist (see e.g., [70]). In the latter reference, it is shown that one can obtain the classical ℓ_0 -based sparsity-enforcing constraint on the LS criterion ([Chapter 9](#)) as a limiting case of one of these variants. Such a path provides another connection between probabilistic and optimization-based techniques for sparsity. Another connection will be discussed in [Section 13.10](#).

13.7 THE RELEVANCE VECTOR MACHINE FRAMEWORK

An important aspect of the work in [75] was the introduction of *Relevance Vector Machines* for regression as well as for classification. Inspired by the support vector regression (SVR) which was discussed in [Chapter 11](#), a specific regression model was considered, that is,

$$y(\mathbf{x}) = \theta_0 + \sum_{k=1}^N \theta_k \kappa(\mathbf{x}, \mathbf{x}_k) + \eta. \quad (13.66)$$

In other words, the general regression model of Eq. (12.1) is considered for $K = N + 1$, where N is the number of observations and

$$\phi_k(\mathbf{x}) = \kappa(\mathbf{x}, \mathbf{x}_k),$$

where $\kappa(\cdot, \cdot)$ is a kernel function, as defined in Chapter 11, centered at the input observation points, $\mathbf{x}_k, k = 1, 2, \dots, N$. Thus, the number of parameters becomes equal (plus one) to the number of training points.

The task can be treated either via the SBL philosophy or via the employment of the variational approximation rationale, in order to impose sparsity. In [75], it is pointed out that the variational Bayesian approach is computationally more intensive and in practice it results to mean values for the hyperparameters, which are identical to the values obtained by using the sparse Bayesian learning (SBL) approach.

Inspired by the definition of the support vectors in the support vector regression (SVR), the surviving data points that contribute to Eq. (13.66) are called *relevance vectors*. Also, the kernels to be used in the RVM framework need not be symmetric positive definite functions, because the modeling is not necessarily associated to a reproducing kernel Hilbert space (RKHS).

13.7.1 ADOPTING THE LOGISTIC REGRESSION MODEL FOR CLASSIFICATION

Besides the relevance vector regression, the relevance vector classification was also introduced in [75]. Recall that in the support vector machine (SVM) classification, a linear (in an RKH space) classifier was designed. The same model is also adopted for the RVM. Given the value of a measured feature vector, \mathbf{x} , classification is performed according to the sign of the discriminant function, namely

$$f(\mathbf{x}) := \boldsymbol{\theta}^T \phi(\mathbf{x}) := \theta_0 + \sum_{k=1}^N \theta_k \phi_k(\mathbf{x}).$$

The goal is to obtain an estimate of the parameters $\boldsymbol{\theta}$ in the Bayesian framework; thus, somehow, we have to “embed” $\boldsymbol{\theta}$ into a pdf that relates the input-output data. In this vein, a well-known and widely used technique is the *logistic regression* model, which was introduced in Section 7.6.

According to this model and for a two-class (ω_1, ω_2) classification task, the posterior probabilities, as required by the Bayesian classifier, are modeled as

$$P(\omega_1 | \mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \phi(\mathbf{x}))} : \text{ Logistic Regression Model,}$$

(13.67)

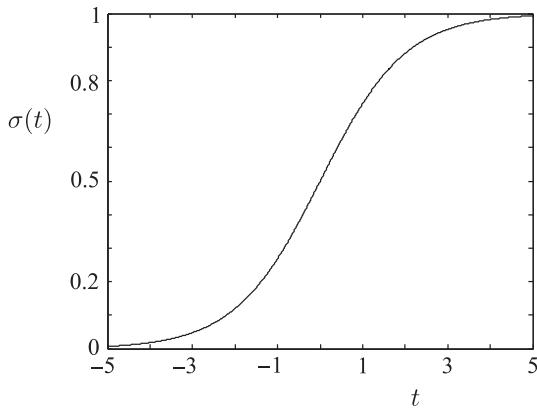
and

$$P(\omega_2 | \mathbf{x}) = 1 - P(\omega_1 | \mathbf{x}). \quad (13.68)$$

There is more than one reason that justifies such a choice (see, e.g., [40] and Problem 13.12). Multiclass generalizations are also possible (e.g., [74], and Chapter 7).

For the sake of the less familiar reader, let us look at Eq. (13.67) more closely. The graph of the function

$$\sigma(t) = \frac{1}{1 + \exp(-t)}, \quad (13.69)$$

**FIGURE 13.8**

The logistic sigmoid function.

known as the *logistic sigmoid* function, is shown in Figure 13.8. For $t > 0$ ($\theta^T \phi(x) > 0$), $P(\omega_1|x) > \frac{1}{2}$ and the decision is in favor of ω_1 . The opposite holds true for $t < 0$, ($\theta^T \phi(x) < 0$). Considering the training set $(y_n, x_n), x_n \in \mathbb{R}^l$, and $y_n \in \{0, 1\}$, and adopting a Bernoulli distribution for $P(y|x)$, the respective likelihood function can be defined as

$$P(y|\theta) = \prod_{n=1}^N \left(\sigma(\theta^T \phi(x_n)) \right)^{y_n} \left(1 - \sigma(\theta^T \phi(x_n)) \right)^{1-y_n}, \quad (13.70)$$

which is the counterpart of Eq. (13.21) for the regression case. We also adopt a Gaussian prior for θ , as in Eqs. (13.54) and (13.55). As in the SBL approach, our goal is to maximize the Type II log-likelihood with respect to the unknown parameters, α . However, $p(y|\theta)$ is no longer Gaussian, and marginalizing out θ cannot be carried out analytically. In [75], the Laplacian approximation is employed and the following stepwise procedure is adopted:

- Assuming α to be currently available, maximize with respect to θ the posterior, which by simple arguments is easily shown to be

$$p(\theta|y, \alpha) = \frac{P(y|\theta)p(\theta|\alpha)}{P(y|\alpha)},$$

or equivalently,

$$\begin{aligned} \hat{\theta}_{\text{MAP}} &= \arg \max_{\theta} \ln (P(y|\theta)p(\theta|\alpha)) \\ &= \arg \max_{\theta} \left\{ \sum_{n=1}^N \left[y_n \ln \sigma(\theta^T \phi(x_n)) + \right. \right. \end{aligned} \quad (13.71)$$

$$\begin{aligned} &\left. \left. (1 - y_n) \ln (1 - \sigma(\theta^T \phi(x_n))) \right] - \right. \\ &\left. \frac{1}{2} \theta^T A \theta + \text{constant} \right\}, \end{aligned} \quad (13.72)$$

where $A := \text{diag}\{\alpha_0, \alpha_2, \dots, \alpha_N\}$. Maximizing Eq. (13.72) with respect to θ results in (Problem 13.13),

$$\hat{\theta}_{\text{MAP}} = A^{-1} \Phi^T (y - s), \quad (13.73)$$

where $s := [s_1, \dots, s_N]^T$ and $s_n := \sigma(\theta^T \phi(x_n))$, $n = 1, 2, \dots, N$.

2. Use $\hat{\theta}_{\text{MAP}}$ and the Laplace approximation method (Section 12.3) to approximate $p(\theta|y, \alpha)$ by a Gaussian centered at $\hat{\theta}_{\text{MAP}}$ [39]. Recall from Section 12.3 that the covariance matrix of the approximate Gaussian is given by

$$\Sigma^{-1} = -\frac{\partial^2 \ln(P(y|\theta)p(\theta|\alpha))}{\partial \theta^2} \Big|_{\theta=\hat{\theta}_{\text{MAP}}},$$

or (Problem 13.14)

$$\Sigma^{-1} = (\Phi^T T \Phi + A), \quad (13.74)$$

where $T = \text{diag}\{t_1, t_2, \dots, t_N\}$ and

$$t_n = \sigma(\theta^T \phi(x_n)) \left(1 - \sigma(\theta^T \phi(x_n))\right) \Big|_{\theta=\hat{\theta}_{\text{MAP}}}$$

3. Having obtained $\hat{\theta}_{\text{MAP}}$ and computed Σ , then adapting Eq. (12.37) to our current notation we obtain

$$P(y|\alpha) = P(y|\hat{\theta}_{\text{MAP}}) p(\hat{\theta}_{\text{MAP}}|\alpha) (2\pi)^{\frac{N}{2}} |\Sigma|^{1/2}. \quad (13.75)$$

Next, maximization of Eq. (13.75) with respect to α provides the updated iteration estimate. Note that the first term of the product on the right-hand side is independent of α . Taking the logarithm and maximizing easily results in (Problem 13.15)

$$-\frac{1}{2} \theta_{\text{MAP},k}^2 + \frac{1}{2\alpha_k} - \frac{1}{2} \Sigma_{kk} = 0. \quad (13.76)$$

Because Σ_{kk} as well as $\theta_{\text{MAP},k}$ depend on α , the equation is solved iteratively and results in exactly the same scheme as in Eq. (13.61), that is,

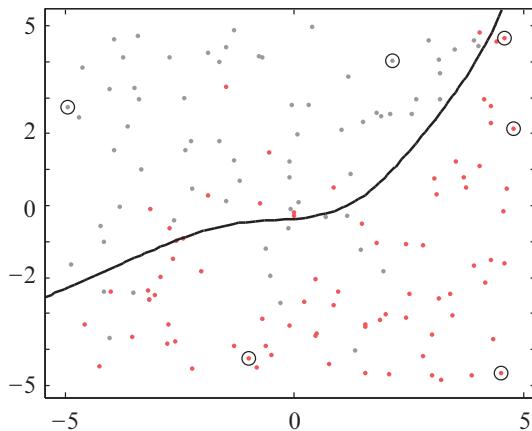
$$\alpha_k^{(\text{new})} = \frac{1 - \alpha_k^{(\text{old})} \Sigma_{kk}^{(\text{old})}}{\left(\theta_{\text{MAP},k}^{(\text{old})}\right)^2}.$$

The procedure continues until a convergence criterion is met [38, 75].

As pointed out in [75], although in general the Laplacian local approximation to a Gaussian may not be a good one, in the case of the current classification task, due to the specific nature of the adopted models, the approximation is expected to provide good accuracy.

Figure 13.9 shows the decision curve that results from the RVM method⁴ and classifies the points of the red/gray classes. The data set is the same as the one used in Example 11.4 of Chapter 11

⁴ The software used was that in <http://www.miketipping.com/sparsebayes.htm#software>

**FIGURE 13.9**

The decision curve that separates the two classes (red vs. gray), which is obtained by the RVM classifier, corresponds to posterior probability values $P(\omega_1|\mathbf{x}) = 0.5$. The Gaussian kernel was used with $\sigma^2 = 3$. Only six relevance vectors survive—the ones that have been circled.

when dealing with the SVM classifier. Six points, which have been circled, are the surviving relevance vectors. The Gaussian kernel was used with $\sigma^2 = 3$, which was found to give the best results. Observe that the number of support vectors surviving is significantly less compared to the case of SVM of [Chapter 11](#).

Remarks 13.2.

- Compared to SVM (SVR), the RVM machinery presents advantages and disadvantages. The SVM approach has the mathematically elegant property of, theoretically, giving a single minimum due to the convexity of the associated cost functions. This is not the case for the RVM framework, where the involved optimization steps refer to a nonconvex cost. It must be kept in mind that solving a nonconvex task, one may have to run the optimization algorithm a number of times, starting each time from different initial conditions, because a nonconvex problem can be trapped in a local minimum.

Concerning complexity, the algorithmic steps for the RVM involve the inversion of the Hessian matrix, which amounts to $O(N^3)$ complexity. As discussed in [Section 11.11](#), the complexity range of the efficient schemes for solving the SVM scales from linear to (approximately) quadratic.

Also, the memory for the RVM exhibits an $O(N^2)$ on the size of the training set as opposed to a linear dependence to the SVM case. Besides complexity, inverting (big) matrices must be done with care in order to avoid numerical instabilities due to possible (near) singularity. Also, in general, RVMs need longer training times to converge, compared to SVMs for similar error rates.

A fast RVM algorithm has been developed in [76] by analyzing the properties of the marginal likelihood. This enables a sequential addition and deletion of candidate basis functions (columns of Φ) to monotonically maximize the marginal likelihood. This iterative algorithm operates in a constructive manner, until all relevant basis functions (for which the associated weights are

nonzero) have been included. If M denotes the number of relevant terms, the complexity amounts to $O(M^3)$, which for $M << N$ is more efficient than the original RVM.

The main advantage of RVMs is that, in general, they result in sparser solutions compared to SVMs for similar levels of generalization errors. This makes the prediction step, after the training has been completed, more efficient compared to the prediction model resulting from SVM.

Moreover, SVMs suffer from their dependence on the user-dependent hyperparameter, C (ϵ for regression), and they are generally found by cross-validation, which involves multiple training for different values.

- In [7], a different algorithmic approach has been adopted based on the *variational bound approximation* method, to be described next.

13.8 CONVEX DUALITY AND VARIATIONAL BOUNDS

In the previous chapter, the Laplacian technique for the approximation of a general pdf by a Gaussian was introduced. The driving force behind such an approximation was to benefit from the computationally friendly nature of the Gaussian pdf. In this section, we will approach this task from a different perspective, involving maximization of a lower bound of the pdf at hand with respect to an extra parameter, which is introduced into the problem and on which the lower bound depends. Our theoretical framework is that of convex duality, a well-known and powerful tool in convex analysis.

Let a function $f : \mathbb{R}^l \mapsto \mathbb{R}$. The function

$$f^* : \mathbb{R}^l \mapsto \mathbb{R},$$

defined as⁵

$$f^*(\xi) = \max_{\mathbf{x}} \left\{ \xi^T \mathbf{x} - f(\mathbf{x}) \right\}, \quad (13.77)$$

is called the *conjugate* of f . The domain of the conjugate function consists of all $\xi \in \mathbb{R}^l$ for which the maximum is finite. A notable property of the conjugate function is its *convexity*; this is true whether or not f is convex. The convexity is the outcome of the point-wise maximization of a family of (convex with respect to ξ) affine functions [10].

Maximizing Eq. (13.77) with respect to \mathbf{x} results in a value of \mathbf{x}_* , such that

$$\mathbf{x}_* : \nabla f(\mathbf{x}_*) = \xi, \quad (13.78)$$

which leads to the value

$$f^*(\xi) = \xi^T \mathbf{x}_* - f(\mathbf{x}_*). \quad (13.79)$$

Equations (13.78) and (13.79) provide the geometric interpretation of the conjugate function. The graph of the linear function $\xi^T \mathbf{x}$ defines a hyperplane whose direction is controlled by ξ ; the latter is now equal to $\nabla f(\mathbf{x}_*)$, which defines the direction of the tangent hyperplane of the graph of $f(\mathbf{x})$ at \mathbf{x}_* . This tangent hyperplane is described by

$$g(\mathbf{x}) = f(\mathbf{x}_*) + (\mathbf{x} - \mathbf{x}_*)^T \nabla f(\mathbf{x}_*),$$

⁵ Strictly speaking, one should use the sup instead of max and inf instead of min, throughout.

or using Eq. (13.79),

$$g(\mathbf{x}) = \xi^T \mathbf{x} - f^*(\xi). \quad (13.80)$$

For $\mathbf{x} = 0$, Eq. (13.80) becomes $g(\mathbf{0}) = -f^*(\xi)$. This is illustrated in Figure 13.10. Thus, $f^*(\xi)$ corresponds to the displacement that the graph of $\xi^T \mathbf{x}$ has to undergo in order to “touch” that of $f(\mathbf{x})$. A byproduct of all these turns out to be very useful for us. It can be shown (Problem 13.16) that if f is a convex function, then $(f^*)^* = f$ and in this case we can write

$$f(\mathbf{x}) = \max_{\xi} \left\{ \mathbf{x}^T \xi - f^*(\xi) \right\}. \quad (13.81)$$

Thus, once f^* is computed, a lower bound for f becomes readily available, that is,

$$f(\mathbf{x}) \geq \mathbf{x}^T \xi - f^*(\xi), \quad (13.82)$$

where now ξ is interpreted as a parameter. To investigate this bound a bit further, plug Eqs. (13.78) and (13.79) into Eq. (13.82) to obtain⁶

$$f(\mathbf{x}) \geq f(\mathbf{x}_*) + (\mathbf{x} - \mathbf{x}_*)^T \nabla f(\mathbf{x}_*),$$

where the right-hand side is the linear function $g(\mathbf{x})$ describing the hyperplane tangent to $f(\mathbf{x})$ at \mathbf{x}_* . The bound becomes tight at $\mathbf{x} = \mathbf{x}_*$; see Figure 13.10. We will soon see how we can make this linear function bound a nonlinear one; it suffices to transform the argument of the function.

All that has been said for convex functions applies to concave ones if we replace the max operation in Eqs. (13.77) and (13.81) with min operations. Note that following this definition, the conjugate function is *concave*, being the result of point-wise minimization of a set of concave functions (an affine function can be considered either convex or concave). Furthermore, if the involved function is neither convex nor concave, one can search for *invertible* transformations that render it convex or concave.

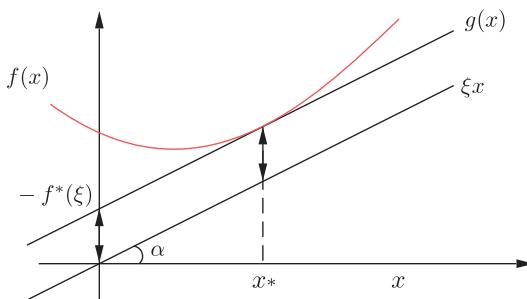


FIGURE 13.10

The direction of the line $y = \xi x$ that crosses the origin is controlled by ξ ($\xi = \tan \alpha$). The (negative) value of the conjugate function f^* at ξ defines the point where the line $y = g(x)$ cuts the vertical axis; $g(x)$ is formed by translating ξx until it becomes tangent to f at x_* .

⁶ Note that this is a necessary and sufficient condition for convexity.

In our context, the purpose of resorting to the notion of the conjugate function is our expectation that such a function, which bounds a (pdf) function as in Eq. (13.82), may lead to a functional form that lends itself to tractable computations of the involved integrations.

Example 13.3. Compute the conjugate of the logarithmic function $f(x) = \ln x, x > 0$.

The logarithmic function is known to be concave. Hence

$$f^*(\xi) = \min_{x>0} \{\xi x - \ln x\},$$

or

$$x_* : \frac{1}{x} = \xi \Rightarrow x_* = \frac{1}{\xi}.$$

Hence,

$$f^*(\xi) = 1 + \ln \xi.$$

Therefore,

$$\ln x = \min_{\xi>0} \{\xi x - 1 - \ln \xi\}.$$

Figure 13.11 shows the respective graphs of the logarithmic function as well as the resulting linear function bound for different values of ξ .

Example 13.4. Consider the univariate Laplacian pdf, which we have already seen in Section 13.5,

$$p(\theta) = \frac{\lambda}{2} \exp(-\lambda|\theta|), \quad \theta \in \mathbb{R}. \quad (13.83)$$

Our goal is to derive a lower bound in terms of its conjugate function. From Eq. (13.83), we get

$$\ln p(\theta) = \ln \frac{\lambda}{2} - \lambda|\theta|.$$

Define

$$f(x) = \ln \frac{\lambda}{2} - \lambda\sqrt{x}, \quad x > 0. \quad (13.84)$$

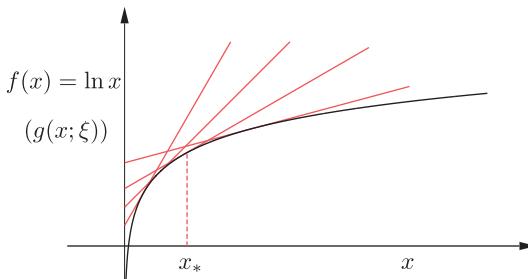


FIGURE 13.11

The linear functions $g(x; \xi) = \xi x - 1 - \ln \xi$ provide upper bounds to $f(x) = \ln x$. Each one of the lines is tangent to $f(x) = \ln x$ at the point $x_* = \frac{1}{\xi}$.

Then

$$\ln p(\theta) = f(\theta^2). \quad (13.85)$$

Note that $f(x)$ is a convex function with respect to x (Problem 13.16). The conjugate of $f(x)$ is obtained as⁷

$$f^*(\xi) = \max_x \left\{ -\frac{\xi}{2}x - f(x) \right\}, \quad \xi > 0. \quad (13.86)$$

Recalling Eq. (13.84), maximization leads to

$$x_* : \lambda x^{-\frac{1}{2}} = \xi \Rightarrow x_* = \lambda^2 \xi^{-2}. \quad (13.87)$$

Combining Eqs. (13.86) and (13.87) gives

$$f^*(\xi) = \frac{\lambda^2}{2} \xi^{-1} - \ln \frac{\lambda}{2}. \quad (13.88)$$

Hence, we obtain the bound

$$f(x) \geq -\frac{\xi}{2}x - \frac{\lambda^2}{2} \xi^{-1} + \ln \frac{\lambda}{2},$$

or

$$\ln p(\theta) \geq -\frac{\xi}{2}\theta^2 - \frac{\lambda^2}{2}\xi^{-1} + \ln \frac{\lambda}{2}, \quad \xi > 0.$$

Because this is true $\forall \xi > 0$, we can replace ξ with ξ^{-1} , for notational convenience, which results in

$$p(\theta) \geq \frac{\lambda}{2} \exp\left(-\frac{\xi^{-1}}{2}\theta^2\right) \exp\left(-\frac{\lambda^2}{2}\xi\right), \quad (13.89)$$

which, after mobilizing the Gaussian notation and its integration property, can be rewritten as

$$p(\theta) \geq \mathcal{N}(\theta|0, \xi)\phi(\xi), \quad \xi > 0 \quad (13.90)$$

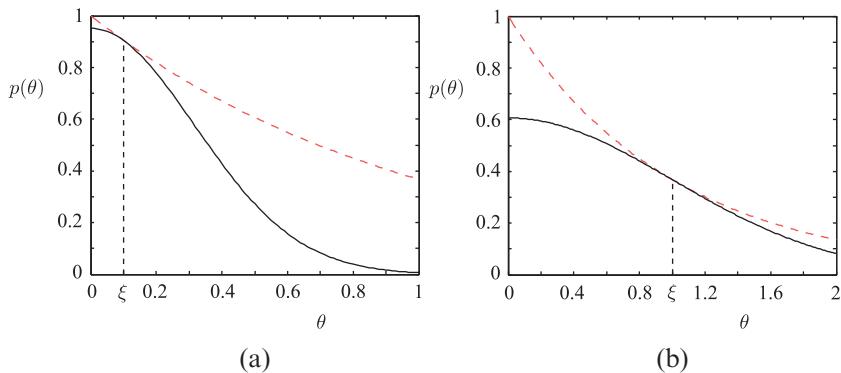
with

$$\phi(\xi) = \frac{\lambda}{2} \sqrt{2\pi\xi} \exp\left(-\frac{\lambda^2}{2}\xi\right), \quad \xi > 0.$$

This is very interesting indeed. The obtained lower bound has a functional dependence on θ , which is of a Gaussian nature; the Gaussian term is centered at zero with variance ξ . Maximizing with respect to ξ , we will obtain the required approximation. Figure 13.12 shows the obtained approximation for different values of ξ . Observe that introducing transformations in the involved variables can render the function in the obtained bound a *nonlinear* one.

⁷ Because maximization takes place for all ξ , we use $-\frac{\xi}{2}$. This is only for notational convenience in order to obtain the result in a convenient form.

⁸ ξ is constrained to positive values, because for $\xi \leq 0$ the maximum with respect to x becomes infinite, which violates the definition of the conjugate functions.

**FIGURE 13.12**

The Laplacian (red curves) and the approximating Gaussians for two different values of ξ .

For a multivariate Laplacian and assuming a parameter vector with independent components, it can be trivially shown that

$$p(\boldsymbol{\theta}) = \prod_{k=0}^{K-1} p(\theta_k) \geq \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \boldsymbol{\Xi}) \prod_{k=0}^{K-1} \phi(\xi_k) := \hat{p}(\boldsymbol{\theta}; \boldsymbol{\xi}), \quad (13.91)$$

where $\boldsymbol{\xi} = [\xi_0, \dots, \xi_{K-1}]^T$ and

$$\boldsymbol{\Xi} := \text{diag}\{\xi_0, \xi_1, \dots, \xi_{K-1}\}.$$

Remarks 13.3.

- The method of representing a convex function via the optimization of lower bound in terms of its conjugate (dual form) is known as the *variational method*, and the associated parameters ξ_k as *variational* parameters [59]. Its use in the context of machine learning was first reported in [27] (see also [31]), and its use subsequently proliferated and was adopted in different scenarios.
- The method has been used to obtain variational approximations for a number of pdfs that are suitable for sparsity-aware learning, for example, Jeffreys', student's-t, generalized Gaussians (see, e.g., [52]), and for the logistic regression model [29] (Problem 13.18). Compared to the Laplacian approximation method, the variational approach provides the extra flexibility of optimizing with respect to the corresponding variational parameters (see [29] for a related discussion). The reader, however, has to keep in mind that both approximations need not always be good ones. This is readily observed from Figure 13.12. One may obtain a good approximation locally, but not everywhere. However, it turns out that in practice, in the context of Bayesian learning, a poor approximation of the prior (for which such approximations are used) does not necessarily lead to a poor approximation of the posterior. There is no guarantee of it, and it is only the performance in practice that has the final verdict. This can be considered a drawback of the Bayesian technique compared to the deterministic methods based on optimization criteria. The latter ones, by adopting usually convex cost functions, can lead to solutions that are well characterized. In contrast,

Bayesian inference techniques suffer from their nonconvex nature and the fact that very often the imposed approximations may not necessarily be good ones. However, this is always the case in life. There is no free lunch. At the time this book was written, both of these paths to machine learning still comprised viable and powerful techniques, with their pros and cons.

13.9 SPARSITY-AWARE REGRESSION: A VARIATIONAL BOUND BAYESIAN PATH

The goal of this section is to demonstrate the use of convex duality and the respective variational bounds in order to approximate the computation of the evidence function, in cases where the corresponding integral is intractable. We have chosen to describe the method in the framework of sparsity-aware learning; this can also help us establish bridges with [Chapter 9](#), and some of the results will be used in [Section 13.10](#) for this purpose.

To comply with the assumptions made in [Chapter 9](#), and without loss of generality, let us assume that the involved data have zero mean values. If not, the training data can be centered by subtracting their respective sample means; thus, we set $\theta_0 = 0$ and assume that the number of parameters is K . Then, our regression model becomes

$$\mathbf{y} = \Phi\boldsymbol{\theta} + \boldsymbol{\eta}, \quad \boldsymbol{\eta}, \mathbf{y} \in \mathbb{R}^N, \quad \boldsymbol{\theta} \in \mathbb{R}^K, \quad K > N,$$

where we are informed about the “secret” that most of the components of $\boldsymbol{\theta}$ are (almost) zero. In [Section 13.5](#), we commented on the inadequacy of a Gaussian prior to provide a reasonable statistical description of a sparse random vector. A heavy-tailed distribution that enjoys popularity for sparse modeling is the Laplacian one. After all, adopting a Laplacian prior to $\boldsymbol{\theta}$, that is,

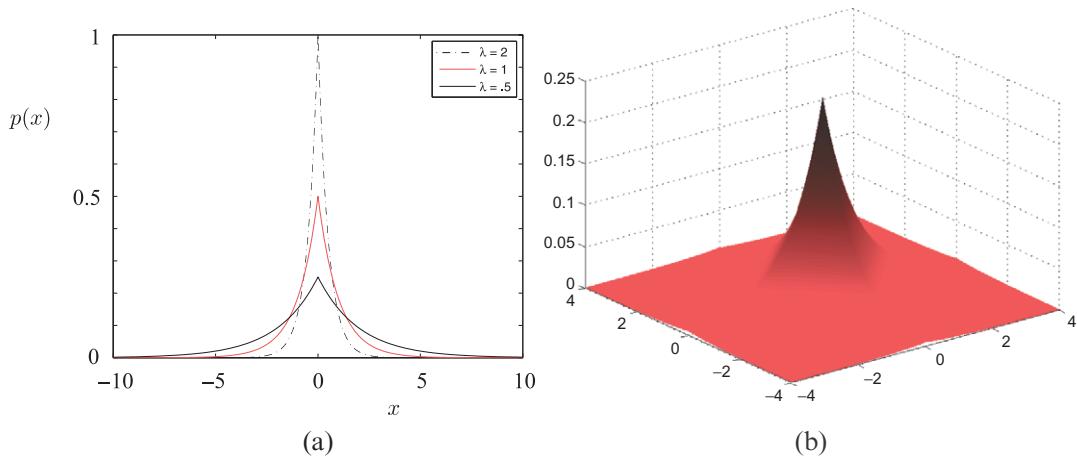
$$p(\boldsymbol{\theta}) = \prod_{k=1}^K p(\theta_k) = \prod_{k=1}^K \frac{\lambda}{2} \exp(-\lambda|\theta_k|),$$

and a Gaussian conditional pdf, $p(\mathbf{y}|\boldsymbol{\theta})$, for the observations \mathbf{y} as in Eq. (13.21), makes the MAP estimation identical to our familiar LASSO task, discussed in [Chapter 9](#). In this vein, we will build this section around the Laplacian pdf. [Figure 13.13a](#) shows the Laplacian for different values of λ . In [13.13b](#) the two-dimensional plot is provided, from which the respect that this pdf shows to sparse solutions is readily observed.

The problem with the Laplacian pdf is that its presence in Eq. (12.14) makes the computation of the integral computationally intractable. Also, recall from [Section 12.4](#) that the Laplacian pdf does not belong to the computationally attractive exponential family. To facilitate its treatment, we will employ the variational bound approximation method in order to approximate the Laplacian by a Gaussian, following Eqs. (13.89) and (13.91). The variational parameters will be determined by maximizing the respective evidence via the EM algorithm. This method in recovering sparse solutions was first introduced in [18] in the context of dictionary learning.

For simplicity, let the noise sequence in our regression model be white with variance $\sigma_\eta^2 := \frac{1}{\beta}$. Then

$$p(\mathbf{y}|\boldsymbol{\theta}; \beta) = \mathcal{N}(\mathbf{y}|\Phi\boldsymbol{\theta}, \beta^{-1}\mathbf{I}),$$

**FIGURE 13.13**

(a) The one-dimensional Laplacian for different values of λ . (b) A plot of a two-dimensional Laplacian pdf.

and using Eq. (13.91), we can write

$$\begin{aligned} p(\mathbf{y}; \beta) &= \int \mathcal{N}(\mathbf{y}|\Phi\boldsymbol{\theta}, \beta^{-1}I)p(\boldsymbol{\theta}) d\boldsymbol{\theta} \\ &\geq \left(\int \mathcal{N}(\mathbf{y}|\Phi\boldsymbol{\theta}, \beta^{-1}I)\mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \Xi) d\boldsymbol{\theta} \right) \prod_{k=1}^K \phi(\xi_k). \end{aligned}$$

We know by now that the integral results in a new Gaussian function (recall Eq. (12.15)), hence

$$p(\mathbf{y}; \beta) \geq \mathcal{N}(\mathbf{y}|\mathbf{0}, \beta^{-1}I + \Phi\Xi\Phi^T) \prod_{k=1}^K \phi(\xi_k) := \hat{p}(\mathbf{y}; \beta, \Xi). \quad (13.92)$$

The unknown values of β and Ξ could be obtained by direct maximization of the previous bound. However, here we will adopt an EM algorithm approach, in a similar way as in Section 12.6; the difference here lies in the existence of the multiplicative terms $\phi(\xi_k)$ that differentiates the M-step. In order to employ the EM, we need to know the posterior $p(\boldsymbol{\theta}|\mathbf{y}; \beta)$. We will accept the following,

$$p(\boldsymbol{\theta}|\mathbf{y}; \beta) \simeq \hat{p}(\boldsymbol{\theta}|\mathbf{y}; \beta, \Xi) := \frac{\mathcal{N}(\mathbf{y}|\Phi\boldsymbol{\theta}, \beta^{-1}I)\mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \Xi)}{\int \mathcal{N}(\mathbf{y}|\Phi\boldsymbol{\theta}, \beta^{-1}I)\mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \Xi) d\boldsymbol{\theta}}, \quad (13.93)$$

where in place of $p(\boldsymbol{\theta})$ we have used its respective bound, $\hat{p}(\boldsymbol{\theta}; \xi)$, from Eq. (13.91). Note that irrespective of which method one adopts to optimize with respect to the unknown parameters, the approximate posterior given in Eq. (13.93) is the quantity of interest in regression; this is used either to predict $\boldsymbol{\theta}$ or to perform predictions of the output value (Eq. (12.18)).

It must be stressed, however, that Eq. (13.93) is not a bound of $p(\boldsymbol{\theta}|\mathbf{y}; \beta)$ anymore, because normalization has taken place and division does not necessarily respect bounds. Recalling what we said in Section 12.2.2 (Eqs. (12.27) and (12.28)) we obtain

$$\hat{p}(\boldsymbol{\theta}|\mathbf{y}; \beta, \Xi) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_{\boldsymbol{\theta}|\mathbf{y}}, \Sigma_{\boldsymbol{\theta}|\mathbf{y}}), \quad (13.94)$$

where

$$\boldsymbol{\mu}_{\theta|y} = \boldsymbol{\Xi}\boldsymbol{\Phi}^T \left(\frac{1}{\beta} I + \boldsymbol{\Phi}\boldsymbol{\Xi}\boldsymbol{\Phi}^T \right)^{-1} \mathbf{y}, \quad (13.95)$$

$$\boldsymbol{\Sigma}_{\theta|y} = \boldsymbol{\Xi} - \boldsymbol{\Xi}\boldsymbol{\Phi}^T \left(\frac{1}{\beta} I + \boldsymbol{\Phi}\boldsymbol{\Xi}\boldsymbol{\Phi}^T \right)^{-1} \boldsymbol{\Phi}\boldsymbol{\Xi}. \quad (13.96)$$

We are now ready to give the algorithmic steps. Recall that in EM, the goal is to maximize the expected value of the complete log-likelihood with respect to the unknown set of deterministic parameters. In our case, our goal will be to maximize the corresponding bound with respect to β and $\boldsymbol{\xi}$,

$$\mathbb{E} \left[\ln p(\mathbf{y}, \boldsymbol{\theta}; \beta) \right] = \mathbb{E} \left[\ln \left(p(\mathbf{y}|\boldsymbol{\theta}; \beta)p(\boldsymbol{\theta}) \right) \right] \geq \mathbb{E} \left[\ln \left(p(\mathbf{y}|\boldsymbol{\theta}; \beta)\hat{p}(\boldsymbol{\theta}; \boldsymbol{\xi}) \right) \right]$$

Assuming $\boldsymbol{\xi}^{(0)}, \beta^{(0)}$ are known, the $(j+1)$ iteration comprises the following computations:

- E-step: From Eq. (13.95), Eq. (13.96) compute

$$\boldsymbol{\mu}_{\theta|y}^{(j)} \quad \text{and} \quad \boldsymbol{\Sigma}_{\theta|y}^{(j)}.$$

Following similar steps as in Section 12.6, we readily obtain that

$$\begin{aligned} \mathcal{Q}(\boldsymbol{\xi}, \beta; \boldsymbol{\xi}^{(j)}, \beta^{(j)}) &= \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \frac{\beta}{2} \mathbb{E}_{\theta|y} \left[\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|^2 \right] \\ &\quad + \sum_{k=1}^K \ln \phi(\xi_k) - \frac{K}{2} \ln(2\pi) - \frac{1}{2} \ln |\boldsymbol{\Xi}| \\ &\quad - \frac{1}{2} \sum_{k=1}^K \frac{\mathbb{E}_{\theta|y} [\theta_k^2]}{\xi_k}, \end{aligned} \quad (13.97)$$

where (recall Eq. (12.76))

$$\mathbb{E}_{\theta|y} \left[\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|^2 \right] = \|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\mu}_{\theta|y}^{(j)}\|^2 + \text{trace} \left\{ \boldsymbol{\Phi}\boldsymbol{\Sigma}_{\theta|y}^{(j)}\boldsymbol{\Phi}^T \right\},$$

and (recall Eq. (12.74))

$$\mathbb{E}_{\theta|y} [\theta_k^2] = \left[\boldsymbol{\mu}_{\theta|y}^{(j)} \boldsymbol{\mu}_{\theta|y}^{(j)T} + \boldsymbol{\Sigma}_{\theta|y}^{(j)} \right]_{kk}.$$

- M-Step: Taking the derivative of $\mathcal{Q}(\boldsymbol{\xi}, \beta; \boldsymbol{\xi}^{(j)}, \beta^{(j)})$ with respect to β and equating to 0 we get

$$\beta^{(j+1)} = \frac{N}{\|\mathbf{y} - \boldsymbol{\Phi}\boldsymbol{\mu}_{\theta|y}^{(j)}\|^2 + \text{trace} \left\{ \boldsymbol{\Phi}\boldsymbol{\Sigma}_{\theta|y}^{(j)}\boldsymbol{\Phi}^T \right\}}. \quad (13.98)$$

The derivation with respect to $\xi_k, k = 1, 2, \dots, K$, results in (Problem 13.19)

$$\xi_k^{(j+1)} = \sqrt{\frac{\mathbb{E}_{\theta|y} [\theta_k^2]}{\lambda^2}}, \quad (13.99)$$

which completes the loop. Iterations continue until a termination criterion is met.

An alternative viewpoint that justifies the maximization of the bound of the evidence with respect to the variational parameters is the following (see also [84] for a related discussion): At each iteration step, the EM algorithm maximizes $\mathbb{E} [p(\mathbf{y}|\boldsymbol{\theta}; \beta) \hat{p}(\boldsymbol{\theta}; \boldsymbol{\xi})]$ due to the monotonicity of the logarithmic function. Equivalently, this can be seen as the following minimization task,

$$\boldsymbol{\xi} = \arg \min_{\boldsymbol{\xi}} \mathbb{E} \left[p(\mathbf{y}|\boldsymbol{\theta}; \beta) |p(\boldsymbol{\theta}) - \hat{p}(\boldsymbol{\theta}; \boldsymbol{\xi})| \right], \quad (13.100)$$

where the lower bound property (Eq. (13.91)) has been used in order to involve the absolute value. Looking at Eq. (13.100), one may think of a reason that justifies what in practice is commonly observed; that is, the method results in good performance although the overall approximation of the prior may not be a good one. The important issue is to have a good approximation in values of $\boldsymbol{\theta}$ that correspond to relatively large values of $p(\mathbf{y}|\boldsymbol{\theta})$. The approximation in ranges of $\boldsymbol{\theta}$ where $p(\mathbf{y}|\boldsymbol{\theta}) \approx 0$ does not affect the main goal of the task. Moreover, Eq. (13.100) could also provide a justification of the relative advantage of the variational approximation method compared to the Laplacian method; in the latter, there is no room left to leverage any extra parameters in order to improve the final goal.

Remarks 13.4.

- As we have already commented in Chapter 9, sparsity-aware learning has been a field of intense research. No doubt this is also the case for the Bayesian approach to sparsity-promoting models. So far, we presented a hierarchical approach in Section 13.5, where sparsity was indirectly imposed by associating a gamma pdf prior on each one of the precision variables individually; this led to an equivalent high-tail student's-t pdf description of the involved parameters $\boldsymbol{\theta}$. In the current section, a Laplacian prior was imposed on $\boldsymbol{\theta}$ in order to promote sparseness. These are not the only possibilities. We focused on them in order to demonstrate two possible paths to treat the evidence maximization whenever the resulting integral is computationally “awkward.”
- In [15], sparsity in the Bayesian framework is attacked by imposing a Gaussian prior on the parameters, treating variance as latent variables with an exponential prior. Such modeling is equivalent to a Laplacian pdf, once variances are integrated out. The EM procedure is then used to compute the required estimates. Also in this paper, the use of Jeffreys' prior ($p(x) \simeq \frac{1}{x}$) is proposed as an alternative to the Laplacian one.
- In [4], sparsity is imposed in a similar way as before, but in the hierarchical model, the parameter controlling the exponential prior of the precisions is also treated as a latent variable with a Jeffreys' prior. In [5], sparsity on the unknown parameters was imposed via a generalized Gaussian pdf, that is,

$$p(\boldsymbol{\theta}|\alpha) \propto \exp \left(-\lambda \sum_{k=1}^K |\theta_k|^p \right). \quad (13.101)$$

Combining this prior with a Gaussian pdf for the conditional, $p(\mathbf{y}|\boldsymbol{\theta})$ in Eq. (13.21), would result in a MAP that corresponds to the LS regularized by a nonconvex $\ell_p, p < 1$ norm; we know that such norms are more aggressive, compared to the ℓ_1 norm, in recovering sparse solutions. For $p = 1$, Eq. (13.101) becomes the Laplacian prior. In [5], gamma priors are used in association with the hyperparameter α and the noise variance, and the variational Bayesian approach is used to obtain the solution.

- In [30], the RVM framework is exploited to obtain sparse solutions and the information related to the variance of the obtained estimates is used to determine the number of measurements, which is sufficient for recovering the solution in the framework of compressed sensing.

13.10 SPARSITY-AWARE LEARNING: SOME CONCLUDING REMARKS

The goal of this section is to establish bridges among the different aspects of sparsity-aware learning that have been addressed in various parts of this book. In [Chapter 9](#), it was treated as an optimization of a regularized cost function. In [Section 13.3](#), the automatic relevance determination (ARD) concept in Bayesian learning was discussed, and in [Section 13.9](#), the variational bound technique was exploited to overcome the computational obstacle associated with the Laplacian prior.

Let us recall our basic regression model (assuming centered data) and K unknown parameters,

$$\mathbf{y} = \Phi\boldsymbol{\theta} + \boldsymbol{\eta}, \quad \mathbf{y}, \boldsymbol{\eta} \in \mathbb{R}^N, \boldsymbol{\theta} \in \mathbb{R}^K, \quad (13.102)$$

where the interest in the theory for sparse modeling is for $K > N$, and $\boldsymbol{\theta}$ is assumed to have most of its components (approximately) equal to zero, and $\Phi \in \mathbb{R}^{N \times K}$ is a fixed predetermined matrix ($\boldsymbol{\theta}$ is replaced by $\boldsymbol{\theta}$ if it is assumed to be treated as a random vector). Viewed differently, this can be seen as a task of expanding a signal vector in terms of the columns of an overcomplete dictionary in the presence of noise ([Chapter 9](#)).

There are two questions that are naturally posed. The first one concerns the relationship between the Bayesian and the regularized cost function optimization approaches. How different are they? Are there any paths that establish connections between them? The second question addresses theoretical issues associated with the performance of the Bayesian techniques; recall that in our Bayesian treatment of the sparse modeling task, no such theorems were mentioned. In contrast, [Chapter 9](#) was full of theorems concerning conditions, properties, and performance bounds related to sparse solutions. Needless to say that all these theorems were proved under assumptions on the nature of the data matrix Φ . A first systematic attempt to address both questions was made in [80, 82, 83]. Our goal is to highlight some of the findings in this innovative reported research.

The cost function for the regularized regression in a more general setting is written as

$$L_I(\boldsymbol{\theta}, \lambda) = \|\mathbf{y} - \Phi\boldsymbol{\theta}\|^2 + \lambda \sum_{k=1}^K g(\theta_k), \quad (13.103)$$

and the resulting estimate is given by

$$\hat{\boldsymbol{\theta}}_I = \arg \min_{\boldsymbol{\theta}} L_I(\boldsymbol{\theta}, \lambda). \quad (13.104)$$

This formulation covers all the regularizers discussed in [Chapter 9](#). For example, if $g(\theta_k) = |\theta_k|$, the ℓ_1 norm results. Also, we have already pointed out that Eq. (13.104) can be seen as the MAP estimator, assuming Gaussian noise and a prior of the form

$$p(\boldsymbol{\theta}) \propto \exp \left(-\frac{1}{2} \sum_{k=1}^K g(\theta_k) \right). \quad (13.105)$$

Hence, from now on, we will refer to the regularized regression approach as Type I estimator (see [Remarks 12.1](#)). We know that in order to promote sparsity, the function g must be of a certain form; for example, it has been established in [Chapter 9](#) that $g(x) = x^2$ is not a good choice. Functions that promote sparsity can be obtained by using

$$g(x) = h(x^2),$$

where $h(x)$ is *concave* and *nondecreasing* in $[0, \infty)$. As a matter of fact, the more concave h is, the more aggressive toward sparsity the optimization becomes (e.g., [\[52\]](#)). A number of densities of the form of Eq. (13.105) satisfy this criterion, such as generalized Gaussians ($\exp(-|x|^p)$, $0 < p \leq 2$), student's-t ($((1 + x^2/\nu)^{-\frac{\nu+1}{2}})$, logistic ($1/\cosh^2(\frac{x}{2})$), and Laplace. An example is $h(x) = \sqrt{x}$ (which is concave) and results in our familiar $g(x) = |x|$. In the context of the regularized cost functions in [Chapter 9](#), we have also discussed other forms, for example, $g(x) = \ln(|x| + \epsilon)$.

Let us now return to the Bayesian alternative path, which we will refer to as Type II ([Remarks 12.2](#)). For notational compliance with Eq. (13.103), assume

$$p(\mathbf{y}|\boldsymbol{\theta}) \propto \exp\left(-\frac{1}{2\lambda} \|\mathbf{y} - \Phi\boldsymbol{\theta}\|^2\right)$$

and $p(\boldsymbol{\theta})$ given as in Eq. (13.105). Following similar arguments as in [Example 13.4](#) and setting $h(x)$ in place of \sqrt{x} , we can readily obtain that

$$p(\theta_k) \geq \mathcal{N}(\theta_k|0, \xi_k)\phi(\xi_k), \quad \xi_k > 0, \quad (13.106)$$

where now $\phi(\xi_k)$ depends on the dual $f^*(\xi)$ of

$$f(x) = C - \frac{1}{2}h(x),$$

and C is a proportionality constant. Therefore, as in [Section 13.9](#), the starting point involves the approximation of the unnormalized prior

$$\hat{p}(\boldsymbol{\theta}; \Xi) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \Xi) \prod_{k=1}^K \phi(\xi_k),$$

as well as the normalized posterior

$$\hat{p}(\boldsymbol{\theta}|\mathbf{y}; \lambda, \Xi) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_{\theta|\mathbf{y}}, \Sigma_{\theta|\mathbf{y}}).$$

Using Eq. (13.95) we get

$$\boldsymbol{\mu}_{\theta|\mathbf{y}} = \Xi \Phi^T (\lambda I + \Phi \Xi \Phi^T)^{-1} \mathbf{y}, \quad (13.107)$$

where λ has replaced β^{-1} , and as in Eq. (13.96),

$$\Sigma_{\theta|\mathbf{y}} = \Xi - \Xi \Phi^T \left(\lambda I + \Phi \Xi \Phi^T \right)^{-1} \Phi \Xi.$$

From Eq. (13.107), it is readily seen that if $\boldsymbol{\xi}$ is sparse, then $\boldsymbol{\mu}_{\theta|\mathbf{y}}$ is also sparse. Also, recall from [Section 13.9](#) that the bound on the evidence, after marginalizing out $\boldsymbol{\theta}$, is given by (13.92),

$$\hat{p}(\mathbf{y}; \lambda, \Xi) = \mathcal{N}(\mathbf{y} | \mathbf{0}, \lambda I + \Phi \Xi \Phi^T) \prod_{k=1}^K \phi(\xi_k). \quad (13.108)$$

Note that if one assumes $\phi(\xi_k) = 1$, $k = 1, 2, \dots, K$, then the task becomes identical with the ARD approach, which was discussed in [Section 13.3](#); in other words, it is the evidence if a Gaussian prior is used for each one of the parameters, θ_k , with a different variance parameter (ξ_k) assigned to each one of them.

The variational parameters ξ (as well as λ) are obtained by maximizing the bound in Eq. (13.108). In [Section 13.9](#), the EM algorithm was adopted. Here, we will act directly on Eq. (13.108) and our goal will be to minimize the negative logarithm of the bound; that is, the cost function will be

$$L_{II}(\xi; \lambda) = -2 \ln \mathcal{N}(\mathbf{y} | \mathbf{0}, \lambda I + \Phi \Xi \Phi^T) - 2 \sum_{k=1}^K \ln \phi(\xi_k)$$

or

$$L_{II}(\xi; \lambda) = \mathbf{y}^T \Sigma_y^{-1} \mathbf{y} + \ln |\Sigma_y| + \sum_{k=1}^K \psi(\xi_k), \quad (13.109)$$

where

$$\psi(\xi_k) := -2 \ln \phi(\xi_k),$$

and

$$\Sigma_y := \lambda I + \Phi \Xi \Phi^T,$$

and the constants have been neglected. In words, the heart of the Type I method beats around Eq. (13.103) and minimization takes place in the θ -parameters space, while that of Type II method beats around Eq. (13.109) and the optimization is performed in the ξ -parameters (variational) space.

However, the two approaches are not as different as they may appear at first glance. The Type I optimization task, based on the loss function in Eq. (13.103), can also be expressed in the ξ -parameters space. To this end, let us apply a variational bound on g in Eq. (13.103) (after all, it is the specific nature of g that led to the bound in Eq. (13.106)); then it is shown ([83], [Problem 13.20](#)) that the Type I estimator can also result by minimizing a rigorous upper bound of $L_I(\theta, \lambda)$, that is,

$$L_I^\xi(\xi, \lambda) := \mathbf{y}^T \Sigma_y^{-1} \mathbf{y} + \sum_{k=1}^K f_I(\xi_k), \quad (13.110)$$

where

$$f_I(\xi_k) := \ln \xi_k + \psi(\xi_k).$$

Once $\hat{\xi}_I$ is obtained, that is,

$$\hat{\xi}_I = \arg \min_{\xi} L_I^\xi(\xi, \lambda), \quad \xi_k > 0, k = 1, 2, \dots, K,$$

then the optimizer in Eq. (13.104) is obtained as

$$\hat{\theta}_I = \Xi_I \Phi^T (\lambda I + \Phi \Xi_I \Phi^T)^{-1} \mathbf{y}, \quad (13.111)$$

with

$$\Xi_I := \text{diag}\{\hat{\xi}_{I1}, \dots, \hat{\xi}_{IK}\}.$$

In [83], it is pointed out that the correspondence between the two formulations carries on to the local minima as well. That is, $\hat{\theta}_I$ is a local minimum of Eq. (13.103), iff $\hat{\xi}_I$ is a local minimum of Eq. (13.110).

Following similar arguments, one can show that optimizing Eq. (13.109), with respect to ξ , can equivalently be expressed in the θ -parameters space ([Problem 13.21](#)) via the cost function

$$L_{II}^{\theta}(\theta, \lambda) = \|\mathbf{y} - \Phi\theta\|^2 + \lambda g_{II}(\theta), \quad (13.112)$$

where

$$g_{II}(\theta, \lambda) = \arg \min_{\xi} \left(\sum_{k=1}^K \frac{\theta_k^2}{\xi_k} + \ln |\Sigma_y| + \sum_{k=1}^K \psi(\xi_k) \right), \quad \xi_k > 0, \quad k = 1, 2, \dots, K, \quad (13.113)$$

and $\hat{\theta}_{II}$ is given by Eq. (13.107) using the optimal ξ parameters. Moreover, as pointed out in [83], this correspondence between the two formulations extends to the local minima, under certain assumptions.

Comparing Eq. (13.110) with Eq. (13.109), and Eq. (13.112) with Eq. (13.103), one observes the essential difference between Type I and Type II approaches, that is, the presence of the $\ln |\Sigma_y|$ term in the latter case. This is the result of the marginalization step in the Bayesian methods. Integration over θ introduces dependencies among variables, giving rise to this coupling term. Viewing it as a penalizing term in the least-squares cost function, this term does not possess the separable structure of $\sum_k g(\theta_k)$. This is the power and at the same time the drawback of the Bayesian approach. It provides more information, which can be used to the benefit of the performance, but at the same time increases the computational load. In [65], a technique for optimizing Eq. (13.112) is proposed as an alternative to the slow, more standard techniques, such as the EM. The method relies on the introduction of a variational bound on $\ln |\Sigma_y|$, which leads to a majorization-minimization scheme (see also [66]).

Finally, taking advantage of the θ -parameters space formulation of the Type II estimation path, the following performance-related results have been shown in [83]:

- If $\psi(\xi_k) = 0$, $k = 1, 2, \dots, K$,⁹ $g_{II}(\theta, \lambda)$ in Eq. (13.113) is nondecreasing and concave and every local minimum of Eq. (13.112) has *at most* N nonzero elements, regardless of λ .
- For the noiseless case task, $\mathbf{y} = \Phi\theta$, if Φ is of full spark and there is at least one feasible solution, such that $\|\theta\|_0 < N$, and $\psi(\xi_k) = 0$, $\forall k$, the set of global minimizers of the Type II task equals the set of global ℓ_0 norm minimizers.

Parameter identifiability and sparse Bayesian modeling

The task of parameter identifiability refers to the conditions under which the set of parameters that define a model can be learned, if this is possible. In a more general setting, given a parametric model, $f(\mathbf{y}; \theta)$, associated with a set of variables, \mathbf{y} , we say that the parameters, θ , are *identifiable* if $f(\mathbf{y}; \theta_1) \neq f(\mathbf{y}; \theta_2)$, $\forall \theta_1 \neq \theta_2$. From such a perspective, some of the sparsity conditions discussed in [Chapter 9](#) are identifiability conditions. The task of parameter identifiability in the context of sparse Bayesian learning has been treated in [56].

⁹ This condition can be relaxed to concave and nondecreasing functions.

The starting point is the model in Eq. (13.102), where a zero mean Gaussian prior is chosen for θ , with a *diagonal* covariance matrix, Σ_θ . Moreover, it is assumed that only k out of the $K > N$ elements of this matrix are nonzero. Then, we know that the marginal of the observations, after marginalizing out θ , (evidence) is also Gaussian with a covariance matrix given by Eq. (12.15). The goal of the task reported in [56] is to study whether it is possible, and, if yes, under which conditions Type II maximum likelihood can recover the elements of Σ_θ . It turns out that this is indeed possible, provided that N and K follow an implicit relation via the Khatri-Rao product of the matrix Φ . It turns out that model identifiability can still be guaranteed even if $k > N$. Moreover, this is possible without the enforcement of an extra constraint; note that this is not possible by minimizing the LS cost function, which is associated with the Type I maximum likelihood. Furthermore, the method recovers the true values of the elements of Σ_θ and asymptotically attains the Cramer-Rao bound. In contrast, if the conditions are violated, then a regularizing constraint, such as the ℓ_1 norm is required to guarantee that the Fisher information matrix is nonsingular.

13.11 EXPECTATION PROPAGATION

Expectation propagation is an alternative to the variational techniques for approximating posterior pdfs. The task of interest is the same as the one treated in the beginning of this chapter, in Section 13.2. Assume that we are given a set of observations \mathcal{X} , which are distributed according to $p(\mathcal{X}|\theta)$, and a prior, $p(\theta)$, corresponding to the set of the unknown parameters¹⁰ θ . The goal is to obtain an estimate of the posterior, $p(\theta|\mathcal{X})$, assuming that its computation is intractable.

Let us denote by $q(\theta)$ the estimate of the posterior. The starting point is to compute q by minimizing the Kullback-Leibler divergence,

$$\text{KL}(p||q) = \int p(\theta|\mathcal{X}) \ln \frac{p(\theta|\mathcal{X})}{q(\theta)} d\theta. \quad (13.114)$$

Note that $\text{KL}(p||q)$ is different than the $\text{KL}(q||p)$ divergence, which is involved in the bound in Eq. (13.2). Because the KL divergence is not symmetric, the two methods minimize a different cost. Before proceeding any further, it is important to highlight some implications associated with the two forms of the KL divergence.

- *I-Projection:* The $\text{KL}(q||p)$ divergence is given by

$$\text{KL}(q||p) = \int q(\theta) \ln \frac{q(\theta)}{p(\theta|\mathcal{X})} d\theta. \quad (13.115)$$

This is sometimes known as *I-projection* or *information projection*. Looking carefully at it, note that in regions of the parameter space where $p(\theta|\mathcal{X})$ assumes small values, $\text{KL}(q||p)$ gets large values and minimization pushes $q(\theta)$ to small values as well. Consider now the case that $p(\theta|\mathcal{X})$ is bimodal, while $q(\theta)$ is constrained to be unimodal. Then, minimizing $\text{KL}(q||p)$ will force q to be placed close to either of the two peaks of p in order to get small values in the regions where p takes small values, too.

¹⁰ If other hidden variables are also involved, we consider them as part of θ .

- *M-Projection:* We now turn our focus to $\text{KL}(p||q)$ divergence, defined in Eq. (13.114). This is also known as *M-projection* or *moment projection*. For the case discussed before, in the regions where p assumes large values, then $\text{KL}(p||q)$ gets large values and minimization estimates q in order to have large values in these regions, too. Thus, the estimate q is placed in order for its mode to lie somewhere between the two modes of p , as a compromise between the two. Obviously, this is not a good result, because the estimate puts high-probability mass in regions where p assumes small values. This discussion points out some limitations on the performance that the expectation propagation method is expected to exhibit in practice, because it is based on $\text{KL}(p||q)$ minimization.

We now assume that $p(\mathcal{X}, \boldsymbol{\theta})$ can be factorized, that is,

$$p(\mathcal{X}, \boldsymbol{\theta}) = \prod_j f_j(\boldsymbol{\theta}). \quad (13.116)$$

For example, such a product can cover the case where

$$p(\mathcal{X}, \boldsymbol{\theta}) = \prod_n p(\mathbf{x}_n | \boldsymbol{\theta}) p(\boldsymbol{\theta}),$$

where $p(\boldsymbol{\theta})$ is the corresponding prior. The more general formulation of the factorization, used in Eq. (13.116), can serve the needs for more general tasks, as for example graphical models to be treated in [Chapter 15](#). Thus, we can now write that

$$p(\boldsymbol{\theta} | \mathcal{X}) = \frac{1}{p(\mathcal{X})} \prod_j f_j(\boldsymbol{\theta}), \quad (13.117)$$

where $p(\mathcal{X})$ is the evidence of the model. The estimate q will be chosen to be given in a factorized form, as in the variational approach in [Section 13.2](#), that is,

$$q(\boldsymbol{\theta}) = \frac{1}{Z} \prod_j \hat{f}_j(\boldsymbol{\theta}), \quad (13.118)$$

where $\hat{f}_j(\boldsymbol{\theta})$ corresponds to $f_j(\boldsymbol{\theta})$ and Z is the normalizing constant. The next assumption is that the $q(\boldsymbol{\theta})$ is constrained to lie within the exponential family of pdfs ([Section 12.4](#)) and

$$q(\boldsymbol{\theta}) := g(\boldsymbol{\eta}) h(\boldsymbol{\theta}) \exp\left(\boldsymbol{\eta}^T \mathbf{u}(\boldsymbol{\theta})\right). \quad (13.119)$$

Minimizing the KL divergence

Plugging into Eq. (13.114) the definition in Eq. (13.119) and collecting all terms that are independent of $\boldsymbol{\eta}$ in a constant, we readily obtain

$$\text{KL}(p||q) = -\ln g(\boldsymbol{\eta}) - \int p(\boldsymbol{\theta} | \mathcal{X}) \left(\boldsymbol{\eta}^T \mathbf{u}(\boldsymbol{\theta}) \right) d\boldsymbol{\theta} + \text{constants}. \quad (13.120)$$

Taking the gradient with respect to $\boldsymbol{\eta}$ and equating to zero we get

$$-\frac{1}{g(\boldsymbol{\eta})} \nabla g(\boldsymbol{\eta}) = \mathbb{E}_p [\mathbf{u}(\boldsymbol{\theta})]. \quad (13.121)$$

However, from Eq. (13.119) we have that

$$g(\boldsymbol{\eta}) \int h(\boldsymbol{\theta}) \exp\left(\boldsymbol{\eta}^T \mathbf{u}(\boldsymbol{\theta})\right) d\boldsymbol{\theta} = 1,$$

and taking the gradient with respect to $\boldsymbol{\eta}$ results in

$$\begin{aligned} \mathbf{0} &= \nabla g(\boldsymbol{\eta}) \int h(\boldsymbol{\theta}) \exp\left(\boldsymbol{\eta}^T \mathbf{u}(\boldsymbol{\theta})\right) d\boldsymbol{\theta} \\ &\quad + g(\boldsymbol{\eta}) \int h(\boldsymbol{\theta}) \exp\left(\boldsymbol{\eta}^T \mathbf{u}(\boldsymbol{\theta})\right) \mathbf{u}(\boldsymbol{\theta}) d\boldsymbol{\theta} \end{aligned}$$

or

$$-\frac{1}{g(\boldsymbol{\eta})} \nabla g(\boldsymbol{\eta}) = \mathbb{E}_q [\mathbf{u}(\boldsymbol{\theta})],$$

which combined with Eq. (13.121) finally results in

$$\boxed{\mathbb{E}_q [\mathbf{u}(\boldsymbol{\theta})] = \mathbb{E}_p [\mathbf{u}(\boldsymbol{\theta})] : \text{ Moment Matching.}} \quad (13.122)$$

The latter is an elegant equation known as *moment matching*. It basically states that at the optimum, $q(\boldsymbol{\theta})$, the expectations of its sufficient statistics are equal to the expectations associated with the pdf to be learned. For example, if q is chosen to be a Gaussian, the sufficient statistics involves the mean and the covariance matrix. Thus, all one has to do is compute the mean and covariance with respect to $p(\boldsymbol{\theta})$ (assuming that they can be obtained) and use them to define the respective Gaussian.

The expectation propagation algorithm

We will now make use of the moment matching result to obtain the factors, $\hat{f}_j(\boldsymbol{\theta})$, one at a time. The algorithm starts from some initial estimates, $\hat{f}_j^{(0)}$. Let us assume that we are currently seeking to update factor $\hat{f}_k(\boldsymbol{\theta})$. Let $q^{(i)}(\boldsymbol{\theta})$ be the currently available estimate of $q(\boldsymbol{\theta})$, at the i th iteration.

Step 1: Remove $\hat{f}_k^{(i)}(\boldsymbol{\theta})$ from $q^{(i)}(\boldsymbol{\theta})$, and define

$$q_{/k}^{(i)}(\boldsymbol{\theta}) := \frac{q^{(i)}(\boldsymbol{\theta})}{\hat{f}_k^{(i)}(\boldsymbol{\theta})}. \quad (13.123)$$

Step 2: Define the pdf

$$\frac{1}{Z_k} f_k(\boldsymbol{\theta}) q_{/k}^{(i)}(\boldsymbol{\theta}). \quad (13.124)$$

In other words, in the current estimate $q^{(i)}(\boldsymbol{\theta})$, $\hat{f}_k^{(i)}(\boldsymbol{\theta})$ is replaced by $f_k(\boldsymbol{\theta})$, and Z_k is the corresponding normalizing constant.

Step 3: Compute the normalizing constant,

$$Z_k = \int f_k(\boldsymbol{\theta}) q_{/k}^{(i)}(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (13.125)$$

Step 4: In this step, the optimization is performed by minimizing the KL divergence,

$$\text{KL}\left(\frac{1}{Z_k} f_k(\boldsymbol{\theta}) q_{/k}^{(i)}(\boldsymbol{\theta}) || q^{(i+1)}(\boldsymbol{\theta})\right).$$

This is achieved by moment matching, and the new $q^{(i+1)}$ is defined so that the expectations of the respective sufficient statistics are matched to those of $\frac{1}{Z_k} f_k(\boldsymbol{\theta}) q_{/k}^{(i)}(\boldsymbol{\theta})$, and this operation is assumed to be computationally tractable.

Step 5: Compute $\hat{f}_k^{(i+1)}$ such that

$$\hat{f}_k^{(i+1)}(\boldsymbol{\theta}) := K \frac{q^{(i+1)}(\boldsymbol{\theta})}{q_{/k}^{(i)}(\boldsymbol{\theta})}, \quad (13.126)$$

where the proportionality constant is computed so that

$$\int \hat{f}_k^{(i+1)}(\boldsymbol{\theta}) q_{/k}^{(i)}(\boldsymbol{\theta}) d\boldsymbol{\theta} = \int f_k(\boldsymbol{\theta}) q_{/k}^{(i)}(\boldsymbol{\theta}) d\boldsymbol{\theta}, \quad (13.127)$$

which results in $K = Z_k$.

The procedure is then applied for the estimation of $\hat{f}_{k+1}^{(i+1)}$. For convergence, more than one passes have to be performed. The evidence can be approximated as

$$p(\mathcal{X}) \approx \int \prod_j \hat{f}_j(\boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (13.128)$$

A detailed application of the algorithm in the context of a simple-to-follow example is given in [42].

Remarks 13.5.

- In general, there is no guarantee that the algorithm will converge, which is a major disadvantage of the method. However, it can be shown that if the iterations do converge, the solution is a stationary point of a particular energy function [42]. Recall that in the variational Bayes approach, there is guarantee of convergence to a local optimum point. Of course, one could optimize the KL divergence for the expectation propagation method directly, which guarantees convergence, but in this case the algorithm is more complex and slow.
- Taking into account our discussion concerning the two forms of KL divergence, the expectation propagation method results in poor performance when the true posterior is multimodal. However, for other scenarios, such as logistic-type models, the expectation propagation method can offer competitive and sometimes better performance compared to the variational methods or methods built around the Laplacian approximation (see, e.g., [34, 42]).
- The expectation propagation algorithm was first proposed in [42], and it is a modification of what was known before as *assumed density filtering* (ADF) or *moment matching* (e.g., [51] and the references therein).
- Looking at the factors, $f_j(\boldsymbol{\theta})$, in a more general view, it turns out that the expectation propagation offers the vehicle for obtaining a range of message passing algorithms in the context of probabilistic graphical models (Chapter 15) [43].
- α -Divergence: Having spent time discussing in some detail the two forms of KL divergence, it is interesting to point out that both formulations can be obtained as special cases of a more general family known as the α family of divergences, defined as

$$D_\alpha(p||q) := \frac{4}{1-\alpha^2} \left(1 - \int p(x)^{(1+\alpha)/2} q(x)^{(1-\alpha)/2} dx \right) : \alpha - \text{Divergence}, \quad (13.129)$$

where $\alpha \in \mathbb{R}$ is a parameter. Note that $\text{KL}(p||q)$ is obtained at the limit $\alpha \rightarrow 1$ and $\text{KL}(q||p)$ is obtained for $\alpha \rightarrow -1$. $D_\alpha(p||q)$ is nonnegative and it becomes zero if $p = q$ (see, e.g., [2]).

13.12 NONPARAMETRIC BAYESIAN MODELING

The Bayesian approach to parametric modeling has been the focus of our attention in the current and previous chapters. The underlying assumption was that the number of the unknown parameters was fixed and finite. We now turn our attention to a more general task. We will assume that the hidden structure of our model is not fixed but is allowed to grow with the data. In other words, its complexity is not specified a priori but is left to be determined from the data. This is the reason that such models are called *nonparametric*; recall from [Chapter 3](#) that a model is called parametric if the number of free parameters is fixed and independent of the size of the data set.

We will avoid treating nonparametric Bayesian models in a mathematically rigorous sense. Such a path would take us a bit far from the purpose of this book and also from the mathematical skills of the average reader. Thus, we will be content with presenting the main concepts in a mathematically “humble” way. Once the basics have been grasped, the keen reader can delve deeper into the topic by referring to more specialized literature (see, e.g., [23]).

The idea behind nonparametric Bayesian models will be demonstrated via our familiar mixture modeling task, treated in [Section 13.4](#). There, K mixtures (clusters) were assumed; each mixture was modeled via a Gaussian pdf with unknown mean and precision matrix, (μ_k, Q_k) , $k = 1, 2, \dots, K$. These, in turn, were considered as random entities and were dressed up with a prior—a Gaussian pdf for the mean and a Wishart one for the precision matrix. The probabilities, P_k , $k = 1, 2, \dots, K$, for each mixture were treated either as constants, which were optimized during the M-step, or they were considered random variables and a Dirichlet pdf prior was associated with them ([Problem 13.7](#)). The goal was to obtain an estimate of the *posterior probabilities* of the labels associated with each observation point, $P(k_n|\mathcal{X})$. There, we resorted to variational techniques and the mean field approximation and the posterior was approximated by the function $q_z(\mathcal{Z}) = q_z(z_1, \dots, z_N)$, where z_n , $n = 1, 2, \dots, N$, were $0 - 1$ coding vectors, with a one placed at the location corresponding to a specific mixture (k) and the rest of the elements being zero. In this way, an equivalent clustering of the observation points was achieved in K Gaussian-distributed clusters.

The nonparametric counterpart of the previous task is expressed in almost the same way with a single important difference. The number of mixtures, K , is not fixed to a finite value; as a matter of fact, the number of mixtures is allowed to be *countably infinite*. There are two questions that now pop up: (a) how can one deal with an infinite number of clusters, and (b) how can one deal with a prior related to infinite many probability values?

To give an indication of how to deal with infinity, recall that in nonparametric modeling the number of data points is still finite and equal to N . Thus, whatever model one adopts, there is no way of having more than N mixtures (clusters); the latter corresponds to the worst case scenario, where each one of the points belongs to a different cluster. Hence, although in theory one can have infinite many clusters, only a finite subset of them is *nonempty*. Thus, all we need to do is obtain an explicit representation of the nonempty mixture components.

Concerning the second question, it can be shown that a prior distribution over an infinite number of groupings, $P(\mathcal{Z})$, that favors assigning data to a small number of groups, is the *Chinese restaurant*

process (CRP); this is a distribution over infinite partitions of the integers, [1, 19, 55]. An easy way to follow the steps leading to CRP as a limiting case ($K \rightarrow \infty$) of the finite mixture modeling is given in [64].

13.12.1 THE CHINESE RESTAURANT PROCESS

The name draws from the seemingly infinite number of tables in some very large Chinese restaurants in California. Each table is associated with one cluster/mixture and each customer with one observation. The first customer sits at the first table. The second customer sits at the first table with probability $\frac{1}{1+\alpha}$ and at a new table with probability $\frac{\alpha}{1+\alpha}$. The n th customer sits at one of the previously occupied tables with a probability proportional to the people who already sit at it, and he/she sits at a new table with a probability proportional to α . The parameter α is known as the *concentration parameter*. The larger its value, the more tables are occupied and the fewer the customers who sit at a single table. In a more formal way, let k_n denote the table for the n th customer. Then, we can write

$$P(k_n = k | k_{1:n-1}) = \begin{cases} \frac{n_k}{n - 1 + \alpha}, & \text{if } k \leq K_{n-1}, \\ \frac{\alpha}{n - 1 + \alpha}, & \text{otherwise,} \end{cases} \quad (13.130)$$

where K_{n-1} is the number of tables occupied by the previously $n - 1$ arrived customers and n_k the number of customers already sitting at table k . It can be shown that the expected number of occupied tables grows as $\alpha \ln n$; that is, the expected number of clusters grows with the number of data (e.g., [16]). The rule in Eq. (13.130) provides the sampling philosophy for assigning data to clusters as new data arrive sequentially. It can be shown (see, e.g., [16]) that the resulting probability $P(k_1, \dots, k_n)$ is independent (up to label changes) of the sequence in which data arrive; this is an important invariance property.

13.12.2 INFERENCE

Having defined an appropriate prior over the partitions (clusters), the next goal is to perform inference and compute the corresponding posterior, $P(k_1, \dots, k_N | \mathcal{X})$; this step basically reveals the clustering structure by telling us which latent structure is more likely to have generated the data. This is not a computationally tractable task, and one has to resort either to variational approximation methods, for example, [9, 33], or to Monte Carlo techniques, to be discussed in Chapter 14. For the latter case, the CRP model is particularly convenient for Gibbs sampling (Section 14.9); see, for example, [49]. For variational inference, a different representation of Dirichlet processes seems more appropriate (Section 13.12.4).

13.12.3 DIRICHLET PROCESSES

This section provides a brief discussion of the more general mathematical framework in which CRPs belong. This section can be bypassed in a first reading.

The notion of stochastic processes was introduced in Chapter 2, Section 2.4. The *Dirichlet process* (DP), G , is a distribution of distributions and it is defined in terms of (a) the *concentration parameter*, α ; and (b) the *base distribution*, G_0 , over a space Θ , and we write $G \sim \text{DP}(\alpha, G_0)$. We say that

$G \sim \text{DP}(\alpha, G_0)$ is a Dirichlet process, if for *any* partition¹¹ T_i , $i = 1, 2, \dots, K$ of Θ , that is, $\Theta = \cup_{i=1}^K T_i$, the following holds true:

$$(G(T_1), \dots, G(T_K)) \sim \text{Dir}(\alpha G_0(T_1), \dots, \alpha G_0(T_K)), \quad (13.131)$$

where $G_0(T_i)$ is the probability mass corresponding to T_i . In other words, $G(T_i)$, $i = 1, 2, \dots, K$, are jointly distributed according to a Dirichlet distribution (Chapter 2). Dirichlet processes were defined by Ferguson [13]. Two important theorems were proved there:

- Random distributions drawn from a DP, $G \sim \text{DP}(\alpha, G_0)$, are *discrete*. That is, they place their probability mass on a *countably* infinite number of points, θ_k , $k \in \mathbb{Z}$. Thus, we can write

$$G = \sum_{k=1}^{\infty} P_k \delta_{\theta_k}(\theta). \quad (13.132)$$

The points θ_k are called the *atoms* and are i.i.d. drawn from the base distribution, $\theta_k \sim G_0$, and P_k are the corresponding probabilities.

- Due to the discrete nature of the atoms, if we keep sampling we are going to get more and more repetitions of the previously drawn samples. This leads to a clustering structure, which is associated with DPs. Moreover, the structure of the shared values defines a partition of the integer set whose distribution is a Chinese restaurant process (CRP).

13.12.4 THE STICK-BREAKING CONSTRUCTION OF A DP

Besides CRP, the *stick-breaking representation* of a DP was developed in [67]. This representation is the one that is usually used in the variational inference approach to nonparametric mixture modeling (e.g., [9]).

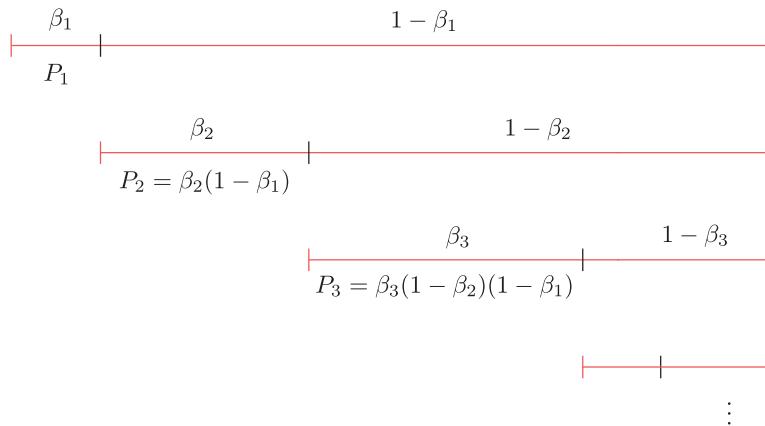
Consider a stick of unit length, Figure 13.14. The stick will be divided into a sequence of infinite many segments, of length P_k , $k = 1, 2, \dots$, according to the following algorithm. First, choose a beta (Chapter 2) distributed variable, $\beta_1 \sim \text{Beta}(\beta|1, \alpha)$, and break off a segment of the stick of length equal to β_1 . Then, choose another variable, $\beta_2 \sim \text{Beta}(\beta|1, \alpha)$, and break off another segment of length β_2 . Then, the k th step of the algorithms can be written as

$$\begin{aligned} \beta_k &\sim \text{Beta}(\beta|1, \alpha), \\ P_k &= \beta_k \prod_{i=1}^{k-1} (1 - \beta_i), \quad k = 1, 2, 3, \dots \end{aligned}$$

Using the resulting from the algorithm sequence of P_k , a random distribution is formed according to Eq. (13.132), with θ_k drawn i.i.d. from G_0 . It can be shown that the distribution of this random distribution is a DP, $G \sim \text{DP}(\alpha, G_0)$.

We will return to the DPs in Chapter 19, in the context of factor analysis for latent variables modeling. A concise tutorial concerning DPs can also be found in [16]. There, a number of sites with publicly available software tools are also provided.

¹¹ Strictly speaking, we should say measurable partition; a partition is measurable if it is closed under complementation and countable union.

**FIGURE 13.14**

The stick-breaking construction of a DP.

Example 13.5. This example illustrates the computational evolution of the variational inference method in [9] for a two-dimensional Gaussian mixture model based on the Chinese restaurant process. The data are generated according to five separate Gaussian distributions, with parameters

$$\boldsymbol{\mu}_1 = [-12.5, 2.5]^T, \boldsymbol{\mu}_2 = [-4, -0.1]^T, \boldsymbol{\mu}_3 = [2, -3.5]^T,$$

$$\boldsymbol{\mu}_4 = [10, 8]^T, \boldsymbol{\mu}_5 = [3, 3]^T,$$

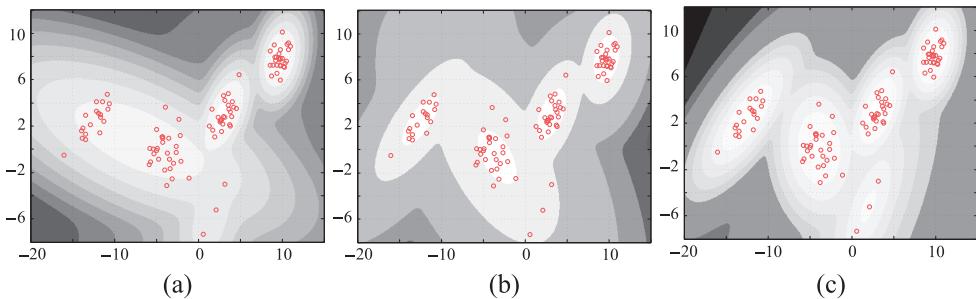
and

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix} 1.4 & 0.81 \\ 0.81 & 1.3 \end{bmatrix}, \quad \boldsymbol{\Sigma}_2 = \begin{bmatrix} 1.5 & 0.2 \\ 0.2 & 2.1 \end{bmatrix}, \quad \boldsymbol{\Sigma}_3 = \begin{bmatrix} 1.6 & 1 \\ 1 & 2.9 \end{bmatrix},$$

$$\boldsymbol{\Sigma}_4 = \begin{bmatrix} 0.5 & 0.22 \\ 0.22 & 0.8 \end{bmatrix}, \quad \boldsymbol{\Sigma}_5 = \begin{bmatrix} 1.5 & 1.4 \\ 1.4 & 2.4 \end{bmatrix},$$

for the means and covariance matrices, respectively. One hundred data points were generated from the Gaussian mixture, where each Gaussian was assigned an arbitrary number of points. Figure 13.15 depicts the data points as red circles. Variational inference on the model was performed based on the Matlab implementation of the method¹² in [9]. A data set comprising equidistant, closely spaced test points in the area $[-20, 15] \times [-8, 12]$ was used to compute the approximate predictive distribution estimated by the variational inference method. The contours of the predictive distribution computed during the first, second, and fifth iterations of the algorithm are plotted in Figure 13.15. The algorithm has clearly identified the clusters of the data.

¹² <http://sites.google.com/site/kenichikurihara/academic-software>

**FIGURE 13.15**

Contours of predictive distribution for Example 13.5, after (a) the first, (b) the second, and (c) the fifth iteration.

13.13 GAUSSIAN PROCESSES

In Section 13.12, the way to impose priors onto the model was similar in spirit with that used for parametric modeling techniques; that is, priors were imposed on the set of unknown parameters. In this section, a different rationale will be adopted. The prior will be placed directly over the space of nonlinear functions, rather than specifying a parametric family of nonlinear functions and placing priors over their parameters.

Let us recall the nonlinear regression task given in Eq. (12.1), that is,

$$y = \theta_0 + \sum_{k=1}^{K-1} \theta_k \phi_k(\mathbf{x}) + \eta = \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}) + \eta, \quad (13.133)$$

where the parameters, $\boldsymbol{\theta}$, are treated as a random vector. Let us define

$$\mathbf{f}(\mathbf{x}) = \boldsymbol{\theta}^T \boldsymbol{\phi}(\mathbf{x}),$$

where $\mathbf{f}(\mathbf{x})$ is a *random process*. From Chapter 2, we know that a random process is a random entity whose realization (the outcome of an experiment) is a function, $f(\mathbf{x})$, instead of a single value. The idea that spans this section is to work directly on $\mathbf{f}(\mathbf{x})$ instead of the indirect approach of modeling it via the set of parameters, $\boldsymbol{\theta}$. This is not the first time we have adopted such a path. We silently did it in Chapter 11 while searching for functions in RKHS spaces. As a matter of fact, this section can be considered a bridge between the current chapter and Chapter 11.

Recall from Chapter 11 that instead of expanding an unknown function in parameterized form in terms of a number of *preselected* basis functions as in Eq. (13.133), we preferred to search directly for functions that reside in an RKHS; the optimization was carried out with respect to the function itself (not with respect to a set of parameters). In the context of the LS cost function, the optimization was cast as

$$\min_{f \in \mathbb{H}} \sum_{n=1}^N (y_n - f(\mathbf{x}_n))^2 + C \|f\|^2,$$

where $\|\cdot\|$ denotes the norm in \mathbb{H} . The goal in this section is to state the “Bayesian counterpart” to this approach. To this end, we will focus on a specific family of processes, known as Gaussian processes, proposed in [50].

Definition 13.1. A random process, $f(\mathbf{x})$, is called a *Gaussian process* (GP) iff for *any* finite number of points, $\mathbf{x}_{(1)}, \dots, \mathbf{x}_{(N)}$, the respective joint probability density function, $p(f(\mathbf{x}_{(1)}), \dots, f(\mathbf{x}_{(N)}))$, is Gaussian.

We know that a set of jointly Gaussian distributed random variables is fully described by the respective mean value and the covariance matrix. In a similar spirit, a Gaussian process is fully determined by its mean value and its *covariance function*, that is,

$$\mu_x = \mathbb{E}[f(\mathbf{x})], \quad \text{cov}_f(\mathbf{x}, \mathbf{x}') = \mathbb{E}[(f(\mathbf{x}) - \mu_x)(f(\mathbf{x}') - \mu_{x'})].$$

A Gaussian process is said to be *stationary* if $\mu_x = \mu$ and its covariance function is of the form (see also [Chapter 2](#))

$$\text{cov}_f(\mathbf{x}, \mathbf{x}') = \text{cov}_f(\mathbf{x} - \mathbf{x}').$$

In addition, if $\text{cov}_f(\cdot, \cdot)$ depends on the *magnitude* of the distance between \mathbf{x} and \mathbf{x}' (i.e., $\|\mathbf{x} - \mathbf{x}'\|$), the Gaussian process is called *homogeneous*. From now on, we will assume $\mu_x = \mathbf{0}$. Before we proceed further, let us establish another connection with [Chapter 11](#).

13.13.1 COVARIANCE FUNCTIONS AND KERNELS

For any N and *any* collection of N points, $\mathbf{x}_{(1)}, \dots, \mathbf{x}_{(N)}$, the respective covariance matrix is defined by

$$\Sigma = \mathbb{E}[\mathbf{f}\mathbf{f}^T],$$

where

$$\mathbf{f} := [f(\mathbf{x}_{(1)}), \dots, f(\mathbf{x}_{(N)})]^T, \tag{13.134}$$

with elements given by

$$[\Sigma]_{ij} = \text{cov}_f(\mathbf{x}_{(i)}, \mathbf{x}_{(j)}), \quad i, j = 1, 2, \dots, N.$$

Because Σ is a positive semidefinite matrix, this guarantees that the covariance function is a *kernel* function ([Section 11.5.1](#)). To stress this, from now on we will use the notation

$$\text{cov}_f(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}, \mathbf{x}'),$$

and the covariance matrix becomes the corresponding *kernel matrix* denoted as \mathcal{K} ([Chapter 11](#)). This change of notation will make the connections with RKH spaces readily spotted. Some typical examples of kernel functions used for Gaussian processes are

- *Linear Kernel:*

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'.$$

Note that this kernel does not correspond to a stationary process.

- *Squared Exponential or Gaussian Kernel:*

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2h^2}\right),$$

where h is a parameter determining the *length scale* of the process. The smaller the value of h , the larger the “statistical” similarity (stronger correlation) of two points having a distance $d = \|\mathbf{x} - \mathbf{x}'\|$ apart.

- *Ornstein - Uhlenbeck Kernel:*

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|}{h}\right).$$

- *Rational Quadratic Kernel:*

$$\kappa(\mathbf{x}, \mathbf{x}') = \left(1 + \|\mathbf{x} - \mathbf{x}'\|^2\right)^{-\alpha}, \quad \alpha \geq 0.$$

Recall from [Chapter 2](#), where random processes were first presented, that a stationary covariance function/kernel has as its Fourier transform the power spectrum of the respective random process; by definition, the power spectrum of a process is a nonnegative function in the frequency domain. This suggests a way of constructing kernels for random processes; that is, take the inverse Fourier transform of a positive function in the frequency domain. Moreover, in principle, all the rules for constructing kernels, which are discussed in [Section 11.5.2](#), can also be applied to construct covariance functions. For example, a popular choice of a kernel for a Gaussian process is

$$\kappa(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \theta_1 \exp\left(-\sum_{m=1}^M \frac{(x_i - x'_i)^2}{2h_i^2}\right) + \theta_2,$$

where θ_1, θ_2 are hyperparameters, which define the process.

[Figure 13.16a](#) shows examples of different realizations of a stationary Gaussian process with $h = 2$, and [Figure 13.16b](#) for $h = 0.2$.

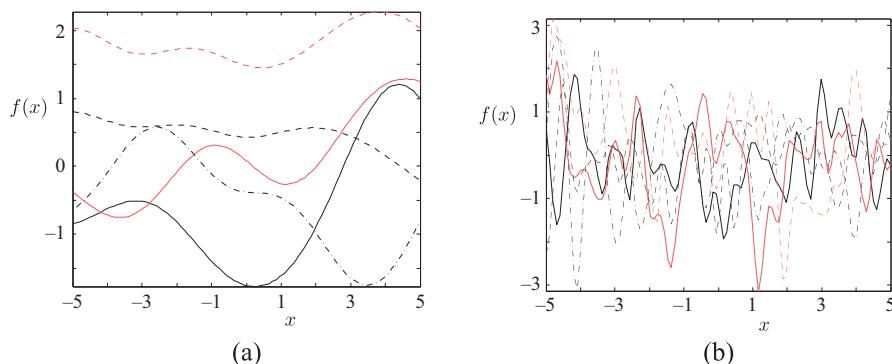


FIGURE 13.16

Different realizations of a Gaussian process. (a) Gaussian covariance kernel $h = 2$, (b) $h = 0.2$. Note that when the correlation function fades away fast, the graph of the respective realizations shows a fast variation as a function of the free variable (x).

13.13.2 REGRESSION

Let us assume that we are given a set \mathcal{X} of input observations, $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$. Recall from [Section 12.2](#) that the main goal in a Bayesian regression task is to obtain the two pdfs,

$$p(\mathbf{y}|\mathcal{X}) \quad \text{and} \quad p(\mathbf{y}|\mathbf{x}, \mathbf{y}, \mathcal{X}),$$

where

$$\mathbf{y} = \mathbf{f} + \boldsymbol{\eta}, \quad \mathbf{y} := [\mathbf{y}_1, \dots, \mathbf{y}_N]^T, \quad (13.135)$$

and

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) + \boldsymbol{\eta},$$

and \mathbf{f} is defined in Eq. (13.134). The first of the two pdfs is the joint probability density of the output variables, which are generated by input points in \mathcal{X} ; the associated randomness is due to \mathbf{f} as well as to the noise $\boldsymbol{\eta}$. The second pdf refers to the prediction of the value of \mathbf{y} , given the value of \mathbf{x} and the training data $(\mathbf{y}_n, \mathbf{x}_n), n = 1, 2, \dots, N$. We will drop out \mathcal{X} to unclutter notation, as we did in [Section 12.2](#).

Assuming $\mathbf{f}(\cdot)$ to be a zero-mean Gaussian process, then \mathbf{f} is jointly Gaussian with zero mean and covariance matrix \mathcal{K} , dictated by the covariance function/kernel $\kappa(\cdot, \cdot)$, that is,

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathcal{K}).$$

Also, let $\boldsymbol{\eta}$ be of zero mean with covariance matrix Σ_η and independent of $\mathbf{f}(\cdot)$; without harming generality, let $\Sigma_\eta = \sigma_\eta^2 I$. Thus,

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{y}|\mathbf{f}, \sigma_\eta^2 I).$$

Then, following exactly the same arguments as in [Section 12.2](#), we obtain

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathcal{K} + \sigma_\eta^2 I). \quad (13.136)$$

This is also obvious from the fact that the sum of two independent Gaussian variables is also Gaussian and the mean and covariance matrix can directly be obtained from Eq. (13.135).

To obtain $p(\mathbf{y}|\mathbf{x}, \mathbf{y})$ we can use (13.136) and apply it recursively. It will also be useful here to bring into the notation the number of available observations, N , explicitly and write

$$\mathbf{y}_{N+1} = \begin{bmatrix} \mathbf{y} \\ \mathbf{y}_N \end{bmatrix}, \quad \mathbf{y}_N := [\mathbf{y}_1, \dots, \mathbf{y}_N]^T.$$

From Eq. (13.136), \mathbf{y}_{N+1} follows a Gaussian distribution

$$p(\mathbf{y}_{N+1}|\mathbf{0}, \Sigma_{N+1}),$$

with

$$\Sigma_{N+1} := \mathcal{K}_{N+1} + \sigma_\eta^2 I_{N+1}.$$

Then from Bayes theorem, we have

$$p(\mathbf{y}|\mathbf{y}_N) = \frac{p(\mathbf{y}_{N+1})}{p(\mathbf{y}_N)}. \quad (13.137)$$

However, because the joint pdf is Gaussian, the conditional in Eq. (13.137) is also Gaussian. The respective mean and variance are computed by partitioning the matrix Σ_{N+1} (see Section 12.9, Eqs. (12.130) and (12.131)),

$$\Sigma_{N+1} = \begin{bmatrix} \kappa(\mathbf{x}, \mathbf{x}) + \sigma_\eta^2, & \kappa^T(\mathbf{x}) \\ \kappa(\mathbf{x}), & \Sigma_N \end{bmatrix}, \quad \kappa(\mathbf{x}) := [\kappa(\mathbf{x}, \mathbf{x}_1), \dots, \kappa(\mathbf{x}, \mathbf{x}_N)]^T$$

and

$$\boxed{\begin{aligned} \mu_y(\mathbf{x}) &= \kappa^T(\mathbf{x}) \Sigma_N^{-1} \mathbf{y}, \\ \sigma_y^2(\mathbf{x}) &= \sigma_\eta^2 + \kappa(\mathbf{x}, \mathbf{x}) - \kappa^T(\mathbf{x}) \Sigma_N^{-1} \kappa(\mathbf{x}). \end{aligned}} \quad (13.138)$$

Compare Eq. (13.138) with Eq. (11.27). Taking into account that $\Sigma_N = \mathcal{K}_N + \sigma_\eta^2 I$, $\mu_y(\mathbf{x})$ is identical to \hat{y} obtained by the kernel ridge regression, for appropriate choices of C and σ_η^2 . However, now we have also obtained information concerning the respective variance of the resulting estimate.

At this point, it is interesting to look back at the Bayesian regression task for parametric modeling in Section 12.2.3, and to remember that the obtained mean value in Eq. (12.20) was the same (for a zero mean prior $p(\boldsymbol{\theta})$) as that provided by the ridge regression, for an appropriate choice of λ .

Remarks 13.6.

- From the previous discussion it is apparent that solving the regression task by resorting to Gaussian processes is the Bayesian answer to solving a regression task in an RKHS. Both approaches share a common advantage. Although the underlying mapping to an RKHS (implied by the adopted kernel) may live in a high-dimensional space, the complexity for solving the task depends on the number of training points, N . The source of complexity associated with the Gaussian processes is the inversion of the matrix, which amounts to $\mathcal{O}(N^3)$ operations.
- Both equations in Eq. (13.138) can be obtained from the corresponding equation derived for the linear case of Bayesian learning, covered in Section 12.2.3. Indeed, setting $\boldsymbol{\theta}_0 = \mathbf{0}$ in Eq. (12.27) and combining it with Eq. (12.22), we obtain

$$\mu_y(\mathbf{x}) = \sigma_\theta^2 \mathbf{x}^T X^T \left(\sigma_\eta^2 I + \sigma_\theta^2 X X^T \right)^{-1} \mathbf{y}, \quad (13.139)$$

where X has replaced Φ , because the linear case is treated. Applying now the kernel trick, as discussed in Chapter 11, to replace $\sigma_\theta^2 \mathbf{x}_i^T \mathbf{x}_j$ with a kernel $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ operation, one readily obtains the corresponding equation in Eq. (13.138).

In a similar way, one can obtain $\sigma_y^2(\mathbf{x})$ in Eq. (13.138) from Eq. (12.23) by using Woodbury's formula for matrix inversion from Appendix A.1 to reformulate Eq. (12.23) according to Eq. (12.28) (try it).

Dealing with hyperparameters

As we have already stated, the kernel function can be given in terms of some parameters, say, $\boldsymbol{\theta}$, which in turn have to be estimated from the data. There are various ways to deal with this task. The first that comes to mind is to optimize the resulting parameterized log-likelihood, $\ln p(\mathbf{y}; \boldsymbol{\theta})$, with respect to $\boldsymbol{\theta}$, by taking the gradient and equating to zero. Another way is to assume a prior on the parameters and use Bayesian arguments to integrate them out. The integration is usually intractable and approximate techniques must be used, for example, Monte Carlo methods, Chapter 14. Needless to say that both techniques have their

drawbacks. Optimizing the log-likelihood is a nonconvex task that cannot guarantee, in general, a global maximum. On the other hand, Monte Carlo techniques tend to be computationally intensive, requiring many iterations to converge. More on these issues can be found in [58].

Computational considerations

In order to reduce the $\mathcal{O}(N^3)$ computational load associated with the inversion of Σ_N , a number of approximate techniques have been proposed. A possible path is the *sparse GPs*; in these methods, the full Gaussian process model is approximated by using an expansion in terms of a finite set of basis functions. For example, it is common to use as bases the set $\kappa(\mathbf{x}, \mathbf{u}_m)$, where $\mathbf{u}_m, m = 1, 2, \dots, M \ll N$, is a subset of the input samples known as *active set*. Such techniques can lead to a reduced cost of the order of $\mathcal{O}(M^2N)$ (e.g., [57]). Other alternatives that do not require the active set to be a subset of the training samples have also been proposed (e.g., [35, 69]). In [77], a variational sparse method is proposed that attempts to alleviate problems encountered when one increases the size of the active set.

A variation of the Gaussian processes approach is to equip it with the ability to forget past samples for time-varying environments; this method has been proposed in [54, 78] as an alternative to the kernel RLS algorithm discussed in [Chapter 11](#). Other variants use transformations of the output variables to make Gaussian models applicable to a wider range of problems, [36, 68].

In [62], the connection between Gaussian processes and Kalman filtering is exploited and the solution is obtained via the involvement of stochastic differential equations, which makes the dependence of the complexity on time to be linear.

13.13.3 CLASSIFICATION

In contrast to the regression task, under the Gaussian assumptions for the noise and the involved random process, the classification task gets more involved. In [Section 13.7](#), the logistic regression in its parametric form, given in Eq. (13.67), was employed. In the context of the Gaussian processes, the model becomes

$$P(\omega_1 | \mathbf{x}) = \frac{1}{1 + \exp(-f(\mathbf{x}))} = \sigma(f(\mathbf{x})),$$

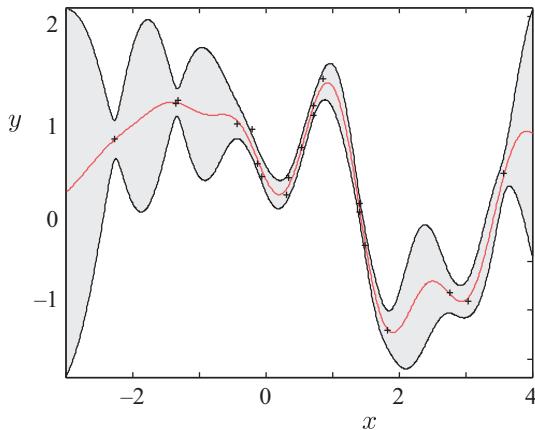
where now $f(\mathbf{x})$ will be treated in terms of a Gaussian random process, associated with a kernel function $\kappa(\cdot, \cdot)$. Given a set of training samples, $(y_n, \mathbf{x}_n), n = 1, 2, \dots, N, y_n \in \{0, 1\}$, and following the same arguments as in [Section 12.3](#), we can now write that

$$P(\mathbf{y}|\mathbf{f}) = \prod_{n=1}^N \sigma(f_n)^{y_n} (1 - \sigma(f_n))^{1-y_n},$$

where $f_n := f(\mathbf{x}_n)$, and

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{f} | \mathbf{0}, \mathcal{K}).$$

Note that $P(\mathbf{y}|\mathbf{f})$ is no longer Gaussian and the involved integrations needed to obtain $P(\mathbf{y})$ and/or $P(\mathbf{y}|\mathbf{x}, \mathbf{y})$ cannot be performed analytically. There are various ways to perform approximations. One path is to resort to the Laplacian approximation of $p(f(x)|y)$ (see [Section 12.3](#)) [79]. Another is to use Monte Carlo techniques [47]. In [17], a variational approach has been used to obtain bounds on the logistic sigmoid and approximate the respective product with a product of Gaussians. The expectation propagation method has been used in [51].

**FIGURE 13.17**

The red line corresponds to the mean of the posterior Gaussian process. The shaded area corresponds to \pm twice the standard deviation.

For further reading on Gaussian processes, the interested reader may consult the classical reference [58].

Example 13.6. The goal of this example is to demonstrate the usage of Gaussian processes in regression. To this end, $N = 20$ points were randomly sampled from a realization of a Gaussian process, with zero mean and covariance function based on the Gaussian kernel with length scale $h = 0.5$. The corresponding input points were drawn according to a normal distribution of zero mean and unit variance. In the sequel, Gaussian noise was added to these GP points, with variance 0.01, to form the set of observed data (shown as ‘+’ in Figure 13.17). Using these as the training data, predictions of the output variables, corresponding to $D = 1000$ equidistant input points in the interval $[-3, 4]$, were performed; for the prediction, the expressions for the posterior GP mean and variance in Eq. (13.138) were used. The mean of the posterior Gaussian process is illustrated in Figure 13.17 as a solid red line. The shaded area surrounding the curve of the posterior mean corresponds to the error bars $\mu_y \pm 2\sigma_y$ of the posterior prediction. Notice the increase of the posterior prediction variance in regions where observed data points are scarce.

13.14 A CASE STUDY: HYPERSPECTRAL IMAGE UNMIXING

Hyperspectral image unmixing (HSI) is a typical application of sparse regression modeling under a set of constraints. It is a good “excuse” for us to demonstrate the application of the hierarchical Bayesian modeling approach via a task of great practical importance.

In *hyperspectral remote sensing*, the electromagnetic solar energy emanating from the earth’s surface is measured by sensitive scanners located aboard a satellite, an aircraft, or a space station. The scanners are sensitive to a number of wavelength bands of the electromagnetic radiation. Different properties of the earth’s surface contribute to the reflection of the energy in the different bands.

For example, in the visible-infrared range, properties such as the mineral and moisture contents of soils, the sedimentation of water, and the moisture content of vegetation are the main contributors to the reflected energy. In contrast, at the thermal end of the infrared, it is the thermal capacity and thermal properties of the surface that contribute to the reflection. Thus, each band measures different properties of the same patch of the earth's surface. In this way, images of the earth's surface corresponding to the spatial distribution of the reflected energy in each band can be created. The task now is to exploit this information in order to identify the various ground cover types, that is, built-up land, agricultural land, forest, fire burn, water, diseased crop, and so on.

[Figure 13.18](#) illustrates the process of generating a pixel's spectral signature out of a hyperspectral image data cube (the cube consists of two spatial and one spectral dimension). Each image corresponds to a single wavelength (band) and each pixel to a specific patch of the earth's surface. The *spectral signature* of a pixel is simply a vector containing radiance values measured in the various spectral bands. Technological advances in recent years have allowed the implementation of imaging spectrometers, which have the ability to collect data in hundreds of adjacent spectral bands. The highly increased volume of data conveys spatial/spectral information that can be properly exploited to accurately determine the type and nature of the objects being imaged.

An intimate limitation of hyperspectral remote sensing is that a single pixel often records a mixed spectral signature of different distinct materials, due to the low spatial resolution of the remote sensor. This raises the need for spectral unmixing (SU) [32], which is a very important step in hyperspectral image processing that has recently attracted strong scientific interest. SU is the procedure of decomposing the measured spectrum of an observed pixel into a collection of constituent spectral

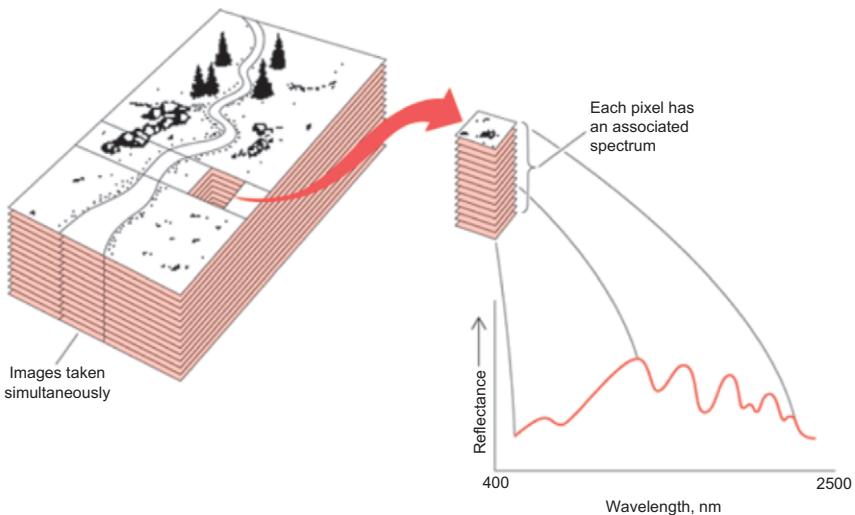


FIGURE 13.18

Each image corresponds to a specific wavelength band and each pixel to a particular patch of the earth's surface. The signature of a pixel is a vector whose coefficients measure the radiance of the respective earth's patch in the different bands (modified image taken from [\[61\]](#)).

signatures (or *endmembers*) and their corresponding proportions (or *abundances*). A widely used model to perform SU is the linear mixing model.

Assume a remotely sensed hyperspectral image consisting of M spectral bands, and let $\mathbf{y} \in \mathbb{R}^M$ be the vector containing the measured spectral signature (i.e., the radiance values in all spectral bands) of a single pixel (specific earth patch). Also let $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_l]$ stand for the $M \times l$ endmember signature matrix, where $\mathbf{x}_i \in \mathbb{R}^M$, $i = 1, 2, \dots, l$, comprises the spectral signatures of the i th endmember, and l is the total number of (possible) distinct endmembers (earth-surface/material types) present in the scene. Finally, let $\boldsymbol{\theta} = [\theta_1, \theta_2, \dots, \theta_l]^T$ be the *abundance vector* associated with \mathbf{y} , where θ_i denotes the abundance fraction of \mathbf{x}_i in \mathbf{y} . The linear mixing model assumes that there is a linear relationship between the spectra of the measured pixel and the endmembers, expressed as

$$\mathbf{y} = X\boldsymbol{\theta} + \boldsymbol{\eta} \quad (13.140)$$

where $\boldsymbol{\eta}$ stands for the additive noise values, which are assumed to be samples of a zero-mean Gaussian distributed random vector, with (i.i.d.) elements, that is, $\boldsymbol{\eta} \sim \mathcal{N}(\boldsymbol{\eta}|\mathbf{0}, \beta^{-1}I_M)$, where β denotes the inverse of the noise variance (precision), and I_M is the $M \times M$ identity matrix. Note that the model in Eq. (13.140) is a typical regression model in its multivariate formulation, because now the output for each measurement is a vector and not a scalar (see also Section 4.9 of Chapter 4). The output variables are measured and the matrix X is assumed known, and indeed there are methods to estimate its elements.

Treating such a model to recover the abundance coefficients would be a straightforward application of what has been said so far in the current and previous chapters of this book. However, there is a physical constraint that has to be considered and that makes the task more interesting. The abundance coefficients are nonnegative, that is,

$$\theta_i \geq 0, \quad i = 1, 2, \dots, l. \quad (13.141)$$

Additionally, a valid assumption is that only a few of the endmembers present in the image will contribute to the spectrum of a single pixel \mathbf{y} . In other words, the abundance vector $\boldsymbol{\theta}$ accepts a *sparse* representation in X .

Thus, our goal is to estimate $\boldsymbol{\theta}$ subject to the nonnegativity as well as the sparsity constraints, given the spectral measurements, \mathbf{y} , and the endmember matrix, X . Obviously, there are different paths to achieve this goal. Because we are currently exploring the Bayesian world, we will employ the Bayesian framework. To this end, an appropriate prior model that expresses our prior belief on the parameters of interest will be first be adopted, and we will then perform Bayesian inference using the variational Bayes methodology, as has been previously discussed.

13.14.1 HIERARCHICAL BAYESIAN MODELING

The presence of Gaussian noise in Eq. (13.140) dictates that

$$\begin{aligned} p(\mathbf{y}|\boldsymbol{\theta}, \beta) &= \mathcal{N}(\mathbf{y}|X\boldsymbol{\theta}, \beta^{-1}I_M) \\ &= (2\pi)^{-\frac{M}{2}} \beta^{\frac{M}{2}} \exp\left(-\frac{\beta}{2}\|\mathbf{y} - X\boldsymbol{\theta}\|^2\right). \end{aligned} \quad (13.142)$$

We now turn our attention to selecting suitable priors for the model parameters, which are treated as random variables, Θ , β . As a prior for the nonnegative noise precision β we adopt a Gamma distribution (Section 13.3, Eq. (13.24)), expressed as

$$p(\beta) = \text{Gamma}(\beta|c, d) = \frac{d^c}{\Gamma(c)} \beta^{c-1} \exp(-d\beta), \quad (13.143)$$

where c and d are the respective parameters (set equal to 10^{-6} in the experiments).

For the abundance vector Θ , we define a two-level hierarchical prior that is expressed in a conjugate form and imposes sparsity as well as nonnegativity on the abundance coefficients. Inspired by [60], a nonnegatively truncated Gaussian prior is selected, that is,

$$p(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \mathcal{N}_{\mathbb{R}_+^l}(\boldsymbol{\theta}|\mathbf{0}, A^{-1}), \quad (13.144)$$

where $\boldsymbol{\alpha} := [\alpha_1, \alpha_2, \dots, \alpha_l]^T$ is the precision parameter vector, $A = \text{diag}\{\alpha_1, \dots, \alpha_l\}$ is the corresponding diagonal matrix, and $\mathcal{N}_{\mathbb{R}_+^l}$ signifies the l -variate normal distribution truncated at the nonnegative orthant of \mathbb{R}^l , denoted by \mathbb{R}_+^l [71]. In the second level of hierarchy, the precision parameters are also considered random variables, α_i 's, $i = 1, 2, \dots, l$, that follow an inverse Gamma distribution, that is,

$$p(\alpha_i) = \text{IGamma}\left(\alpha_i|1, \frac{b_i}{2}\right) = \frac{b_i}{2} \alpha_i^{-2} \exp\left(-\frac{b_i}{2} \frac{1}{\alpha_i}\right), \quad (13.145)$$

where $b_i, i = 1, 2, \dots, N$, are scale hyperparameters. These two levels of hierarchy form a nonnegatively truncated multivariate Laplace prior over the abundance vector Θ , which can be established by integrating out the precision $\boldsymbol{\alpha}$ [71], that is,

$$p(\boldsymbol{\theta}|\mathbf{b}, \beta) = \prod_{i=1}^l \sqrt{\beta b_i} \exp\left(-\sqrt{\beta b_i} |\theta_i|\right) I_{\mathbb{R}_+^l}(\boldsymbol{\theta}), \quad (13.146)$$

where $I_{\mathbb{R}_+^l}(\boldsymbol{\theta})$ is the indicator function, with $I_{\mathbb{R}_+^l}(\boldsymbol{\theta}) = 1$ (resp. 0) if $\boldsymbol{\theta} \in \mathbb{R}_+^l$ (resp. $\boldsymbol{\theta} \notin \mathbb{R}_+^l$). In our formulation, the sparsity-promoting scale hyperparameters in Eq. (13.145) are also assumed to be random and are inferred from the data, by assuming the following Gamma prior distribution for each $b_i, i = 1, 2, \dots, l$,

$$p(b_i) = \text{Gamma}(b_i|\kappa, \nu) = \frac{\nu^\kappa}{\Gamma(\kappa)} b_i^{\kappa-1} \exp(-\nu b_i). \quad (13.147)$$

Hyperparameters κ and ν in Eq. (13.147) are also set to small values (10^{-6} in the experiments).

Having adopted the hierarchical Bayesian model, the variational EM algorithm discussed in Section 13.3 is applied with the goal of obtaining estimates, $q(\theta_i), i = 1, 2, \dots, l$, of the posteriors of the abundance parameters given the observations. In the experiments, the respective mean values of $q(\theta_i)$ will be used as estimates of the unknown parameter values. Details on the derivation can be obtained from [72]. The alternative path to the variational EM algorithm is to employ Monte Carlo techniques (see, e.g., [12]).

13.14.2 EXPERIMENTAL RESULTS

The previously described model was applied to a real hyperspectral image, collected by the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) over a Cuprite mining district in Nevada in the

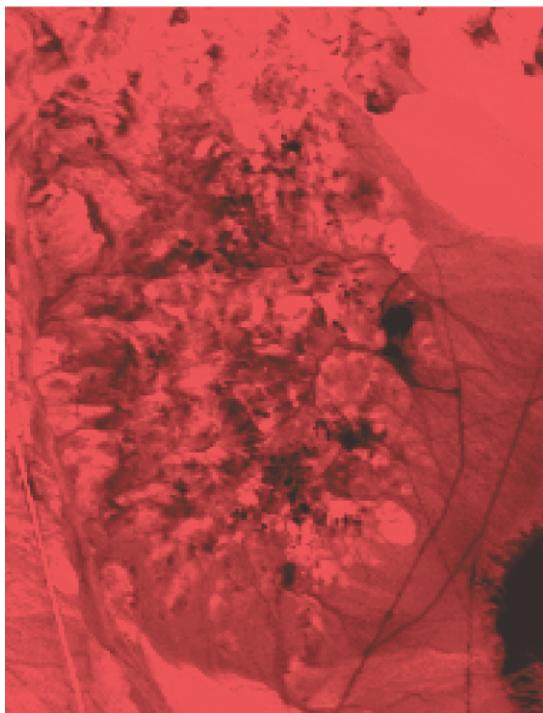


FIGURE 13.19

Composite of the AVIRIS Cuprite subimage using bands 183, 193, and 203 (from [61]). The full RGB color image is available from the site of this book.

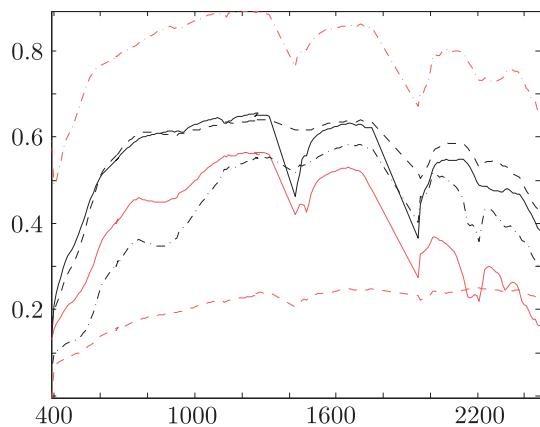
summer of 1997.¹³ The Cuprite data set has been extensively used to evaluate remote sensing technologies and spectral unmixing algorithms (e.g., [26, 45, 71]). It comprises 224 spectral bands in the range from 400 to 2500 nanometers. A subimage of the Cuprite data set with size 250×191 pixels is used in our experiments. Figure 13.19 displays a composite of our image, where bands 183, 193, and 203 have been used, respectively.

After removing some low signal-to-noise ratio (SNR) bands and water-vapor absorption bands, $M = 188$ spectral bands remain available for processing. As a preprocessing step, the VCA algorithm¹⁴ has been used to extract 14 endmembers from our hyperspectral image, as in [45]. The vertex component analysis (VCA) algorithm identifies the signatures of the “pure” pixels in the image and considers them pure material signatures. A plot of the spectral signatures of the extracted endmembers versus the wavelength is displayed in Figure 13.20.

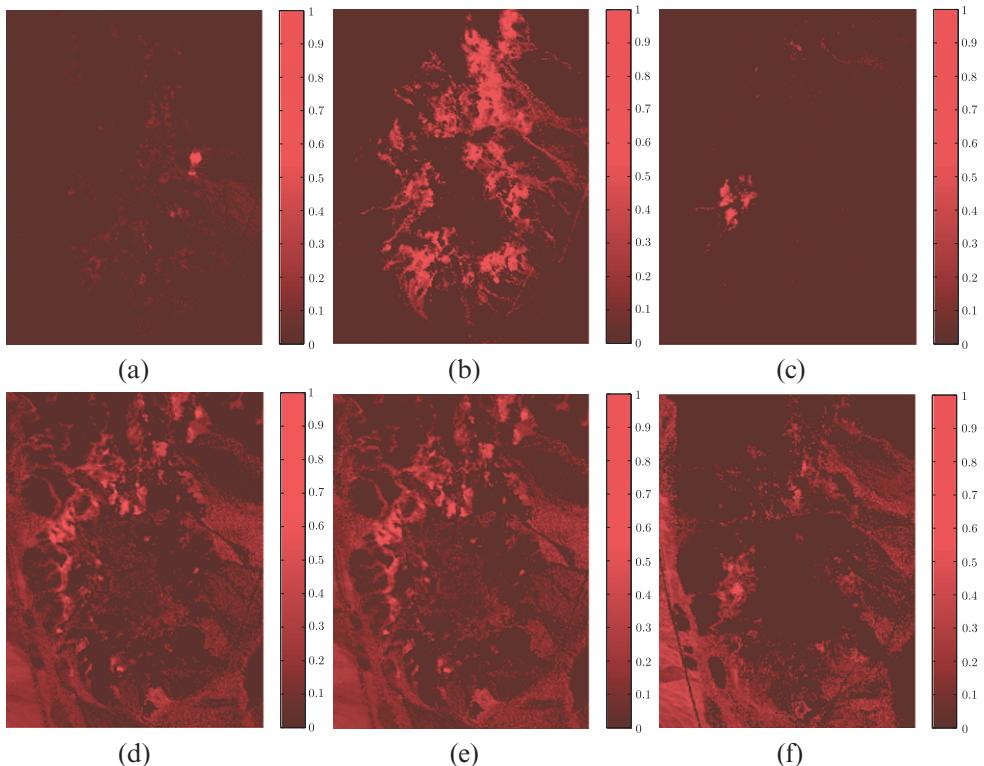
Figure 13.21 shows the resulting abundance maps for six different endmembers, using the variational Bayes method. A dark (resp. light) pixel reveals a low (resp. high) proportional percentage for the

¹³ The data are publicly available at http://aviris.jpl.nasa.gov/data/free_data.html.

¹⁴ The VCA code is available at <http://www.lx.it.pt/~bioucas/code.htm>.

**FIGURE 13.20**

Spectral signatures of 6 out of the 14 endmembers extracted from the Cuprite image using the VCA algorithm [45]. A figure showing all 14 signatures can be downloaded from the site of this book.

**FIGURE 13.21**

Estimated abundance maps for the materials (a) Muscovite, (b) Alunite, (c) Buddingtonite, (d) Montmorillonite, (e) Kaolinite 1, and (f) Kaolinite 2. The full-color image is available from the site of this book.

respective endmember in that pixel. In other words, each image shows the distribution of values of a specific abundance coefficient, θ_i , over the sensed earth surface.

More important, we are able to identify the presented endmembers in Figure 13.21 as muscovite, alunite, buddingtonite, montmorillonite, kaolinite 1, and kaolinite 2.

PROBLEMS

- 13.1** Show Eq. (13.5).
- 13.2** Show Eq. (13.37).
- 13.3** Show Eqs. (13.42)–(13.44).
- 13.4** Show that if

$$p(x) \propto \frac{1}{x},$$

then the random variable $z := \ln x$ follows a uniform distribution.

- 13.5** Derive the lower bound after convergence of the variational Bayesian EM for the linear regression task, which is modeled in Section 13.3.
- 13.6** Consider the Gaussian mixture model

$$p(\mathbf{x}) = \sum_{k=1}^K P_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, Q_k^{-1}),$$

with priors

$$p(\boldsymbol{\mu}_k) = \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{0}, \beta^{-1} I), \quad (13.148)$$

and

$$p(Q_k) = \mathcal{W}(Q_k | v_0, W_0).$$

Given the set of observations $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}, \mathbf{x} \in \mathbb{R}^l$, derive the respective variational Bayesian EM algorithm, using the mean field approximation for the involved posterior pdfs. Consider $P_k, k = 1, 2, \dots, K$, as deterministic parameters and optimize the respective lower bound of the evidence with respect to the P_k 's.

- 13.7** Consider the Gaussian mixture model of Problem 13.6, with the following priors imposed on $\boldsymbol{\mu}$, Q , and \mathbf{P} :

$$\begin{aligned} p(\boldsymbol{\mu}, Q) &= p(\boldsymbol{\mu} | Q)p(Q) \\ &= \prod_{k=1}^K \mathcal{N}(\boldsymbol{\mu}_k | \mathbf{0}, (\lambda Q_k)^{-1}) \mathcal{W}(Q_k | v_0, W_0), \end{aligned}$$

that is, a Gaussian-Wishart product and

$$p(\mathbf{P}) = \text{Dir}(\mathbf{P} | a) \propto \prod_{k=1}^K P_k^{a-1},$$

that is, a Dirichlet prior. That is, \mathbf{P} is treated as a random vector. Derive the E algorithmic steps of the variational Bayesian approximation, adopting the mean field approximation for

the involved posterior pdfs. We have adopted the notation $\boldsymbol{\mu}$ in place of $\boldsymbol{\mu}_{1:K}$ and \boldsymbol{Q} in place of $\boldsymbol{Q}_{1:K}$, for notational simplicity.

- 13.8** If $\boldsymbol{\mu}$ and \boldsymbol{Q} are distributed according to a Gaussian-Wishart product,

$$p(\boldsymbol{\mu}, \boldsymbol{Q}) = \mathcal{N}(\boldsymbol{\mu} | \hat{\boldsymbol{\mu}}, (\lambda \boldsymbol{Q})^{-1}) \mathcal{W}(\boldsymbol{Q} | \nu, \boldsymbol{W}).$$

Compute the expectation

$$\mathbb{E}[\boldsymbol{\mu}^T \boldsymbol{Q} \boldsymbol{\mu}].$$

- 13.9** Derive the Hessian matrix with respect to $\boldsymbol{\theta}$ of the cost function

$$\begin{aligned} J(\boldsymbol{\theta}) &= \sum_{n=1}^N \left[y_n \ln \sigma(\boldsymbol{\phi}^T(x_n) \boldsymbol{\theta}) + (1 - y_n) \ln (1 - \sigma(\boldsymbol{\phi}^T(x_n) \boldsymbol{\theta})) \right] \\ &\quad - \frac{1}{2} \boldsymbol{\theta}^T \boldsymbol{A} \boldsymbol{\theta}, \end{aligned}$$

where

$$\sigma(z) = \frac{1}{1 + \exp(-z)}.$$

- 13.10** Show that the marginal of a Gaussian pdf with a gamma prior on the variance, after integrating out the variance, is the student's-t pdf, given by

$$\text{st}(x|\boldsymbol{\mu}, \lambda, \nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})} \left(\frac{\lambda}{\pi \nu} \right)^{1/2} \frac{1}{\left(1 + \frac{\lambda(x-\mu)^2}{\nu} \right)^{\frac{\nu+1}{2}}}. \quad (13.149)$$

- 13.11** Derive the pair of recursions Eqs. (13.61)–(13.62).

- 13.12** Consider a two-class classification task and assume that the feature vectors in each one of the two classes, ω_1 , ω_2 , are distributed according to the Gaussian pdf. Both classes share the same covariance matrix $\boldsymbol{\Sigma}$, and the mean values are $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$, respectively. Prove that, given an observed feature vector, $\mathbf{x} \in \mathbb{R}^l$, the posterior probabilities for deciding in favor of one of the classes is given by the logistic function, that is,

$$P(\omega_2|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x} + \theta_0)},$$

where

$$\boldsymbol{\theta} := \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1),$$

and

$$\theta_0 = \frac{1}{2}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_2 + \boldsymbol{\mu}_1) + \ln \frac{P(\omega_1)}{P(\omega_2)}$$

- 13.13** Derive Eq. (13.73).

- 13.14** Show Eq. (13.74).

- 13.15** Derive the recursion Eq. (13.76).

- 13.16** Show that if f is a convex function, $f : \mathbb{R}^l \rightarrow \mathbb{R}$, then it is equal to the conjugate of its conjugate, i.e., $(f^*)^* = f$.

13.17 Prove that

$$f(x) = \ln \frac{\lambda}{2} - \lambda \sqrt{x}, \quad x \geq 0$$

is a convex function.

13.18 Derive variational bounds for the logistic regression function

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

one of them in terms of a Gaussian function. For the latter case, use the transformation,

$$t = \sqrt{x}.$$

13.19 Prove Eq. (13.99).

13.20 Derive Eqs. (13.110) and (13.111).

13.21 Derive Eq. (13.112) from Eq. (13.109).

MATLAB Exercises

- 13.22** Generate $N = 60$ data points from each of the five Gaussian distributions given in [Example 13.1](#). Implement the EM algorithm to obtain estimates of the parameters of the Gaussian mixture model ([Exercise 12.17](#)). Run the EM algorithm on our generated data, assuming $K = 25$ clusters, using randomly chosen values for the initial mean values and the covariance matrices. Next, implement the variational Bayes algorithm that treats the same problem, according to the steps reported in [Section 13.4](#). Plot the initial and final estimates of the EM and the variational Bayes algorithm to reproduce the results of [Figure 13.4](#). Play with different values of the parameters.
- 13.23** Generate a vector comprising $N = 100$ equidistant sampling points x_n in the interval $[-10, 10]$. Compute N basis functions, each one located at a sampling point x_n , of the form $\phi_n(x) = \exp(-(x - x_n)^2/2\sigma_\phi^2)$, where $\sigma_\phi^2 = 0.1$. Select two of the basis functions randomly to compute the output samples, y_n , according to the regression model of [Example 13.2](#). The additive noise power should correspond to an SNR level of 6dB. Implement the EM algorithm expressed in Eqs. (12.70), (12.71), (12.78) and (12.79), in order to fit a (generalized) linear regression model comprising the N basis functions to the generated data y_n . Also, implement the variational Bayes EM, summarized in [Algorithm 13.1](#). Plot the reconstructed signals and compare the results.
- 13.24** Generate $N = 150$ two-dimensional data points \mathbf{x}_n , uniformly distributed in the region $[-5, 5] \times [-5, 5]$. Assign a binary label to each \mathbf{x}_n , according to the graph of the function

$$f(x) = 0.5x^3 + 0.5x^2 + 0.5x + 1,$$

in the two-dimensional space. To generate the training data, each $\mathbf{x}_n = [x_{n1}, x_{n2}]^T$ is assigned to one of two classes ω_1, ω_2 , depending on which side of the above graph the following quantity,

$$y_n = 0.5x_{n1}^3 + 0.5x_{n1}^2 + 0.5x_{n1} + 1 + \eta,$$

lies, where η stands for zero-mean Gaussian noise of variance $\sigma_\eta^2 = 4$. In other words, the class assignment is done according to whether $y_n > f(x_{n1})$ or $y_n < f(x_{n1})$. Download and run

the MATLAB code of the RVM classifier¹⁵ for the generated data set. Use the Gaussian kernel with $\sigma^2 = 3$. Repeat the experiments with different values of σ^2 . Plot and discuss the classification results.

- 13.25** Consider an one-dimensional Gaussian process with zero mean and Gaussian (kernel) covariance function, with length scale $h = 0.5$
- (a) Sample $D = 100$ equidistant input points in the interval $[-2, 2]$. Use these as input points to compute the covariance function of the GP and form the respective 100×100 covariance matrix. Use the corresponding multivariate Gaussian to generate samples for five different realizations and plot the results, as in [Figure 13.16](#). Repeat the same experiment with different values for the parameter h .
 - (b) Now, sample $N = 20$ input points from a zero-mean, unit-variance normal distribution. Based on these input points, evaluate the covariance function and the respective 20×20 covariance matrix, as before. Then, generate noisy GP data, by first sampling N points from our Gaussian process, and then adding zero-mean Gaussian noise with variance 0.1. Next, sample $D = 100$ points in the interval $[-3, 4]$. Compute the corresponding mean and the variance of the predictive GP, as given in Eq. (13.138). In a single figure, plot the observed data, the posterior mean, and the error bars of the predictive mean as in [Figure 13.17](#).
- 13.26** Download the Matlab code for the Chinese restaurant process mixture model from <http://sites.google.com/site/kenichikurihara/academic-software>. Generate two-dimensional data from the Gaussian mixture model of [Example 13.5](#) and reproduce the results in [Figure 13.15](#).
- 13.27** Reproduce the hyperspectral unmixing results of [Figure 13.21](#) by running the script “HSIvB.m,” which is available at the website of the book.

REFERENCES

- [1] D. Aldous, Exchangeability and related topics, in: *École d’ Été de Probabilités de Saint-Flour XIII-1983*, Lecture Notes in Mathematics, Springer, New York, 1985, pp. 1-198.
- [2] S. Amari, Differential Geometrical Methods in Statistics, Springer, New York, 1985.
- [3] H. Attias, Inferring parameters and structure of latent variable models by variational Bayes, in: K.B. Laskey, H. Prade (Eds.), *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, Morgan-Kaufmann, San Mateo, 1999, pp. 21-30.
- [4] S. Babacan, R. Molina, A. Katsaggelos, Fast Bayesian compressive sensing using Laplace priors, in: *Proceedings International Conference on Acoustics, Speech and Signal Processing, ICASSP*, Taipei, Taiwan, 2009.
- [5] S.D. Babacan, L. Maniera, R. Molina, A. Katsaggelos, Non-convex priors in Bayesian compressive sensing, in: *Proceedings, 17th European Signal Processing Conference, EURASIP*, Glasgow, Scotland, 2009.
- [6] M.J. Beal, *Variational Algorithms for Approximate Bayesian Inference*, Ph.D. Thesis, University College London, 2003.

¹⁵ The RVM software can be found at <http://www.miketipping.com/sparsebayes.htm>.

- [7] C. Bishop, M. Tipping, Variational relevance vector machines, in: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, 2000, pp. 46-53.
- [8] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, New York, 2006.
- [9] D. Blei, M. Jordan, Variational inference for Dirichlet process mixtures, *Bayesian Anal.* 1 (1) (2006) 121-144.
- [10] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, Cambridge, 2004.
- [11] A. Ben-Israel, T.N.E. Greville, Variational Bayesian model selection for mixture distribution, in: T. Jaakula, T. Richardson (Eds.), Artificial Intelligence and Statistics, Morgan-Kaufmann, San Mateo, 2001, pp. 27-34.
- [12] N. Dobigeon, J.-Y. Tourneret, C.-I. Chang, Semi-supervised linear spectral unmixing using a hierarchical Bayesian model for hyperspectral imagery, *IEEE Trans. Signal Process.* 56 (7) (2008) 2684-2695.
- [13] T. Ferguson, A Bayesian analysis of some nonparametric problems, *Ann. Stat.* 1 (2) (1973) 209-230.
- [14] R.P. Feynman, A Set of Lectures, Perseus, Reading, MA, 1972.
- [15] M.A.P. Figuerido, Adaptive sparseness for supervised learning, *IEEE Trans. Pattern Anal. Mach. Learn.* 25 (9) (2003) 1150-1159.
- [16] J. Gershman, D.M. Blei, A tutorial on Bayesian nonparametric models, *J. Math. Psychol.* 56 (2012) 1-12.
- [17] M.N. Gibbs, D.J.C. MacKay, Variational Gaussian process classifiers, *IEEE Trans. Neural Netw.* 11 (6) (2000) 1458-1464.
- [18] M. Girolami, A variational method for learning sparse and overcomplete representations, *Neural Comput.* 13 (2001) 2517-2532.
- [19] P. Green, S. Richardson, Modeling heterogeneity with and without the Dirichlet process, *Scand. J. Stat.* 28 (2) (2001) 355-375.
- [20] L.K. Hansen, C.E. Rasmussen, Pruning from adaptive regularization, *Neural Comput.* 6 (1993) 1223-1232.
- [21] G.E. Hinton, Van Camp. D., Keeping neural networks simple by minimizing the description length of weight, in: Proceedings 6th ACM Conference on Computing Learning, Santa Cruz, 1993.
- [22] G.E. Hinton, D.S. Zemel, Autoencoders, minimum description length and Helmholtz free energy, in: J.D. Conan, G. Tesauro, J. Alspector (Eds.), Advances in Neural Information Processing System, vol. 6, Morgan-Kaufmann, San Mateo, 1999.
- [23] N. Hjort, C. Holmes, P. Muller, S. Walker, Bayesian Nonparametrics, Cambridge University Press, Cambridge, 2010.
- [24] M.D. Hoffman, M.D. Blei, C. Wang, J. Paisley, Stochastic variational inference, *J. Mach. Learn. Res.* 14 (2013) 1303-1347.
- [25] H. Ishwaran, J.S. Rao, Spike and slab variable selection: Frequentist and Bayesian strategies, *Ann. Stat.* 33 (2) (2005) 730-773.
- [26] M.D. Iordache, J.M. Bioucas-Dias, A. Plaza, Collaborative sparse regression for hyperspectral unmixing, *IEEE Trans. Geosci. Remote Sens.* 52 (1)(2014) 341-354.
- [27] T.J. Jaakola, Variational methods for inference and estimation in graphical models, Ph.D. Thesis, Department of Brain and Cognitive Sciences, MIT, Cambridge, USA, 1997.
- [28] T.J. Jaakola, M.I. Jordan, Improving the mean field approximation via the use of mixture distributions, in: M.I. Jordan (Ed.), Learning in Graphical Models, Kluwer, Dordrecht, 1998, pp. 163-173.
- [29] T.J. Jaakola, M.I. Jordan, Bayesian logistic regression: a variational approach, *Stat. Comput.* 10 (2000) 25-37.
- [30] S. Ji, Y. Xue, L. Carin, Bayesian compressive sensing, *IEEE Trans. Signal Process.* 56 (6) (2008) 2346-2356.
- [31] M.I. Jordan, Z. Ghahramaniz, T.J. Jaakola, L.K. Saul, An introduction to variational methods in graphical models, *Mach. Learn.* 37 (1999) 183-233.
- [32] N. Keshava, A survey of spectral unmixing algorithms, *Lincoln Lab. J.* 14 (1) (2003) 55-78.
- [33] K. Kurihara, M. Welling, Y. Teh, Collapsed variational Dirichlet process mixture models, in: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 20, 2007, pp. 2796-2801.

- [34] M. Kuss, C. Rasmussen, Assessing approximations for Gaussian classification, in: *Advances in Neural Information Processing Systems*, vol. 18, MIT Press, Cambridge, MA, 2006.
- [35] M. Lazaro-Gredilla, A. Figueiras-Vidal, Inter-domain Gaussian processes for sparse inference using inducing features, in: *Advances in Neural Information Processing Systems*, vol. 22, MIT Press, Cambridge, MA, 2010.
- [36] M. Lazaro-Gredilla, Bayesian warped Gaussian processes, in: *Advances in Neural Information Processing Systems*, vol. 25, MIT Press, Cambridge, MA, 2013.
- [37] F.B. Lemper, *Posterior Probabilities of Alternative Linear Models*, Rotterdam University Press, Rotterdam, 1971.
- [38] D.J.C. MacKay, Bayesian interpolation. *Neural Comput.* 4 (3) (1992) 417-447.
- [39] D.J.C. MacKay, The evidence framework applied to classification networks, *Neural Comput.* 4 (1992) 720-736.
- [40] D.J.C. MacKay, *Information Theory, Inference and Learning Algorithms*, Cambridge University Press, Cambridge, 2003.
- [41] D.J.C. MacKay, Bayesian nonlinear modeling for the energy prediction competition, *ASHRAE Trans.* 100 (2) (1994) 1053-1062.
- [42] T. Minka, Expectation propagation for approximate Bayesian inference, in: J. Breese, D. Koller (Eds.), *Proceedings 17th Conference on Uncertainty in Artificial Intelligence*, 2001, pp. 362-369.
- [43] T. Minka, Divergence measures and message passing, Technical Report, Microsoft Research Laboratory, Cambridge, UK, 2005.
- [44] T. Mitchell, J. Beauchamp, Bayesian variable selection in linear regression, *J. Am. Stat. Assoc.* 83 (1988) 1023-1036.
- [45] J.M.P. Nascimento, J.M. Bioucas-Dias, Vertex component analysis: a fast algorithm to unmix hyperspectral data, *IEEE Trans. Geosci. Remote Sens.* 43 (4) (2005) 898-910.
- [46] R.M. Neal, Bayesian learning for neural networks, in: *Lecture Notes in Statistics*, vol. 118, Springer-Verlag, New York, 1996.
- [47] R.M. Neal, Monte Carlo implementation for Gaussian process models for Bayesian regression and classification, Technical Report CRG-TR-97-2, Department of Computer Science, University of Toronto, 1997.
- [48] R.M. Neal, Assessing relevance determination methods using DELVE, in: C. Bishop (Ed.), *Neural Networks and Machine Learning*, Springer-Verlag, New York, 1998, pp. 97-120.
- [49] R. Neal, Markov chain sampling methods for Dirichlet process mixture models, *J. Comput. Graph. Stat.* 9 (2) (2000) 249-265.
- [50] A. O'Hagan, J.F. Kingman, Curve fitting and optimal design for prediction, *J. R. Stat. Soc. B* 40 (1) (1978) 1783-1816.
- [51] M. Opper, O. Winther, A Bayesian approach to on-line learning, in: D. Saad (Ed.), *On-line Learning in Neural Networks*, Cambridge University Press, Cambridge, 1999, pp. 363-378.
- [52] J. Palmer, D. Wipf, K. Kentz-Delgade, B. Rao, Variational EM algorithms for non-Gaussian latent variable models, in: *Advances in Neural Information Systems*, vol. 18, 2006, pp. 1059-1066.
- [53] G. Parisi, *Statistical Field Theory*, Addison Wesley, New York, 1988.
- [54] F. Perez-Cruz, S. Van Vaerenbergh, J.J. Murillo-Fuentes, M. Lazaro-Gredilla, I. Santamaria, Gaussian processes for nonlinear signal processing, *IEEE Signal Process. Mag.* 30 (4) (2013) 40-50.
- [55] J. Pitman, Combinatorial stochastic processes, Technical report 621, Notes for Saint Flour Summer School, Department of Statistics, UC, Berkeley, 2002.
- [56] P. Pal, P.P. Vaidyanathan, Parameter identifiability in sparse Bayesian learning, in: *Proceedings International Conference on Acoustics, Speech and Signal Processing, ICASSP*, Florence, Italy, 2014.

- [57] J. Quiñonero-Candela, C.E. Rasmussen, A unifying view of sparse approximate Gaussian process regression, *Mach. Learn. Res.* 6 (2005) 1939-1959.
- [58] C.E. Rasmussen, C.K.I. Williams, *Gaussian Processes for Machine Learning*, MIT Press, Cambridge, MA, 2006.
- [59] R. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, NJ, 1970.
- [60] G.A. Rodriguez-Yam, R.A. Davis, L.L. Scharf, A Bayesian model and Gibbs sampler for hyperspectral imaging, in: Proceedings, IEEE Sensor Array and Multichannel Signal Processing Workshop, 2002, pp. 105-109.
- [61] S. Ryan, M. Lewis, Mapping soils using high resolution airborne imagery, Barossa Valley, SA, in: Proceedings of the Inaugural Australian Geospatial Information and Agriculture Conference Incorporating Precision Agriculture in Australasia 5th Annual Symposium, 2001, pp. 17-19.
- [62] S. Sarkka, A. Solin, J. Hartikainen, Spatiotemporal learning via infinite-dimensional Bayesian filtering and smoothing, *IEEE Signal Process. Mag.* 30 (4) (2013) 51-61.
- [63] M.A. Sato, Online model selection based on the variational Bayes, *Neural Comput.* 13 (7) (2001) 1649-1681.
- [64] M.N. Schmidt, M. Morup, Nonparametric Bayesian modeling of complex networks, *IEEE Signal Process. Mag.* 30 (3) (2013) 110-128.
- [65] M.W. Seeger, H. Nickish, Large scale variational inference and experimental design for sparse generalized linear models, Technical report, # TR-175, Max Plank Institute für Biologische Kybernetic, 2008.
- [66] M.W. Seeger, D.P. Wipf, Variational Bayesian inference techniques, *IEEE Signal Process. Mag.* 27 (1) (2010) 81-91.
- [67] J. Sethuraman, A constructive definition of Dirichlet priors, *Stat. Sin.* 4 (2) (1994) 639-650.
- [68] E. Snelson, C.E. Rasmussen, Z. Ghahramani, Warped Gaussian processes, in: *Advances in Neural Information Processing Systems*, vol. 16, MIT Press, Cambridge, MA, 2003.
- [69] E. Snelson, Z. Ghahramani, Sparse Gaussian processes using pseudo-inputs, in: *Advances in Neural Information Processing Systems*, vol. 18, MIT Press, Cambridge, MA, 2006, pp. 1259-1266.
- [70] C. Soussen, J. Idier, D. Brie, J. Duan, From Bernoulli-Gaussian deconvolution to sparse signal restoration, *IEEE Trans. Signal Process.* 59 (10) (2011) 4572-4584.
- [71] K.E. Themelis, A.A. Rontogiannis, K.D. Kourtoumbas, A novel hierarchical Bayesian approach for sparse semisupervised hyperspectral unmixing, *IEEE Trans. Signal Process.* 60 (2) (2012) 585-599.
- [72] K.E. Themelis, A.A. Rontogiannis, K.D. Kourtoumbas, Semisupervised hyperspectral image unmixing using a variational Bayes algorithm, 2014, <http://arxiv.org/abs/1406.4705>.
- [73] K. Themelis, A. Rontogiannis, K. Kourtoumbas, A variational Bayes framework for sparse adaptive estimation, *IEEE Trans. Signal Process.* (2014), <http://dx.doi.org/10.1109/TSP.2014.2338839>, to appear 2015.
- [74] S. Theodoridis, K. Kourtoumbas, *Pattern Recognition*, fourth ed., Academic Press, Boston, 2009.
- [75] M.E. Tipping, Sparse Bayesian learning and the relevance vector machine, *J. Mach. Learn. Res.* 1 (2001) 211-244.
- [76] M.E. Tipping, A.C. Faul, Fast marginal likelihood maximisation for sparse Bayesian models, in: C.M. Bishop, B.J. Frey (Eds.), *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, Key West, FL, 2003.
- [77] M.K. Titsias, Variational learning of inducing variables in sparse Gaussian processes, in: *Proceedings 12th International Workshop on Artificial Intelligence and Statistics*, 2009, pp. 567-574.
- [78] S. Van Vaerenbergh, M. Lazaro-Gredilla, I. Santamaría, Kernel recursive least-squares tracker for time-varying regression, *IEEE Trans. Neural Netw. Learn. Syst.* 23 (8) (2012) 1313-1326.
- [79] C.K.I. Williams, D. Barber, Bayesian classification with Gaussian processes, *IEEE Trans. Pattern Anal. Mach. Intell.* 20 (1998) 1342-1351.

- [80] D. Wipf, Bayesian methods for finding sparse representations, Ph.D. Thesis, University of California, San Diego, 2006.
- [81] D.P. Wipf, B.D. Rao, An empirical Bayesian strategy for solving the simultaneous sparse approximation problem, *IEEE Trans. Signal Process.* 55 (7) (2007) 3704-3716.
- [82] D. Wipf, S. Nagarajan, A new view of automatic relevance determination, in: *Advances in Neural Information Systems (NIPS)*, vol. 20, 2008.
- [83] D. Wipf, B. Rao, S. Nagarajan, Latent variable models for promoting sparsity, *IEEE Trans. Informat. Theory* 57(9) (2011) 6236-6255.
- [84] D.P. Wipf, B.D. Rao, S. Nagarajan, Latent variable Bayesian methods for promoting sparsity, *IEEE Trans. Inf. Theory* 57 (9) (2011) 6236-6255.
- [85] X. Zhang, B.D. Rao, Sparse signal recovery with temporally correlated source vectors using sparse Bayesian learning, *IEEE Trans. Select. Areas Signal Process.* 5 (5) (2011) 912-926.
- [86] X. Zhang, B.D. Rao, Extension of SBL Algorithms for the recovery of block sparse signals with intra-block correlation, *IEEE Trans. Signal Process.* 61 (8) (2013) 2019-2015.

MONTE CARLO METHODS

14

CHAPTER OUTLINE

14.1	Introduction	717
14.2	Monte Carlo Methods: The Main Concept.....	718
14.2.1	Random Number Generation	719
14.3	Random Sampling Based on Function Transformation	721
14.4	Rejection Sampling	725
14.5	Importance Sampling	728
14.6	Monte Carlo Methods and the EM Algorithm	730
14.7	Markov Chain Monte Carlo Methods	731
14.7.1	Ergodic Markov Chains	733
14.8	The Metropolis Method	738
14.8.1	Convergence Issues	741
14.9	Gibbs Sampling	743
14.10	In Search of More Efficient Methods: A Discussion	745
	Variational Inference or Monte Carlo Methods	746
14.11	A Case Study: Change-Point Detection	747
Problems.....		750
	MATLAB Exercise	752
References.....		752

14.1 INTRODUCTION

In [Chapters 12](#) and [13](#), the Bayesian inference task was considered. A large part of the latter chapter was dedicated to dealing with approximation techniques, which offered escape routes when the involved pdfs were complex enough to render integral computations intractable. All these techniques were of a deterministic nature; that is, the goal was to approximate the mathematical expression of the corresponding pdf by another one that could ease the associated calculations. Such methods include the Laplacian approximation, as well as the variational methods based on the mean field theory or the convex duality concept. Deterministic approximation methods will also be used for approximate inference in [Chapter 15](#), to deal with graphical models.

In this chapter, we turn our attention to approximation methods with a much stronger statistical flavor, which are based on randomly generating samples using numerical techniques; these samples

are typical of an underlying distribution, which may be of either continuous or discrete nature. This is an old field, with origins tracing back to the late forties and early fifties in the pioneering work of Stanislaw Ulam, John Von-Neumann, and Nicholas Metropolis in Los Alamos, when the term *Monte Carlo* was coined as an umbrella name of such techniques, inspired by the famous casino in Monaco (see, e.g., [28] for a historical note). The first application of such techniques, which coincided with the development of the first computers, was in the context of the Manhattan project for developing the hydrogen bomb; soon after, Monte Carlo methods were embraced by almost every scientific area where statistical computations are involved.

As is often the case with pioneering ideas, when they are looked at a posteriori, that is, once they have been stated, the basic idea seems simple. Our current task of interest is the computation of an integral, which involves a pdf; this can alternatively be interpreted as the computation of an “expectation.” Such a view provides the permit to approximate the integral as the sample mean of the involved quantities, given a sufficient number of samples and exploiting the law of large numbers.

To condense a field with a history of a number of decades in a single chapter is obviously impossible. Our goal is to present the basic concepts, definitions, and directions, with the aim of serving the needs associated with typical machine learning tasks rather than looking at it as an entity on its own.

We start with the more classical methods using transformations and then move on to the rejection and importance sampling techniques. In the sequel, the more powerful methods based on arguments from the theory of Markov chains are reviewed. The Metropolis-Hastings and the Gibbs sampling methods are presented and discussed. Finally, a case study concerning the change-point detection task is considered.

14.2 MONTE CARLO METHODS: THE MAIN CONCEPT

Our starting point is the evaluation of integrals of the form

$$\mathbb{E}[f(\mathbf{x})] := \int_{-\infty}^{\infty} f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}, \quad (14.1)$$

where $\mathbf{x} \in \mathbb{R}^l$ is a random vector and $p(\mathbf{x})$ is the corresponding distribution.¹ Our interest lies in cases where the forms of $f(\mathbf{x})$ and/or $p(\mathbf{x})$ are such that the evaluation of such integrals is intractable. For example, such integrations occur in the evaluation of the evidence in Eq. (12.14), in the prediction task (Eq. (12.18)), and in the E-step of the EM algorithm (Eq. (12.61)). In Eq. (12.14), the random variable is the parameter vector $\boldsymbol{\theta}$ and $f(\boldsymbol{\theta}) = p(\mathbf{y}|\boldsymbol{\theta})$.

Coming back to Eq. (14.1), assume that one has at her/his disposal a number of i.i.d. samples, $\mathbf{x}_1, \dots, \mathbf{x}_N$, drawn from $p(\mathbf{x})$. Then, the following approximation

$$\mathbb{E}[f(\mathbf{x})] \simeq \frac{1}{N} \sum_{i=1}^N f(\mathbf{x}_i) := \bar{\mathbb{E}}_{f,N}, \quad (14.2)$$

¹ In the case of discrete variables, $p(\mathbf{x})$ becomes the probability mass function, $P(\mathbf{x})$, and integrations are replaced by summations.

is justified by (a) the *law of large numbers* and (b) the *central limit theorem* [32]. Let us denote as $\mathbb{E}[f(\mathbf{x})] = \mu_f$ and the respective variance as $\text{var}[f(\mathbf{x})] := \mathbb{E}[(f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})])^2] = \sigma_f^2$. Then, the previously referred two theorems guarantee that

$$\lim_{N \rightarrow \infty} \bar{\mathbb{E}}_{f,N} = \mu_f, \quad (14.3)$$

and

$$p(\bar{\mathbb{E}}_{f,N}) \simeq \mathcal{N}\left(\bar{\mathbb{E}}_{f,N} \mid \mu_f, \frac{\sigma_f^2}{N}\right). \quad (14.4)$$

The limit in Eq. (14.3) refers to the notion of *almost sure convergence*, that is,

$$\text{Prob} \left\{ \lim_{N \rightarrow \infty} |\mu_f - \bar{\mathbb{E}}_{f,N}| = 0 \right\} = 1.$$

The approximate Gaussian distribution in Eq. (14.4) guarantees that the variance (as one changes the set of N samples) of the obtained estimate, $\bar{\mathbb{E}}_{f,N}$, around the true value, μ_f , decreases with N .

Thus, if one generates the samples \mathbf{x}_n , $n = 1, 2, \dots, N$, from the distribution $p(\mathbf{x})$, the use of Monte Carlo techniques offers the means for an approximation of the integral in Eq. (14.1) with the following nice properties: (a) the approximation error is decreasing as $\frac{1}{\sqrt{N}}$; (b) the obtained estimate using N samples is an unbiased estimate of the true value; and (c) the convergence rate is *independent* on the dimensionality, l . The latter property is in contrast to methods based on the deterministic numerical integration, which, in general, have a rate of convergence that slows down as the dimensionality increases. In Monte Carlo techniques, if one is not satisfied with the obtained accuracy, all he/she has to do is generate more samples.

The crucial point now becomes that of developing techniques to generate i.i.d. samples from $p(\mathbf{x})$. This is not an easy task, especially for high-dimensional spaces. Note that achieving a certain accuracy for the estimator in Eq. (14.2) is independent on the dimensionality, once i.i.d. samples drawn from $p(\mathbf{x})$ are available. On the other hand, drawing i.i.d. samples typical of $p(\mathbf{x})$ becomes harder as the dimensionality increases. We will return to this point soon. In the sequel, we will focus on some basic directions on how to achieve the aforementioned goal.

14.2.1 RANDOM NUMBER GENERATION

Random number generation can be achieved either as the result of an experiment or via the use of computers. For example, the tosses of a fair coin can generate a random sequence of 0's (heads) or 1's (tails). Another example is the sequence of numbers corresponding to the distance between radioactive emissions; such an experiment generates a sequence of exponentially distributed samples. However, such approaches are not of much practical value and the emphasis has been on techniques that generate samples via a computer, using a *pseudorandom number generator*. At the heart of such methods lie algorithms that guarantee the generation of a sequence of *integers*, z_i , which approximately follow a *uniform distribution* in an interval in the real axis. In the sequel, the generation of random numbers/vectors, which follow an arbitrary distribution, is obtained *indirectly* via a variety of methods, each with its pros and cons. The path for generating integers in an interval, $(0, M)$, follows the general recursion,

$$z_i = g(z_{i-1}, \dots, z_{i-m}) \bmod M,$$

where g is a function depending on the m previously generated samples and mod denotes the modulus operation; that is, z_i is the remainder of the division of $g(z_{i-1}, \dots, z_{i-m})$ by M . The simpler form is the linear version,

$$z_i = \alpha z_{i-1} \bmod M, \quad z_0 = 1, \quad i \geq 1, \quad (14.5)$$

where M is a large prime number and α is an integer. Recursion (14.5) generates a sequence of numbers between 1 and $M - 1$. The method is known as *linear congruential generator* or Lehmer's algorithm [20].

If α is properly chosen, then the resulting sequence of numbers turns out to be periodic with period $M - 1$. This is the reason we call these generators pseudorandom, because a periodic sequence can never be claimed to be random. However, for large values of M , the obtained sequence can be sufficiently random with uniform distribution, provided, of course, that $N \leq M - 1$. For example, a value of M of the order of 10^9 is sufficient for most applications. Note that not all possible choices of the parameter α guarantee a good generator. In practice, a sequence is accepted as being random only if it meets a number of related tests of randomness and is subsequently used successfully in a variety of applications (see, e.g., [32]). A common choice of parameters that leads to a reasonably good uniformly distributed random sequence is $\alpha = 7^5$ and $M = 2^{31} - 1$ (see, e.g., [34]). More on this topic can be found in Knuth's classical text and the references therein [18]. Once a sequence of integers is available, a sequence of uniformly distributed real random numbers is obtained as the ratio $x_i = \frac{z_i}{M} \in (0, 1)$ (as a matter of fact, this is the sequence on which the randomness tests are applied).

Remarks 14.1.

- Note that even the generation of a sequence of (pseudo) random numbers with uniform distribution in $(0, 1)$, is not an easy task, in spite of the fact that the uniform distribution is an “easy” one; that is, all values are equally probable. Moreover, often in practice, a pdf is known up to its normalizing constant, that is,

$$p(\mathbf{x}) = \frac{\phi(\mathbf{x})}{Z},$$

where

$$Z = \int_{-\infty}^{+\infty} \phi(\mathbf{x}) d\mathbf{x}.$$

However, if $\phi(\mathbf{x})$ has a complicated form, the previous integration may be intractable. This is often met when computing posterior pdfs. The previous points make the process of sampling from a general $p(\mathbf{x})$ much harder than for the case of a uniform one. The task becomes even harder in high dimensions, even if Z is available. The required number of points, in order to cover sufficiently a region in a high-dimensional space, exhibits an exponential dependence on the respective dimensionality (curse of dimensionality). Thus, a huge number of points is needed in order to get a good representation of $p(\mathbf{x})$ in high-dimensional spaces. In practice, one would be more content to generate samples from the regions where $p(\mathbf{x})$ gets relatively high values. However, the higher the dimensionality, the more difficult the task of locating the high-probability regions. Similar arguments hold for random variables of a discrete nature, where the number of states that the variable can take is very large. Ideally, in order to have a representative sequence of samples, all states have to be visited.

14.3 RANDOM SAMPLING BASED ON FUNCTION TRANSFORMATION

In this section, we deal with some of the most basic techniques for drawing samples from a pdf, $p(x)$.

Function Inversion. Let x be a real random variable with a pdf, $p(x)$, and a corresponding cumulative distribution function

$$F_x(x) = \int_{-\infty}^x p(\tau) d\tau.$$

It is known from probability theory that the random variable, u , defined as

$$u := F_x(x), \quad (14.6)$$

is uniformly distributed in the interval $0 \leq u \leq 1$ irrespective of the nature of $p(x)$ [32] ([Problem 14.1](#)). If, in addition, we assume that the function F_x has an inverse, F_x^{-1} , then we can write that

$$x = F_x^{-1}(u). \quad (14.7)$$

Thus, following the reverse arguments, samples from $p(x)$ can be generated by first generating samples from the uniform distribution, $\mathcal{U}(u|0, 1)$, and then applying on them the inverse function, F_x^{-1} ([Problem 14.2](#)).

This method works well provided that F_x has an inverse that can be easily computed. However, only a few pdfs can be “proud” of having inverses that can be expressed in an analytical form.

Example 14.1. Generate samples, x_n , that follow the exponential distribution,

$$p(x) = \lambda \exp(-\lambda x), \quad x \geq 0, \lambda > 0, \quad (14.8)$$

using a pseudorandom generator that generates samples, u_n , from the uniform distribution $\mathcal{U}(u|0, 1)$.

We have that

$$F_x(x) = \int_0^x \lambda \exp(-\lambda \tau) d\tau = 1 - \exp(-\lambda x).$$

By letting

$$u := F_x(x),$$

and solving for x , we get

$$x = -\frac{1}{\lambda} \ln(1 - u) := F_x^{-1}(u).$$

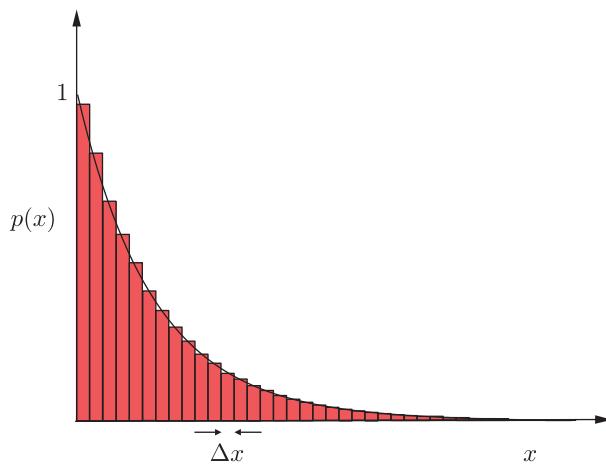
Hence, if u_n are samples drawn from a uniform distribution, the sequence

$$x_n = -\frac{1}{\lambda} \ln(1 - u_n), \quad n = 1, 2, \dots, N,$$

are samples drawn from the exponential pdf in Eq. (14.8). [Figure 14.1](#) shows the histogram of the generated samples for $N = 1000$, alongside $p(x)$ for $\lambda = 1$.

Example 14.2. Generating samples from discrete distributions. Here, an intuitive method for generating samples from discrete distributions is presented. We will use such distributions in [Section 17.2](#).

Let, x_1, x_2, \dots, x_K , denote discrete random events occurring with probabilities, P_1, P_2, \dots, P_K , respectively, such that $\sum_{k=1}^K P_k = 1$. Then, the following simple algorithm draws samples from this distribution.

**FIGURE 14.1**

The histogram of the samples generated from the uniform distribution, and using the inverse of F_x , which describes the exponential pdf, whose curve is shown in black. The length of the bin interval, Δx , was chosen equal to 0.02.

Algorithm 14.1 (Sampling discrete distributions).

- Define $a_k = \sum_{i=1}^{k-1} P_i$, $b_k = \sum_{i=1}^k P_i$, $k = 1, 2, \dots, K$, $a_1 = 0$.
- **For** $i = 1, 2, \dots, K$, **Do**
 - $u \sim \mathcal{U}(0, 1)$
 - Select

$$x_k \text{ if } u \in [a_k, b_k), \quad k = 1, 2, \dots, K$$

- **End For**

[Figure 14.2](#) provides an illustration of the algorithm. Note that the probability jumps at the beginning of each interval and the corresponding cumulative distribution function (CDF) is constructed; the algorithm basically computes the inverse of u , according to this CDF (see, e.g., [4]).

Function Transformation. We will demonstrate the method via an example involving the transformation of two random variables, say r and ϕ , to two new ones, x and y . Let

$$x = g_x(r, \phi),$$

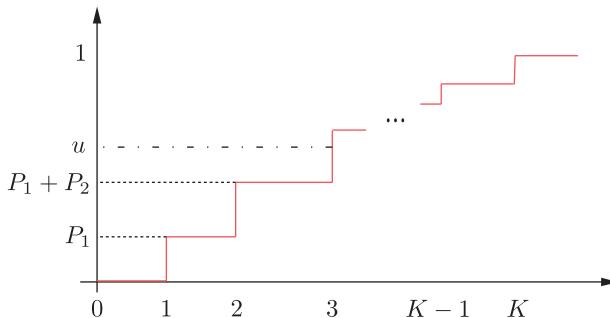
and

$$y = g_y(r, \phi).$$

Let us now assume that there is a unique solution for the inverses and that they can be expressed in an analytic form (which is not the case in general), that is,

$$r = g_r(x, y),$$

$$\phi = g_\phi(x, y).$$

**FIGURE 14.2**

The CDF for a discrete distribution of K discrete random events. If $P_1 + P_2 \leq u < P_1 + P_2 + P_3$, the event x_3 is drawn. Note that the larger the probability of an event, the larger the corresponding interval jump in the CDF is, hence the higher the probability of this event being drawn.

We know from [Section 2.2.5](#) that if $p_{r,\phi}(r, \phi)$ is the joint distribution of r and ϕ , then the joint distribution of x and y is given by

$$\begin{aligned} p_{x,y}(x, y) &= \frac{p_{r,\phi}(g_r(x, y), g_\phi(x, y))}{|\det(J(x, y; r, \phi))|} \\ &= p_{r,\phi}(g_r(x, y), g_\phi(x, y)) |\det(J(r, \phi; x, y))|, \end{aligned} \quad (14.9)$$

where $|\det(J(x, y; r, \phi))|$ is the absolute value of the determinant of the Jacobian matrix,

$$J(x, y; r, \phi) = \begin{bmatrix} \frac{\partial g_x}{\partial r} & \frac{\partial g_x}{\partial \phi} \\ \frac{\partial g_y}{\partial r} & \frac{\partial g_y}{\partial \phi} \end{bmatrix}, \quad (14.10)$$

$J(r, \phi; x, y)$ is analogously defined, and we have assumed, for simplicity, that to each value of (r, ϕ) there corresponds one value of (x, y) . Let us now see how one can generate samples from a Gaussian $p(x) = \mathcal{N}(x|0, 1)$ by using samples drawn from a uniform and an exponential distribution, respectively, for ϕ and r ; recall that in [Example 14.1](#), we described a technique for generating samples from an exponential distribution.

The Box-Müller method. Let r be distributed according to an exponential distribution,

$$p_r(r) = \frac{1}{2} \exp\left(-\frac{r}{2}\right), \quad r \geq 0, \quad (14.11)$$

and ϕ to a uniform distribution, $\mathcal{U}(\phi|0, 1)$,

$$p_\phi(\phi) = \begin{cases} \frac{1}{2\pi} & 0 \leq \phi \leq 2\pi, \\ 0 & \text{otherwise,} \end{cases} \quad (14.12)$$

and also assume that they are independent, that is,

$$p_{r,\phi}(r, \phi) = p_r(r)p_\phi(\phi). \quad (14.13)$$

Generate two new random variables as

$$x = \sqrt{r} \cos \phi, \quad (14.14)$$

$$y = \sqrt{r} \sin \phi. \quad (14.15)$$

The physical interpretation of the previous transformation is that x, y correspond to the cartesian coordinates of a point and r, ϕ are its polar ones, [Figure 14.3](#). From Eqs. (14.14) and (14.15), we can write that

$$r = x^2 + y^2, \quad (14.16)$$

$$\phi = \arctan\left(\frac{y}{x}\right). \quad (14.17)$$

Adjusting Eq. (14.9) to our current needs, using Eqs. (14.11) to (14.13), we obtain

$$\begin{aligned} p_{x,y}(x, y) &= \frac{1}{2\pi} \frac{1}{2} \exp\left(-\frac{x^2 + y^2}{2}\right) 2 \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right), \end{aligned} \quad (14.18)$$

where we have used that ([Problem 14.3](#))

$$|J(x, y; r, \phi)| = \frac{1}{2}.$$

Thus, we have shown that using the transformation given in Eqs. (14.14) and (14.15), we can generate samples from normalized Gaussians.

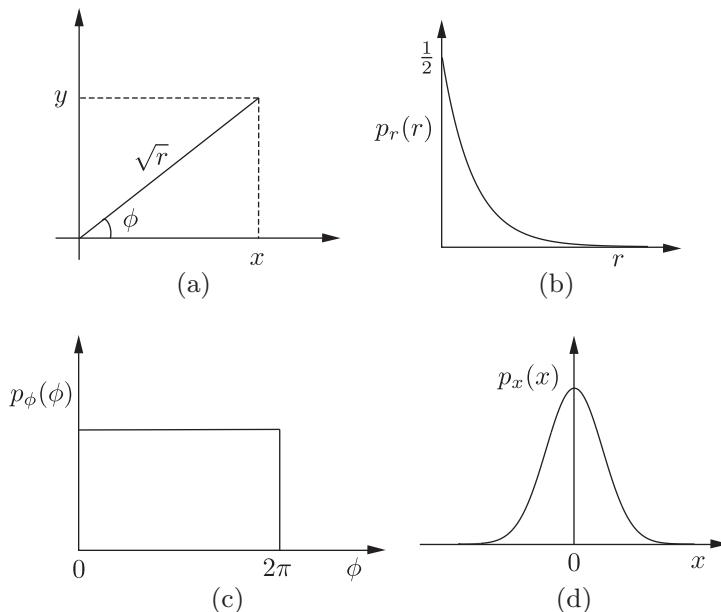
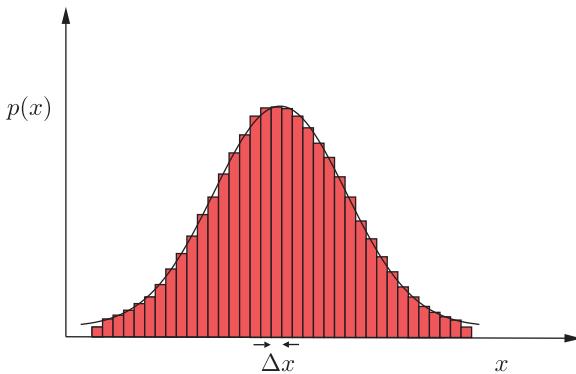


FIGURE 14.3

(a) Relation of the cartesian (x, y) to the polar coordinates (r, ϕ) . (b) and (c) If r and ϕ are random variables following an exponential and a uniform in $[0, 2\pi]$ distributions, respectively, then x and y are independent and they both follow a normalized Gaussian, as shown in (d) for the x variable.

**FIGURE 14.4**

The histogram of $N = 100$ points generated in [Example 14.3](#) together with the graph of $p(x) = \mathcal{N}(x|1, 0.5)$. The bin length was chosen to be equal to $\Delta x = 0.05$.

Once samples from a normalized Gaussian, $\mathcal{N}(x|0, 1)$, are available, samples from a general Gaussian, $\mathcal{N}(y|\mu, \sigma^2)$, are obtained via the obvious transformation,

$$y = \sigma x + \mu. \quad (14.19)$$

The previous approach is also generalized to random vectors in \mathbb{R}^l . One can first draw samples from $\mathbf{x} \sim \mathcal{N}(\mathbf{x}|\mathbf{0}, \mathbf{I})$ by stacking together l i.i.d. samples drawn from a normalized Gaussian, $\mathcal{N}(x|0, 1)$, and then apply the transformation

$$\mathbf{y} = L\mathbf{x} + \boldsymbol{\mu},$$

which is equivalent with drawing samples from

$$\mathbf{y} \sim \mathcal{N}(\mathbf{y}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \quad (14.20)$$

where $\boldsymbol{\Sigma} = LL^T$ (Cholesky factorization, [Problem 14.4](#)).

Example 14.3. Generate $N = 100$ samples, r_n , $n = 1, 2, \dots, 100$, from the exponential distribution in Eq. (14.11) (following [Example 14.1](#)) and $N = 100$ samples, ϕ_n , $n = 1, 2, \dots, 100$, from the uniform in Eq. (14.12). Then use the transformations in Eqs. (14.14), (14.15) and (14.19) to obtain samples, x_n , $n = 1, 2, \dots, 100$, from $p(x) = \mathcal{N}(x|1, 0.5)$. The histogram of the obtained samples is shown in [Figure 14.4](#).

14.4 REJECTION SAMPLING

Applying the previously reported transformation techniques relies on having the involved transform functions available in a convenient (analytic) form, which in general is the exception instead of the rule. From now on, we turn our attention to alternative methods.

Rejection Sampling (e.g., [7, 37]) is conceptually a simple technique; in order to generate independent samples from a desired pdf, $p(x)$, one draws samples from another one, say, $q(x)$, that is easier to

handle, and then, instead of applying a transformation, some of the points are *rejected* according to an appropriate criterion.

Given two random variables, x and u , recall that the marginal, $p(x)$, is obtained by integrating the joint pdf, $p_{x,u}(x, u)$, that is,

$$p(x) = \int_{-\infty}^{+\infty} p_{x,u}(x, u) du. \quad (14.21)$$

Let us now consider the following identity,

$$p(x) \equiv \int_0^{p(x)} 1 dx = \int_{-\infty}^{+\infty} \chi_{[0,p(x)]}(u) du, \quad (14.22)$$

where $\chi_{[0,p(x)]}(\cdot)$ is our familiar characteristic function in the interval $[0, p(x)]$, that is,

$$\chi_{[0,p(x)]}(u) = \begin{cases} 1 & 0 \leq u \leq p(x), \\ 0 & \text{otherwise.} \end{cases}$$

Comparing Eqs. (14.21) and (14.22), it turns out that $\chi_{[0,p(x)]}(u)$ can be interpreted as the joint pdf of the couple (x, u) defined over the set

$$\mathcal{A} = \{(x, u) : x \in \mathbb{R}, 0 \leq u \leq p(x)\}. \quad (14.23)$$

Looking more carefully at $p_{x,u}(x, u) = \chi_{[0,p(x)]}(u)$, it does not take long to realize that this is the *uniform density* under the area of the graph $u = p(x)$, as seen in Figure 14.5a. In other words, if one fills in the shaded area in Figure 14.5a uniformly at random with points (x, u) and then neglects the u dimension, then the obtained points are samples drawn from $p(x)$. We can now go one step further and assume that $p(x)$ is not exactly known, that is,

$$p(x) = \frac{1}{Z} \phi(x),$$

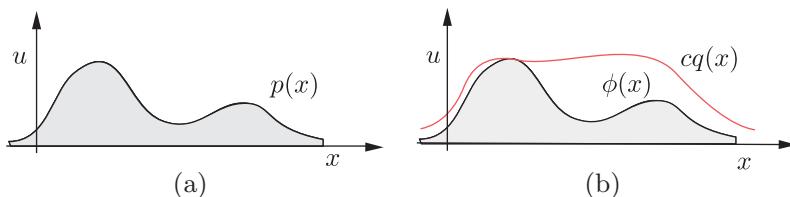


FIGURE 14.5

(a) Filling in the shaded area uniformly at random with points (x_n, u_n) , and after neglecting the coordinate u_n , is equivalent with drawing points, x_n , from $p(x)$. (b) The proposal distribution, $cq(x)$, is everywhere larger or equal to $\phi(x)$.

and that the normalizing constant is not available (as we know, often, the computation of the normalizing constant is not easy). Then, we have that

$$p(x) = \frac{1}{Z} \phi(x) = \frac{1}{Z} \int_0^{\phi(x)} du = \frac{1}{Z} \int_{-\infty}^{+\infty} \chi_{[0, \phi(x)]}(u) du,$$

and Z is given by

$$Z = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \chi_{[0, \phi(x)]}(u) du dx.$$

Hence,

$$p(x) = \frac{\int_{-\infty}^{+\infty} \chi_{[0, \phi(x)]}(u) du}{\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \chi_{[0, \phi(x)]}(u) du dx}. \quad (14.24)$$

In other words, even if $p(x)$ is not exactly known, $p(x)$ can still be obtained, this time in terms of the uniform distribution, $\chi_{[0, \phi(x)]}(x, u)$, normalized appropriately. However, rescaling the uniform does not affect the marginal. It suffices to sample uniformly at random the region \mathcal{A} , which now should be defined in terms of $\phi(x)$ instead of $p(x)$. What we have said so far applies also to random vectors, $\mathbf{x} \in \mathbb{R}^l$, by considering the extended space (\mathbf{x}, u) , and we talk about the volume under the surface $\phi(\mathbf{x})$ (or $p(\mathbf{x})$).

We now turn our attention to see how one can fill in the volume under the surface formed by $u = \phi(\mathbf{x})$ (or $u = p(\mathbf{x})$ if it is fully available), with points uniformly at random. Let $q(\mathbf{x})$ be a distribution from which we know how to draw samples, and we refer to it as the *proposal distribution*. We select a constant c , such that²

$$\phi(\mathbf{x}) \leq cq(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbb{R}^l.$$

The respective geometry is shown in Figure 14.5b. The goal is to draw points in the interval $[0, cq(\mathbf{x})]$ and then keep only those that lie in the region under the surface $u = \phi(\mathbf{x})$. The following algorithm does the job.

Algorithm 14.2 (Rejection sampling).

- **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $x_i \sim q(\mathbf{x})$
 - Draw $u_i \sim \mathcal{U}(0, cq(x_i))$
 - Retain the sample if

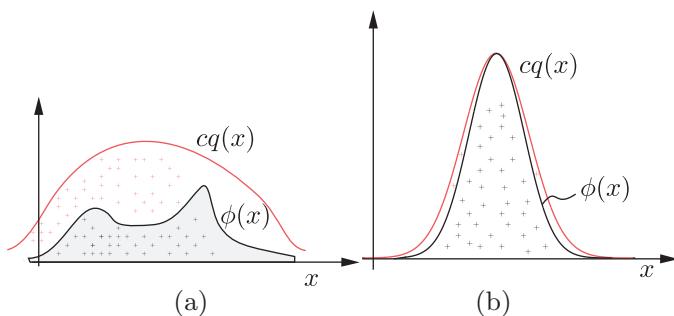
$$u_i \leq \phi(x_i).$$

- **End For**

The probability of accepting a point, \mathbf{x} , is given by

$$\text{Prob}\{u \leq \phi(\mathbf{x})\} = \frac{1}{cq(\mathbf{x})} \phi(\mathbf{x}),$$

² If $q(\mathbf{x}) = 0$ in an interval, then $\phi(\mathbf{x})$ should be zero there.

**FIGURE 14.6**

(a) If $cq(x)$ is much larger than $\phi(x)$ most of the samples are rejected (red points) which is inefficient. (b) If $cq(x)$ and $\phi(x)$ have a good match, most of the samples are retained.

and the total probability, over all the possible values of x , for accepting samples is equal to

$$\text{Prob}\{\text{acceptance}\} = \frac{1}{c} \int \frac{\phi(x)}{q(x)} q(x) dx = \frac{1}{c} \int \phi(x).$$

Hence, if c has a large value, only a small percentage of points is finally retained. In order to have a practical algorithm, $cq(x)$ must be chosen in order to be a good fit of $\phi(x)$. Figure 14.6a is an example of a bad choice, while Figure 14.6b corresponds to a good example. This is a reason that rejection sampling *does not scale well with dimensionality*. In high dimensions, guaranteeing that $cq(x) \geq \phi(x)$ may oblige us to select a c with an excessively large value (Problem 14.5).

Besides the basic rejection scheme, a number of variants have also been proposed, in order to overcome the difficulty of selecting a proposal distribution that “looks like” the desired one. *Adaptive rejection sampling* is such a technique (see, e.g., [11] and the references therein). According to this method, the proposal distribution is adaptively constructed, based on the derivatives of $\ln p(x)$. For log-concave functions, this is a nondecreasing function and can be used to construct an envelope function of $p(x)$.

Although rejection sampling is not appropriate for difficult tasks, still it has been used, sometimes in its more refined forms, to generate samples from a number of standard distributions, such as the Gaussian, gamma, and student’s-t (see, e.g., [22]).

14.5 IMPORTANCE SAMPLING

Importance sampling (IS) is a method for estimating expectations. Let $f(\mathbf{x})$ be a known function of a random vector variable, \mathbf{x} , which is distributed according to $p(\mathbf{x})$. If one could draw samples from $p(\mathbf{x})$, then the expectation in Eq. (14.1) could be approximated as in Eq. (14.2). We will now assume that we are not able to draw samples from $p(\mathbf{x})$, and to go one step further, assume that $p(\mathbf{x})$ is only known up to a normalizing constant, that is,

$$p(\mathbf{x}) = \frac{1}{Z} \phi(\mathbf{x}).$$

Let $q(\mathbf{x})$ be another distribution from which samples can be drawn. Then we can write

$$\begin{aligned}\mathbb{E}[f(\mathbf{x})] &= \frac{1}{Z} \int_{-\infty}^{\infty} f(\mathbf{x}) \phi(\mathbf{x}) d\mathbf{x} = \frac{1}{Z} \int_{-\infty}^{\infty} f(\mathbf{x}) \frac{\phi(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) d\mathbf{x} \\ &\simeq \frac{1}{NZ} \sum_{i=1}^N f(\mathbf{x}_i) w(\mathbf{x}_i),\end{aligned}\tag{14.25}$$

where \mathbf{x}_i , $i = 1, 2, \dots, N$, are samples drawn from $q(\mathbf{x})$ and

$$w(\mathbf{x}) := \frac{\phi(\mathbf{x})}{q(\mathbf{x})}.\tag{14.26}$$

The normalizing constant can readily be obtained as

$$Z = \int_{-\infty}^{\infty} \phi(\mathbf{x}) d\mathbf{x} = \int_{-\infty}^{\infty} \left(\frac{\phi(\mathbf{x})}{q(\mathbf{x})} \right) q(\mathbf{x}) d\mathbf{x} \simeq \frac{1}{N} \sum_{i=1}^N w(\mathbf{x}_i).\tag{14.27}$$

Combining Eqs. (14.25) and (14.27), we finally obtain

$$\mathbb{E}[f(\mathbf{x})] \simeq \frac{\sum_{i=1}^N w(\mathbf{x}_i) f(\mathbf{x}_i)}{\sum_{i=1}^N w(\mathbf{x}_i)},\tag{14.28}$$

or

$$\mathbb{E}[f(\mathbf{x})] \simeq \sum_{i=1}^N W(\mathbf{x}_i) f(\mathbf{x}_i) : \quad \text{Importance Sampling Approximation},$$

where $W(\mathbf{x}_i) = \frac{w(\mathbf{x}_i)}{\sum_{i=1}^N w(\mathbf{x}_i)}$ are the normalized weights. It is not difficult to show (Problem 14.6) that the estimate

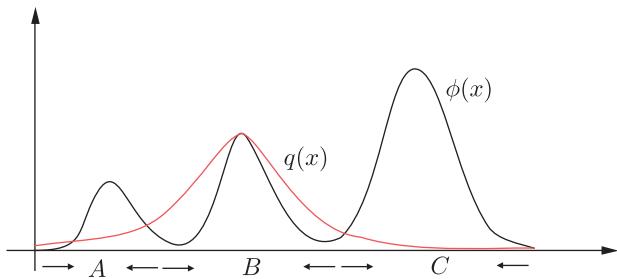
$$\hat{Z} = \frac{1}{N} \sum_{i=1}^N w(\mathbf{x}_i)\tag{14.29}$$

corresponds to an unbiased estimator of the normalizing constant. This is very interesting, because computing the normalizing constant is particularly useful information in a number of tasks. Recall that the evidence function, discussed in Chapter 12, is a normalizing constant; see also [26] for related comments.

In contrast, the estimator associated with Eq. (14.28), being the result of a ratio, is unbiased only asymptotically and it is a *biased* one for finite values of N (Problem 14.6). Hence, if one would have the luxury of a very large number N of samples, Eq. (14.28) would be a good enough estimate. However, in practice, N cannot be made arbitrarily large and the resulting estimate may not be satisfactory.

If $q(\mathbf{x}) \simeq p(\mathbf{x})$, or at least $q(\mathbf{x})$ is a fairly good approximation of $\phi(\mathbf{x})$, then Eq. (14.28) would approximate Eq. (14.2). However, for most practical cases, this is not easy to obtain, especially in high-dimensional spaces. If $q(\mathbf{x})$ is not a good match to $\phi(\mathbf{x})$, it is very likely that there will be regions where $\phi(\mathbf{x})$ is large while $q(\mathbf{x})$ is much smaller. The corresponding weights will have large values, relative to those from other regions, and they will be the dominant ones in the summation (Eq. (14.28)).

The effect of it is equivalent to reducing the number, N , of samples. Moreover, it is also possible that $q(\mathbf{x})$ takes very small values in some regions, which makes it very likely that samples from such regions

**FIGURE 14.7**

If $q(\mathbf{x})$ is not a good match of $\phi(\mathbf{x})$, a number of undesired effects appear. Samples from region A will give rise to weights of much larger values compared to these in region B. Due to the extremely low values of $q(\mathbf{x})$ in C, it is highly likely that given the finite size of the number of the samples, N , no samples will be drawn from this region, in spite of the fact that this is the most dominant region for $\phi(\mathbf{x})(p(\mathbf{x}))$.

are completely absent in Eq. (14.28); see Figure 14.7. In such cases, not only may the resulting estimate be wrong, but we will not be aware of it, and the variance of the weights, $w(\mathbf{x}_n)$ and $w(\mathbf{x}_n)f(\mathbf{x}_n)$, may exhibit low values. *These phenomena are accentuated in high-dimensional spaces* (see, e.g., [26] and Problem 14.7).

To alleviate the previous shortcomings, a number of variants have been proposed to search for high-probability regions and make local approximations around the modes and use them in order to generate samples (see, e.g., [30] and the references therein).

14.6 MONTE CARLO METHODS AND THE EM ALGORITHM

In Section 12.5.1, the EM algorithm was introduced for maximizing the log-likelihood function when some of the variables are hidden or missing. During the E-step (Eq. (12.61)), the function $\mathcal{Q}(\cdot, \cdot)$ is computed, which at the $(j + 1)$ th iteration step of the algorithms is written as

$$\begin{aligned}\mathcal{Q}(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)}) &= \mathbb{E} [\ln p(\mathcal{X}^l, \mathcal{X}; \boldsymbol{\xi})] \\ &= \int p(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi}^{(j)}) \ln p(\mathcal{X}^l, \mathcal{X}; \boldsymbol{\xi}) d\mathcal{X}^l,\end{aligned}\quad (14.30)$$

where \mathcal{X}^l is the set of hidden variables, \mathcal{X} the set of observed values, and $\boldsymbol{\xi}$ the unknown set of parameters. In case the computation of the integral is not tractable, Monte Carlo techniques can be mobilized to generate L samples for the hidden variables, $\mathcal{X}_1^l, \dots, \mathcal{X}_L^l$, from the posterior $p(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\xi}^{(j)})$ and obtain an approximation

$$\hat{\mathcal{Q}}(\boldsymbol{\xi}, \boldsymbol{\xi}^{(j)}) \approx \frac{1}{L} \sum_{i=1}^L \ln p(\mathcal{X}_i^l, \mathcal{X}; \boldsymbol{\xi}).\quad (14.31)$$

Maximization with respect to $\boldsymbol{\xi}$ is now carried out via $\hat{\mathcal{Q}}(\cdot, \cdot)$.

A specific form of Monte Carlo EM results in the context of mixture modeling and it is known as *stochastic EM*. The idea is to generate a *single* sample from the posterior (which now refers to the

labels of the mixtures) and assign corresponding observations in the respective mixtures. That is, a hard assignment takes place. The M-step is then applied based on this approximation [3].

14.7 MARKOV CHAIN MONTE CARLO METHODS

As we have already discussed, a major drawback associated with rejection as well as importance sampling methods is that they cannot tackle tasks in high-dimensional spaces very well.

In this section, we will deal with methods that scale well with the dimensionality of the sample space. Such techniques build upon arguments that come from the theory of Markov chains; we start by presenting some definitions and basics related to this important theory. Hidden Markov models, treated in [Section 16.5](#), are instances of Markov chains. Here we will shed more light on such models from a different perspective.

Markov chains/processes are named after the Russian mathematician Andrey Andreyevich Markov (1856–1922), who contributed seminal papers in the field of stochastic processes. As a professor at Saint Petersburg University during the students’ riots in 1908, he refused the government’s order to monitor and spy on his students, and he retired from the university.

Definition 14.1. A *Markov chain* is a sequence of random (vector) variables, $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2 \dots$ with conditional distributions that obey the rule

$$p(\mathbf{x}_n | \mathbf{x}_{n-1}, \{\mathbf{x}_t : t \in \mathcal{I}\}) = p(\mathbf{x}_n | \mathbf{x}_{n-1}), \quad (14.32)$$

where $\mathcal{I} = \{0, 1, \dots, n-2\}$. The index n is usually interpreted as time.

In words, Eq. (14.32) says that \mathbf{x}_n is independent of the variables with indices in \mathcal{I} , given the values of the variables in \mathbf{x}_{n-1} . The distribution, p , can be either a density function or a probability distribution corresponding to discrete variables, taking values in a discrete set, known as *states*. We will assume that all variables share a common range, known as *state space*. Most of our discussion will evolve along finite state spaces, where states take values in a finite discrete set, say, $\{1, 2, \dots, K\}$. A Markov chain is specified in terms of (a) the distribution (vector of probabilities), \mathbf{p}_0 , associated with the first vector in the sequence, \mathbf{x}_0 ; and (b) the $K \times K$ matrices of the *transition probabilities*, that is,

$$P_n(\mathbf{x}_n | \mathbf{x}_{n-1}) = [P_n(i|j)],$$

where

$$P_n(i|j) := P_n(\mathbf{x}_n = i | \mathbf{x}_{n-1} = j), \quad i, j = 1, 2, \dots, K,$$

denotes the probability of the variable at time $n-1$ to be at state j and the variable at time n to be at state i .³ Given the matrix of the transition probabilities, we write

$$\mathbf{p}_n = P_n(\mathbf{x}_n | \mathbf{x}_{n-1}) \mathbf{p}_{n-1}, \quad (14.33)$$

where

$$\mathbf{p}_n := [P(\mathbf{x}_n = 1), P(\mathbf{x}_n = 2), \dots, P(\mathbf{x}_n = K)]^T \quad (14.34)$$

$$:= [P_n(1), \dots, P_n(K)]^T \quad (14.35)$$

³ Note that equality here, $\mathbf{x}_n = i$, means that the (vector) variable \mathbf{x}_n is at state i .

is the vector of the respective probabilities at time n . The Markov chain is said to be *homogeneous* or *stationary* if the transition matrix is independent on time, that is,

$$P_n(x_n = i | x_{n-1} = j) = P(i|j) := P_{ij}, \quad i, j = 1, 2, \dots, K,$$

and

$$P_n(x_n | x_{n-1}) = P = [P_{ij}].$$

In this case, we can write

$$\mathbf{p}_n = P\mathbf{p}_{n-1} = P^2\mathbf{p}_{n-2} = \dots = P^n\mathbf{p}_0, \quad (14.36)$$

or equivalently,

$$P_n(i) = \sum_{j=1}^K P_{ij} P_{n-1}(j). \quad (14.37)$$

In the sequel, we will focus on stationary Markov chains.

Properties of the Transition Probabilities Matrix. The transition matrix has a special structure leading to certain properties, which will be used later on.

- The matrix P is a *stochastic* matrix. That is, all its entries are nonnegative, and the entries across each column add to one, that is,

$$\sum_{i=1}^K P_{ij} = 1,$$

which is a direct consequence of the definition of probabilities.

- The value $\lambda = 1$ is always an eigenvalue of P ([Problem 14.8](#)). Moreover, there is no eigenvalue with magnitude larger than one ([Problem 14.9](#)).
- The eigenvectors corresponding to eigenvalues $\lambda \neq 1$ comprise components that add to zero ([Problem 14.10](#)).
- The left eigenvector corresponding to $\lambda = 1$,

$$\mathbf{b}_1^T P = \mathbf{b}_1^T, \quad (14.38)$$

has all its elements equal. This is easily verified by plugging in $\mathbf{b}_1 = [1, 1, \dots, 1]^T$ and checking that this is indeed an eigenvector.

- *Invariant distribution.* A distribution is said to be invariant over the states of a Markov chain if

$$\mathbf{p} = P\mathbf{p}.$$

Note that \mathbf{p} is necessarily an eigenvector corresponding to the eigenvalue $\lambda = 1$. Moreover, because \mathbf{p} consists of probabilities, its elements must add to one. Depending on the multiplicity of $\lambda = 1$, there may be more than one invariant distribution. For example, if $P = I$ any probability distribution is an invariant for the respective Markov chain. It turns out that any Markov chain with a finite number of states has at least one invariant distribution. However, if the elements of P are strictly positive, then there is a unique invariant distribution that coincides with the unique eigenvector corresponding to the maximum eigenvalue, $\lambda = 1$, which in this case has a multiplicity of one. Furthermore, the eigenvector corresponding to $\lambda = 1$ comprises positive elements, which after scaling can always be made to add to one, and it is a by-product of the celebrated

Perron-Frobenius theorem, which is assured if P has strictly positive entries⁴ (see, e.g., [32]). We will cover invariant distributions in more detail soon.

- *Detailed Balanced Condition.* Let P be the transition probability matrix of a stationary Markov chain. Let also $\mathbf{p} = [P_1, \dots, P_K]^T$ be the set of probabilities describing a discrete distribution. We say that the *detailed balanced condition* is satisfied if

$$P(i|j)P_j = P(j|i)P_i. \quad (14.39)$$

That is, there exists a type of *symmetry*. If this condition holds, then the respective distribution is invariant for the Markov chain. Indeed,

$$\sum_{j=1}^K P(i|j)P_j = \sum_{j=1}^K P(j|i)P_i = P_i, \quad (14.40)$$

or

$$\mathbf{p} = P\mathbf{p}. \quad (14.41)$$

Although this is not a necessary condition for distribution invariance, it is very useful in practice; it helps us construct Markov chains with a desired invariant distribution. As we will soon see, this will be the type of distributions from which we want to draw samples.

14.7.1 ERGODIC MARKOV CHAINS

We now turn our attention to a specific type of Markov chains, which are known as *ergodic*. Such chains have a *unique* invariant distribution, which can be obtained as the limit

$$\lim_{n \rightarrow \infty} \mathbf{p}_n = \lim_{n \rightarrow \infty} P^n \mathbf{p}_0,$$

which is *independent* of the choice of the initial values in \mathbf{p}_0 . We will now focus on a class of ergodic processes and elaborate on their convergence.

Let us consider a stationary Markov chain, with a transition matrix P , with eigenvalues $1 = \lambda_1 > |\lambda_2| \geq \dots \geq |\lambda_K|$. That is, only one eigenvalue has the maximum value and the rest have magnitude strictly less than one. Moreover, we assume that one can find a complete set of *linearly independent* eigenvectors. Such assumptions are not restrictive and hold true for a wide class of stochastic matrices. Then, we can write that

$$P = A \Lambda A^{-1}, \quad (14.42)$$

where Λ is the diagonal matrix $\Lambda = \text{diag}\{1, \lambda_2, \dots, \lambda_K\}$ and A has as columns the respective eigenvectors. Hence, from Eq. (14.36) we get

$$\mathbf{p}_n = A \Lambda^n A^{-1} \mathbf{p}_0 = A \begin{bmatrix} 1 & & & \\ & \lambda_2^n & & \\ & & \ddots & \\ & & & \lambda_K^n \end{bmatrix} A^{-1} \mathbf{p}_0,$$

with $\lambda_k^n \rightarrow 0, k = 2, \dots, K$.

⁴ This is also true for a class of matrices with nonnegative elements, known as *primitive* matrices. That is, there exists an n such that P^n has positive elements.

Hence,

$$\mathbf{p}_\infty := \lim_{n \rightarrow \infty} \mathbf{p}_n = P_\infty \mathbf{p}_0, \quad (14.43)$$

where

$$P_\infty = A \begin{bmatrix} 1 & & & \\ & 0 & & \mathbf{0} \\ & & \ddots & \\ \mathbf{0} & & & 0 \end{bmatrix} A^{-1} = \mathbf{a}_1 \mathbf{b}_1^T, \quad (14.44)$$

with \mathbf{a}_1 being the first eigenvector (first column of A), corresponding to $\lambda = 1$ and \mathbf{b}_1^T the first row of A^{-1} . In other words, P_∞ is a rank-one matrix. However, it is straightforward to see from Eq. (14.42) ($A^{-1}P = \Lambda A^{-1}$) that \mathbf{b}_1^T is a left eigenvector of P , that is,

$$\mathbf{b}_1^T P = \mathbf{b}_1^T,$$

and recalling the properties (Eq. (14.38) and the comments just after it) of P , $\mathbf{b}_1^T = [1, 1, \dots, 1]$ (within a proportionality constant, c). Thus,

$$P_\infty = [\mathbf{a}_1, \dots, \mathbf{a}_1],$$

and from Eq. (14.43), because the components of \mathbf{p}_0 add to one, we finally obtain that

$$\mathbf{p}_\infty = \mathbf{a}_1.$$

That is, the limiting distribution is equal (after scaling) to the unique eigenvector of P corresponding to $\lambda_1 = 1$; moreover, this is true irrespective of the values of \mathbf{p}_0 . In other words, the limiting distribution is the invariant distribution of P , that is,

$$P\mathbf{p} = \mathbf{p}. \quad (14.45)$$

Note that the convergence rate is controlled by the magnitude of $|\lambda_2|$. Other, more theoretically refined convergence results and bounds can be found in, for example, [24, 39, 40].

Remarks 14.2.

- Needless to say, not all Markov chains are ergodic. For example, if the eigenvalue $\lambda_1 = 1$ of the transition matrix has multiplicity higher than one, then the limiting distribution depends on the values of the initial choice in \mathbf{p}_0 . On the other hand, if the transition matrix has more than one eigenvalue with magnitude equal to one (e.g., $\lambda_1 = 1, \lambda_2 = -1$) then, again, it does not have a limiting distribution but instead exhibits a *periodic* limit cycle (e.g., [32]).
- *Building Markov Chains.* In practice, one can construct transition probability matrices for ergodic chains using a set of simpler transition matrices, B_1, B_2, \dots, B_M , which are known as *base transition* matrices. Each one of them may not be ergodic, but it is required to accept the desired distribution as its invariant. Then, the transition matrix is built as

$$P = \sum_{m=1}^M \alpha_m B_m, \quad \alpha_m > 0, \quad \sum_{m=1}^M \alpha_m = 1.$$

If a distribution is invariant with respect to each B_m , $m = 1, 2, \dots, M$, it will be invariant for P . The same applies with the detailed balance condition.

Another way is to combine individual transition matrices sequentially, that is,

$$P = B_1 B_2 \cdots B_M.$$

For example, each B_m , $m = 1, 2, \dots, M$, may act and change a subset of the random entries comprising the random vector \mathbf{x} . We will see that this is the case with the Gibbs sampling method, to be reviewed soon. It is easy to see that if p is invariant for each individual B_m , $m = 1, 2, \dots, M$, will also be invariant for P .

- In this section, we focused our discussion on Markov chains with finite state space. Everything we have said can be generalized to Markov chains with countably infinite or continuous state spaces. In the latter case, the place of the probability transition matrix is taken by the transition density or kernel, $p(\mathbf{x}_n | \mathbf{x}_{n-1})$, and the probability density of \mathbf{x}_n , at time n , is given by

$$p_n(\mathbf{x}) = \int p(\mathbf{x} | \mathbf{y}) p_{n-1}(\mathbf{y}) d\mathbf{y}. \quad (14.46)$$

The analysis in this case is more difficult and care has to be taken because not all results obtained for the finite discrete case are readily valid for the continuous one. The reason we focused on the discrete finite state space is that one can get the feeling of the theory of Markov chains by spending less “budget” on the required mathematical effort.

Example 14.4. Consider the Markov chain with the transition probability matrix

$$P = \begin{bmatrix} 0.2 & 0.4 & 0.6 \\ 0.5 & 0.1 & 0.3 \\ 0.3 & 0.5 & 0.1 \end{bmatrix}.$$

Its eigenvalues are $\lambda_1 = 1$, $\lambda_2 = -0.3 + 0.1732j$, $\lambda_3 = -0.3 - 0.1732j$, and the respective eigenvectors are

$$\begin{aligned} \mathbf{a}_1 &= [0.6608, 0.5406, 0.5206]^T, \\ \mathbf{a}_2 &= [0.5774, -0.2887 - 0.5j, -0.2887 + 0.5j]^T, \\ \mathbf{a}_3 &= [0.5774, -0.2887 + 0.5j, -0.2887 - 0.5j]^T. \end{aligned}$$

Observe that all the elements of the eigenvector corresponding to $\lambda = 1$ are positive. Also, the elements of the other two eigenvectors add to zero.

We can now write that

$$\begin{aligned} P &= \begin{bmatrix} 0.6608 & 0.5774 & 0.5774 \\ 0.5406 & -0.2887 - 0.5j & -0.2887 + 0.5j \\ 0.5206 & -0.2887 + 0.5j & -0.2887 - 0.5j \end{bmatrix} \\ &\times \begin{bmatrix} 1 & 0 & 0 \\ 0 & -0.3 + 0.1732j & 0 \\ 0 & 0 & -0.3 - 0.1732j \end{bmatrix} \\ &\times \begin{bmatrix} 0.5807 & 0.5807 & 0.5807 \\ 0.5337 - 0.0058i & -0.3323 + 0.4942j & -0.3323 - 0.5058j \\ 0.5337 + 0.0058i & -0.3323 - 0.4942j & -0.3323 + 0.5058j \end{bmatrix}. \end{aligned}$$

Observe that the first row of the last matrix (A^{-1}) has all its elements equal. Having written P in a product form, it is easily obtained that

$$P^2 = \begin{bmatrix} 0.42 & 0.42 & 0.30 \\ 0.24 & 0.36 & 0.36 \\ 0.34 & 0.22 & 0.34 \end{bmatrix},$$

$$P^{10} = \begin{bmatrix} 0.3837 & 0.3837 & 0.3837 \\ 0.3140 & 0.3140 & 0.3139 \\ 0.3023 & 0.3023 & 0.3029 \end{bmatrix}.$$

The sequence has converged for $n = 10$. Note that after convergence, P^n has all its column vectors equal, and the elements add to one. Moreover, observe that

$$P_\infty \propto [\mathbf{a}_1, \mathbf{a}_1, \mathbf{a}_1].$$

Example 14.5. *Random walk with finite states.* Random walks are popular models that can model faithfully a number of real-world phenomena, such as thermal noise, the motion of gas molecules, and stock value variations. Moreover, such chains can help us understand the behavior of more complex Markov chains, to be discussed soon. There are various random walk models, depending on the choice of the transition probabilities (see, e.g., [32]). Here, we assume the variables to be discrete and take integer values in a finite set, $[0, N]$. Hence, the total number of states is $N + 1$. At every time instant, the value of the variable can either increase or decrease by one with probability p , respectively, or stay unchanged, with probability q , provided that the current state is in the interval $[1, N - 1]$. That is, if $0 < x_{n-1} < N$,

$$P(x_n = x_{n-1} + 1) = P(x_n = x_{n-1} - 1) = p,$$

$$P(x_n = x_{n-1}) = q.$$

If $x_{n-1} = 0$, then x_n can either stay in the same state with probability q_e or increase by one with probability p . If $x_{n-1} = N$, then x_n can either stay in the same state with probability q_e or decrease by one with probability p . Obviously,

$$2p + q = 1, \quad p + q_e = 1.$$

The transition probability matrix, for the case of $N = 4$, $p = \frac{1}{4}$, $q = \frac{1}{2}$, $q_e = \frac{3}{4}$, is

$$P = \begin{bmatrix} 3/4 & 1/4 & 0 & 0 & 0 \\ 1/4 & 1/2 & 1/4 & 0 & 0 \\ 0 & 1/4 & 1/2 & 1/4 & 0 \\ 0 & 0 & 1/4 & 1/2 & 1/4 \\ 0 & 0 & 0 & 1/4 & 3/4 \end{bmatrix}.$$

The respective eigenvalues are $\lambda_1 = 1$, $\lambda_2 = 0.904$, $\lambda_3 = 0.654$, $\lambda_4 = 0.345$, $\lambda_5 = 0.095$. Observe that all eigenvalues, except $\lambda_1 = 1$, have magnitude less than one. The corresponding eigenvectors are

$$\mathbf{a}_1 = [0.447, 0.447, 0.447, 0.447, 0.447]^T$$

$$\mathbf{a}_2 = [-0.601, -0.371, 0, 0.371, 0.601]^T$$

$$\mathbf{a}_3 = [-0.511, 0.195, 0.632, 0.195, -0.511]^T$$

$$\mathbf{a}_4 = [-0.371, 0.6015, 0, -0.601, 0.371]^T$$

$$\mathbf{a}_5 = [0.195, -0.511, 0.632, -0.511, 0.195]^T.$$

The eigenvector corresponding to λ_1 has all its components equal and positive. Hence, the invariant distribution ($\mathbf{p}: \mathbf{P}\mathbf{p} = \mathbf{p}$), after the required scaling, becomes the uniform one, $\mathbf{p} = [1/5, 1/5, 1/5, 1/5, 1/5]^T$. Similar arguments apply for any value of N . Observe that the components of all the other eigenvectors add to zero.

[Figure 14.8](#) shows the probability distribution \mathbf{p}_n for the case of $N = 4$, at times $n = 10, 50$, and 100 . The components of \mathbf{p}_0 were randomly chosen. [Figure 14.9](#) corresponds to the case of $N = 9$. Observe that the larger the value of N , the slower the convergence.

Example 14.6. In this example, we consider a random walk with (countable) infinite many states; at every time instant, the value of the random variable can either increase or decrease by one, with probability p , respectively, or stay in the same state with probability q , that is,

$$P(x_n = x_{n-1} + 1) = P(x_n = x_{n-1} - 1) = p,$$

$$P(x_n = x_{n-1}) = q,$$

and

$$2p + q = 1.$$

The difference with the previous example is that now there are not “barrier” points and the random variable can take any integer value. Our goal is to compute the mean and variance as functions of time n , when the starting point is deterministically chosen to be $x_0 = 0$.

It is readily seen that $\mathbb{E}[x_n] = 0$, because the variable is equally likely to increase or decrease and hence it is equally likely to assume any positive or negative value.

For the variance, we obtain ([Problem 14.11](#))

$$\begin{aligned} \mathbb{E}[x_n^2] &= \mathbb{E}[x_{n-1}^2] + 2p \\ &= 2pn + \mathbb{E}[x_0^2] = 2pn, \end{aligned} \tag{14.47}$$

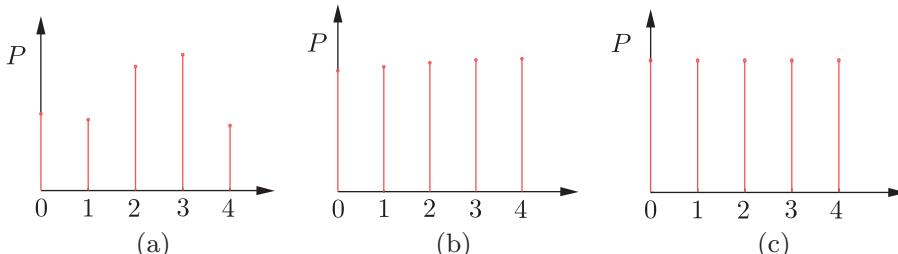
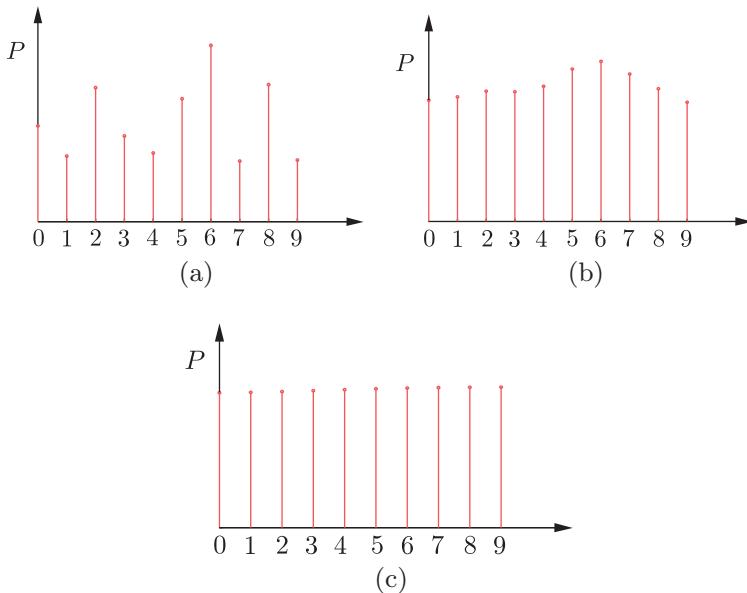


FIGURE 14.8

The probability distribution for the random walk chain of [Example 14.5](#) for $N = 4$ and at time instants (a) $n = 10$, (b) $n = 50$, and (c) $n = 100$.

**FIGURE 14.9**

The probability distribution for the random walk chain of [Example 14.5](#) for $N = 9$ and at time instants (a) $n = 10$, (b) $n = 50$, and (c) $n = 100$. Compared to [Figure 14.8](#), observe that the higher the number N the slower the convergence.

because $\mathbb{E}[x_0^2] = 0$. Note that the variance tends to infinity with time, hence the infinite state-space random walk *does not* have a limiting distribution. This verifies what we said before; results that hold true for finite state-spaces do not necessarily carry on to the case where the number of states becomes infinite.

Looking at Eq. (14.47) more carefully reveals that, on average, after n time instants, x_n would be within $\pm\sqrt{2pn}$. If x_n denotes the distance of a point from the origin from where it starts and moves backward or forward, then the distance it travels is proportional only to the square root of the time it has spent traveling. Although this result has been derived for the infinite state-space case, still it can shed light on the slow convergence to the invariant distribution that we saw in the previous example, for the case of finite state space. As stated in [30], convergence to the invariant distribution can be achieved once all points in the state space have been visited, and this has a square root dependence on time. In order to get a good enough approximation of the limited distribution, one must be patient enough to compute $\mathcal{O}(N^2)$ iterations.

14.8 THE METROPOLIS METHOD

The Metropolis method or algorithm, as it is sometimes called, builds upon a surprisingly simple idea, and it is the first method that exploited the Markov chain theory for sampling. It appeared in the classical paper [27] and it may be the most popular and widely known sampling technique, which has inspired a wealth of variants. In contrast to rejection and importance sampling, the proposal distribution is now

time varying, following the evolution of a Markov chain; the latter is constructed such as its transition probability matrix/density kernel to have the desired distribution $p(\mathbf{x})$ as its invariant. Moreover, in contrast to the rejection and importance sampling techniques, it is not required that the proposal distribution “look like” the desired one in order for the method to be useful in practice. The proposal distribution depends on the value of previous state, \mathbf{x}_{n-1} , that is, $q(\cdot|\mathbf{x}_{n-1})$. In words, drawing a new sample (generating a new state) depends on the value of the previous one. In its original version, the proposal distribution was chosen to be symmetric, that is,

$$q(\mathbf{x}|\mathbf{y}) = q(\mathbf{y}|\mathbf{x}).$$

Later on, it was generalized by Hastings [14] to include nonsymmetric ones. The general scheme is known as the Metropolis-Hastings algorithm, which is summarized next.

Algorithm 14.3 (Metropolis-Hastings algorithm).

- Let the desired distribution be $p(\cdot) = \frac{1}{Z}\phi(\cdot)$.
- Choose the proposal distribution to be $q(\cdot|\cdot)$.
- Choose the value of the initial state \mathbf{x}_0 .
- **For** $n = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x} \sim q(\cdot|\mathbf{x}_{n-1})$
 - Compute the acceptance ratio

$$\alpha(\mathbf{x}|\mathbf{x}_{n-1}) = \min \left\{ 1, \frac{q(\mathbf{x}_{n-1}|\mathbf{x})\phi(\mathbf{x})}{q(\mathbf{x}|\mathbf{x}_{n-1})\phi(\mathbf{x}_{n-1})} \right\}$$

- Draw $u \sim \mathcal{U}(0, 1)$
- **If** $u \leq \alpha(\mathbf{x}|\mathbf{x}_{n-1})$
 - $\mathbf{x}_n = \mathbf{x}$
- **Else**
 - $\mathbf{x}_n = \mathbf{x}_{n-1}$
- **End For**

The following points are readily deduced from the algorithm:

- The algorithm does not need the exact form of p . It suffices to know it up to its normalizing constant Z . This is due to the fact that p enters into the algorithm only in the ratio for computing the acceptance ratio.
- If the proposal distribution is symmetric, the acceptance ratio becomes

$$\alpha(\mathbf{x}|\mathbf{x}_{n-1}) = \min \left\{ 1, \frac{\phi(\mathbf{x})}{\phi(\mathbf{x}_{n-1})} \right\}, \quad (14.48)$$

and in this case, we sometimes refer to it as the Metropolis algorithm.

- Note that if a sample is not accepted, we retain the value of the previous state.
- Observe that a sample is accepted or rejected depending on the value of $\alpha(\mathbf{x}|\mathbf{x}_{n-1})$. This is easier understood by looking at the original form of the algorithm based on Eq. (14.48). If the probability $p(\mathbf{x})$ is larger than $p(\mathbf{x}_{n-1})$, then the new sample is accepted. If not, it is accepted/rejected based on its relative value.
- Successive samples are *not* independent.

There are variants of the previous basic scheme, concerning the choice of the function for the acceptance ratio. In [33], an argument in support of the rationale behind the Metropolis-Hastings scheme is based on an optimality proof concerning the variance of the obtained approximations.

Let us now turn our focus to understanding how the previously stated algorithm relates to the Markov chain theory. We will work with the more general continuous state-space models, and define

$$p(\mathbf{x}|\mathbf{y}) = q(\mathbf{x}|\mathbf{y})\alpha(\mathbf{x}|\mathbf{y}) + \delta(\mathbf{x} - \mathbf{y})r(\mathbf{x}), \quad (14.49)$$

where $r(\mathbf{x})$ is the rejection probability,

$$r(\mathbf{x}) = \int (1 - \alpha(\mathbf{x}|\mathbf{y}))q(\mathbf{x}|\mathbf{y}) \, d\mathbf{y}, \quad (14.50)$$

and $\delta(\cdot)$ is Dirac's delta function. A little thought reveals that $p(\cdot|\cdot)$, as defined above, is the transition density kernel (transition matrix for finite discrete spaces), $p(\mathbf{x}_n|\mathbf{x}_{n-1})$, for an equivalent Markov chain. Moreover, this Markov chain has the desired distribution $p(\mathbf{x})$ as its invariant, that is,

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{y})p(\mathbf{y}) \, d\mathbf{y},$$

which, as already pointed out in [Section 14.7](#), is a direct outcome of the fact that the following detailed balance condition is satisfied ([Problem 14.12](#)):

$$p(\mathbf{x}|\mathbf{y})p(\mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}).$$

It turns out that the equivalent Markov chain is ergodic, hence converging to the invariant (desired) distribution, provided that $p(\mathbf{x}|\mathbf{y})$ as well as $p(\mathbf{x})$ are *strictly positive*. This guarantees that any state has a nonzero probability to be reached starting from any state.

Hence, the Metropolis-Hastings algorithm equivalently draws samples from the Markov chain defined by the transition density given in Eq. (14.49), albeit the samples are drawn from the chosen (easily sampled) proposal distribution. Typical distributions used to play the role of the proposal distribution are the Gaussian and Cauchy distributions. The latter, due to its heavy-tail property, allows large changes to occur from time to time. Sometimes, the uniform distribution is also used. For the discrete case, the uniform distribution seems to be a popular choice.

Burn-in phase: After convergence, the process becomes equivalent with drawing samples from the desired $p(\mathbf{x})$! However, nothing is perfect in this world; a major weakness of the Markov chain Monte Carlo techniques is that it is difficult to assess whether the Markov chain *has converged*, and hence to be sure that the samples one generates are indeed effectively independent and truly representative of $p(\mathbf{x})$. Samples generated before the chain has converged are not representative of the desired distribution and have to be rejected; this is known as the *burn-in phase*. The interval that a Markov chain takes to converge is known as the *mixing time* (e.g., [21]).

To this end, a number of diagnostics have been proposed, though none of them can be considered a panacea (see, e.g., [6, 8, 35]) for a discussion. A theoretical justification concerning the difficulty of assessing convergence of such techniques is provided in [2], where it is shown that this is a computationally intractable task.

In practice, after the rejection of the samples during the burn-in phase, one runs a long chain and discards one out of, say, M samples. For large enough values of M , one expects to obtain independent samples. This process is also known as *thinning*. An alternative path is to run a few, say three to four, different (starting from different initial points) chains of medium size (e.g., 100,000) and take samples from each of them, having discarded the samples in the respective burn-in phases (e.g., the first half of them).

14.8.1 CONVERGENCE ISSUES

When dealing with the rejection and importance sampling, we discussed that these methods do not scale well with dimensionality. In contrast, the Metropolis approach shows much better behavior, and it is an algorithm that lends itself to applications in large spaces. Having said that, the method is not without shortcomings. To elaborate, we will use our experience gained from the random walk examples and employ similar arguments as those given in [30].

Consider a two-dimensional task and adopt as the proposal distribution, $q(\mathbf{x}|\mathbf{x}_{n-1})$, the Gaussian one with covariance matrix $\sigma^2 I$ and each time-centered at \mathbf{x}_{n-1} . The desired distribution, from which samples are to be drawn, is another elongated Gaussian $\mathcal{N}(\mathbf{x}|\mathbf{0}, \Sigma)$, as shown in Figure 14.10a. The values σ_{\max} , σ_{\min} denote the scales (standard deviations) associated with the two axes of the ellipse (recall from Chapter 2 that this is defined by the eigenstructure of Σ), which corresponds to the one standard deviation contour (the exponent in the Gaussian is equal to $-1/2$) of $p(\mathbf{x})$.

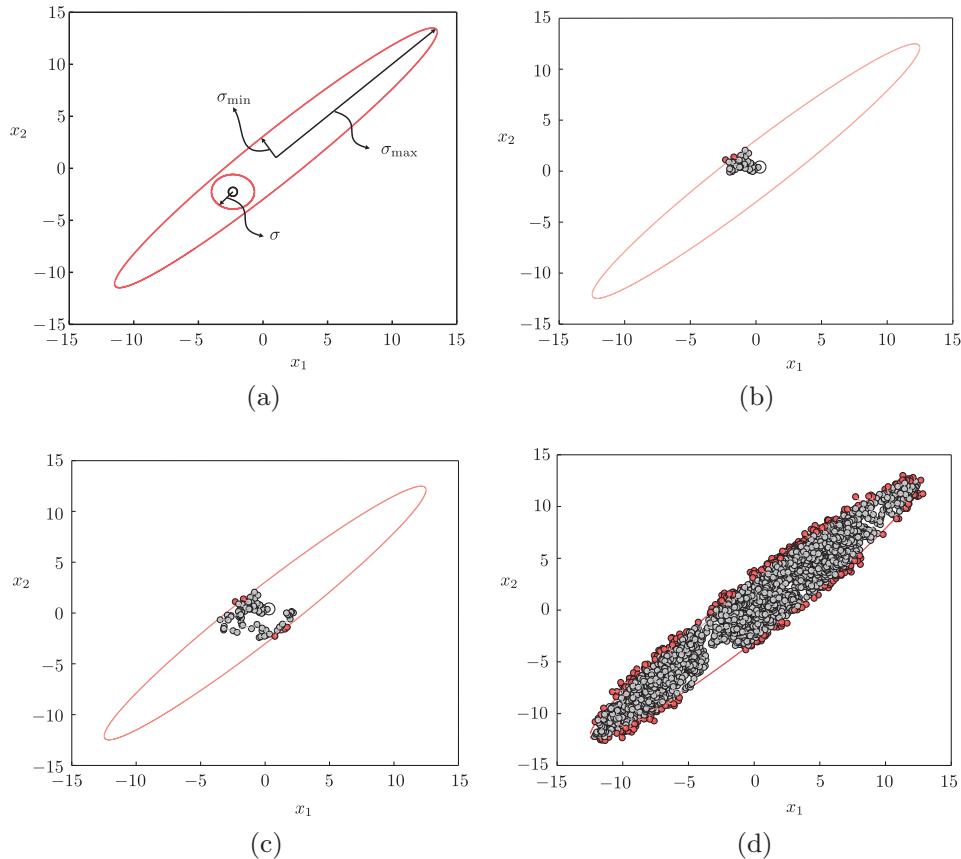
Every time a sample is drawn from $\mathcal{N}(\mathbf{x}|\mathbf{x}_{n-1}, \sigma^2 I)$, the new sample will be within the circle of radius σ around \mathbf{x}_{n-1} , with high probability. In order for the new sample to have a large chance to lie within the high-probability elliptical region, σ must be of the order of σ_{\min} or smaller. If σ is chosen to have a large value, there is high probability for the sample to end up outside the ellipse and be rejected. Hence, once sampling starts inside the ellipse, small values of σ guarantee, with high probability, that samples remain within the ellipse, and hence are accepted. On the other hand, if σ is small, a large number of iterations will be required in order to cover sufficiently with points the interior of the ellipse. If one looks at the process of sampling as a random walk, with approximate step size σ , then the number of iterations needed to cover a scale of the order of σ_{\max} will be $(\frac{\sigma_{\max}}{\sigma})^2$; if $\sigma \simeq \sigma_{\min}$, this becomes $(\frac{\sigma_{\max}}{\sigma_{\min}})^2$. In high dimensions, where there is high probability for one of the dimensions to be of relatively small scale compared to the maximum one, this square-dependence rule of thumb can slow convergence substantially.

Figures 14.10b, c, d, show the case where the desired two-dimensional Gaussian has zero mean and covariance matrix given by

$$\Sigma = \begin{bmatrix} 1.00 & 0.99 \\ 0.99 & 1.00 \end{bmatrix}.$$

The proposal distribution is a Gaussian with covariance matrix $0.1I$. The figures show three snapshots in the sequence of point generation, corresponding to 50, 100, and 3000 points. The rejected ones are denoted in red. The number of rejected points in Figure 14.10d is equal to 5%. This percentage increases to 13% if the proposal distribution has covariance matrix equal to $0.3I$.

Another problem that may arise with the Metropolis method is that of *local trapping*. This may occur when the desired distribution is multimodal, which is common in high-dimensional complex

**FIGURE 14.10**

(a) The region of significant probability mass is enclosed by the ellipse, with scales σ_{\max} and σ_{\min} , respectively, as defined by the major and minor axes. The region of significant probability mass of the proposal distribution is spherical of scale equal to σ , which is of the same order as σ_{\min} . (b) Starting from the point shown as a circle, 50 generated points are shown using a proposal distribution of covariance equal to $\sigma^2 = 0.1I$; rejected points are shown in red. (c) The snapshot with 100 points and d) with 3000 points. Observe that even in the latter case, still there are parts in the high-probability region of the desired distribution that have not been covered.

problems. We will demonstrate the case via a simple example in the two-dimensional space. Let the desired distribution comprise a mixture of two Gaussians,

$$p(\mathbf{x}) = \frac{1}{2}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_1, \Sigma_1) + \frac{1}{2}\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_2, \Sigma_2),$$

where $\boldsymbol{\mu}_1 = [0, 0]^T$, $\boldsymbol{\mu}_2 = [5, 5]^T$, $\Sigma_1 = \Sigma_2 = \text{diag}\{0.25, 2\}$, and the proposal distribution is $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, I)$, where $\boldsymbol{\mu} = [2.5, 2.5]^T$. Figure 14.11 shows the paths traveled by the drawn and accepted points over three different runs. In Figure 14.11a, b after 400 iterations the points drawn cover only one of the two mixtures. Both mixtures are visited in the run corresponding to Figure 14.11c.

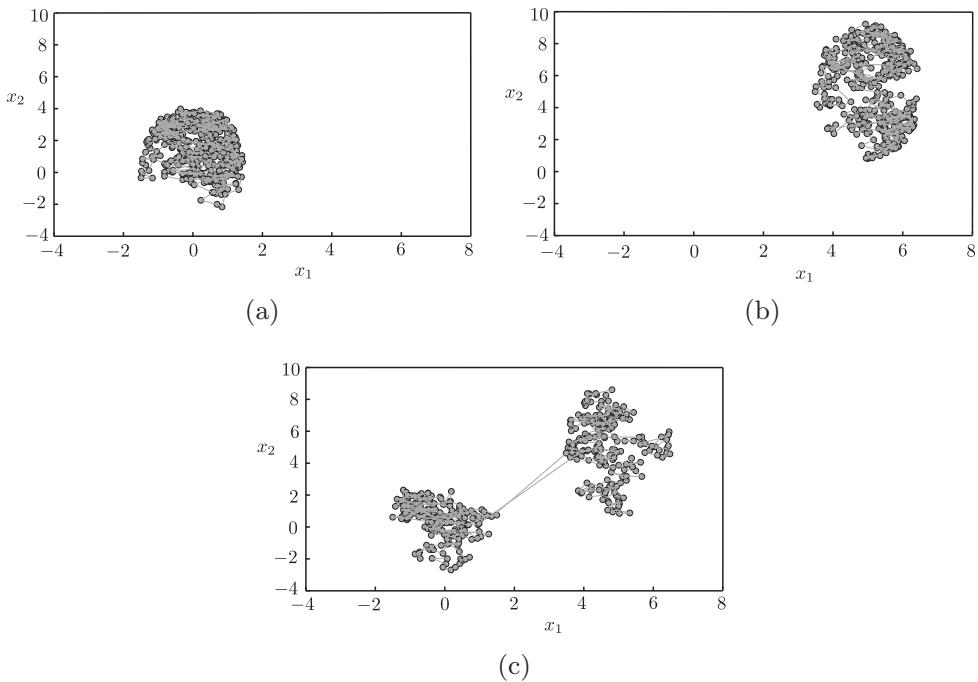


FIGURE 14.11

The desired distribution comprises the mixture of two Gaussians. In each figure, a different initialization point is selected. In all cases, 400 points have been generated. Note that in two of the cases, the process seems to be trapped in either one of the two Gaussians.

14.9 GIBBS SAMPLING

Gibbs sampling is among the most popular and widely used sampling methods. It is also known as the *heat bath* algorithm. Although Gibbs sampling was already known and used in statistical physics, two papers [9, 10] were catalytic for its widespread use in the Bayesian and machine learning communities.

Josiah Willard Gibbs (1839–1903) was an American scientist whose work on thermodynamics laid the foundations of physical chemistry. Together with James Clark Maxwell (1831–1879) and Ludwig Eduard Boltzmann (1844–1906), Gibbs pioneered the field of statistical mechanics. He is also the father, together with Oliver Heaviside (1850–1925), of what is today known as vector calculus.

Gibbs sampling is appropriate for drawing samples from multidimensional distributions, and it can be considered a special instance of the more general Metropolis method.

Let the random vector in the Markov chain at time n be given as

$$\mathbf{x}_n \equiv [\mathbf{x}_n(1), \dots, \mathbf{x}_n(l)]^T.$$

The basic assumption underlying Gibbs sampling is that the *conditional* distributions of each one of the variables, $x_n(d)$, $d = 1, 2, \dots, l$, given the rest, that is,

$$p(x_n(d) | \{x_n(i) : i \not\equiv d\}), \quad (14.51)$$

are known and they can be easily sampled. At each iteration, a sample is drawn for only one of the variables, based on Eq. (14.51), by freezing the values of the rest to those already available from the previous iteration. The scheme is summarized next.

Algorithm 14.4 (Gibbs sampler).

- Initialize, $x_0(1), \dots, x_0(l)$, arbitrarily.
- **For** $n = 1, 2, \dots, N$, **Do**
 - **For** $d = 1, 2, \dots, l$, **Do**
 - Draw

$$x_n(d) \sim p(x | \{x_n(i), i < d \neq 1\}, \{x_{n-1}(i), i > d \neq l\})$$

- **End For**
- **End For**

Note that in the previous scheme, all dimensions are visited in sequence. Another version is to visit them in a random order.

The Gibbs scheme can be viewed as a realization of a Markov chain, where the transition matrix/pdf is sequentially constructed from l base transitions, that is,

$$T = B_1 \cdots B_l,$$

where each individual base transition acts on the corresponding dimension, that is, coordinate-wise. For continuous variables, it can be readily checked out that

$$B_d(\mathbf{x}|\mathbf{y}) = p(x(d)|\{y(i)\} : i \neq d) \prod_{i \neq d} \delta(y(i) - x(i)), \quad d = 1, 2, \dots, l.$$

In words, only the component $x(d)$ changes while the rest are left unchanged. It is not difficult to see that the desired joint distribution $p(\mathbf{x}) = p(x(1), \dots, x(l))$ is invariant with respect to each one of B_d , $d = 1, 2, \dots, l$ (Problem 14.13). Hence, it will be invariant under their product

$$T = B_1 \cdots B_l.$$

Ergodicity is ensured by requiring that all conditional probabilities are *strictly* positive, which guarantees the convergence of the chain to the desired $p(\mathbf{x})$.

Remarks 14.3.

- Gibbs sampling, being an instance of the Metropolis method, inherits its random walk-like convergence performance.
- Gibbs sampling is appropriate for many graphical models (Chapter 16) that are described in terms of conditional distributions. Often, these distributions can be sampled in an easy way, using techniques such as rejection sampling and its variants, as discussed in Section 14.4.
- Note that in Gibbs sampling, no samples are rejected. This can also be shown if Gibbs sampling is considered as an instance of the Metropolis method, via the specific choice of Eq. (14.51) as the proposal distribution (Problem 14.14).
- *Blocking Gibbs sampling:* Gibbs sampling samples one variable at a time. This can make the algorithm move very slowly through the state-space in case the variables are highly correlated. In

such cases, it is preferable to sample *groups* of variables, not necessarily disjoint, and sample from the variables in the block, conditioned on the remaining. This is known as blocking Gibbs sampling [16], and it improves performance by achieving much bigger moves through the state-space.

- *Collapsed Gibbs Sampling:* In collapsed Gibbs sampling, one integrates out (marginalizes over) one or more variables and samples from the remaining ones. For example, in the case of three variables, Gibbs sampling samples from $p(x_1|x_2, x_3)$, then $p(x_2|x_1, x_3)$, and finally $p(x_3|x_1, x_2)$ to complete the iteration step. In collapsed Gibbs sampling, we can integrate out, for example, x_3 (which is *collapsed*), and sample sequentially from $p(x_1|x_2)$ and $p(x_2|x_1)$. Then sampling is performed in a lower dimensional space, hence it is more efficient. Collapsing one variable is tractable if it is a conjugate prior of another involved variable; for example, they are both members of the exponential family. Thus, x_3 does not participate in Gibbs sampling. In the sequel, we can sample $p(x_3|x_1, x_2)$. This can be justified by the *Rao-Blackwell* theorem, which states that the variance of the estimate created by analytically integrating out x_3 will always be lower than (or equal to) the variance of direct Gibbs sampling, [23].

14.10 IN SEARCH OF MORE EFFICIENT METHODS: A DISCUSSION

In order to sidestep the drawbacks associated with the described basic Markov chain-based schemes, namely the slow random walk-like convergence and the local-trap problem, a number of more advanced methods have been suggested. It is beyond the scope of this chapter to present such schemes in more detail and the interested reader may consult more specialized books and articles, that is [5, 22, 24, 30, 38]. Below, we provide a short discussion on some of the most popular directions that have been proposed.

A family of algorithms known as *auxiliary variable Markov chain Monte Carlo* methods is a popular one. Such methods augment with auxiliary variables either the desired or the proposal distribution in the Metropolis-Hastings algorithm. The presence of the auxiliary variable is intended to either help the algorithm to escape from possible local-traps, or to cancel out the normalizing constant, if this is intractable. Such methods include algorithms like the *simulating annealing*, [17]; the *simulated tempering*, [25]; and the *slice sampler*, [15]. The rationale behind the slice-sampling techniques builds around our discussion in Section 14.4; recall that sampling from $p(\mathbf{x})$ is equivalent with sampling uniformly from the region in

$$\mathcal{A} = \left\{ (\mathbf{x}, u) : \mathbf{x} \in \mathbb{R}^l, 0 \leq u \leq p(\mathbf{x}) \right\}.$$

In [31], a Gibbs-type implementation of the slice sampler is suggested; each component of \mathbf{x} is updated sequentially using a single-variable slicing sampling strategy. It turns out that the slice sampler improves upon the convergence speed of the standard Metropolis-Hastings algorithm.

In [29], an auxiliary variable is used so that the computation of the normalizing constant is bypassed. This is important in cases where its computation is intractable.

Another sampling philosophy spans the *population-based* methods. In order to overcome the local-trap problem, a number (population) of Markov chains are run in parallel under an information exchange strategy, which improves convergence. Typical examples of such techniques include *adaptive direction*

sampling, [12]; and the *evolutionary* Monte Carlo method, which builds upon arguments used in genetic algorithms [22].

Another direction is that of the *Hamiltonian* Monte Carlo methods, which exploit arguments from classical mechanics around the elegant Hamiltonian equations [26, 30]. For pdfs of the form

$$p(\mathbf{x}) = \frac{1}{Z_E} \exp(-E(\mathbf{x})),$$

$E(\mathbf{x})$ can be given the interpretation of the system's potential energy (Section 15.4.2). Once such a bridge has been established, an auxiliary random vector, \mathbf{q} , is introduced and interpreted as the momentum of the system; hence, the corresponding kinetic energy is expressed as

$$K(\mathbf{q}) = \frac{1}{2} \sum_{i=1}^l q_i^2.$$

The Hamiltonian function is then given by

$$H(\mathbf{x}, \mathbf{q}) = E(\mathbf{x}) + K(\mathbf{q}),$$

and it defines the distribution

$$\begin{aligned} p(\mathbf{x}, \mathbf{q}) &= \frac{1}{Z_H} \exp(-H(\mathbf{x}, \mathbf{q})) \\ &= \frac{1}{Z_E} \exp(-E) \frac{1}{Z_K} \exp(-K(\mathbf{q})) \\ &:= p(\mathbf{x})p(\mathbf{q}), \end{aligned}$$

where Z_K is the normalizing constant of the respective Gaussian term associated with the kinetic energy. The desired distribution, $p(\mathbf{x})$, is obtained as the marginal of $p(\mathbf{x}, \mathbf{q})$. Hence, if sampling from $p(\mathbf{x}, \mathbf{q})$ is possible, then discarding \mathbf{q} results in samples drawn from the desired distribution. The evolution of the variables in time is given by the associated Hamiltonian dynamics of the equivalent system.

Such methods may lead to a substantial improvement in convergence speed; the reason is that via the Hamiltonian interpretation, information hidden in the derivatives of $E(\mathbf{x})$ (i.e., $\dot{\mathbf{q}} = -\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}}$) is exploited in order for the system to detect directions toward high-probability mass.

In the *reversible jump Markov chain Monte Carlo* algorithms, the Metropolis-Hastings algorithm is extended to account for state-spaces of varying dimensionality [13]. Such methods are appropriate in cases where multiple parameter models of varying dimensionality are involved. Thus, the Markov chain is given the liberty to jump between models of different dimensionality.

Variational inference or Monte Carlo methods

In the beginning of this chapter we mentioned that the variational inference techniques, which were considered in Chapter 13, are the deterministic alternatives of the Monte Carlo methods. We will now attempt to sketch in a few lines the pros and cons of each of the two approaches. The main advantages concerning the former path to Bayesian learning are as follows:

- They are computationally more efficient for small- and medium-scale tasks.
- It is fairly easy to determine when to stop iterations and to know when convergence has been achieved.
- One can compute lower bounds for the likelihood function.

The advantages concerning Monte Carlo methods are as follows:

- They can be applied to more general cases, for example, models without computationally convenient priors, or to models whose structure is changing.
- They do not rely on approximations such as the mean-field approximation.
- They can be more efficient for large-scale tasks.

14.11 A CASE STUDY: CHANGE-POINT DETECTION

The task of change-point detection is of major importance in a number of scientific disciplines, ranging from engineering and sociology to economics and environmental studies. The accumulated literature is vast; see, for example, [1, 19, 36], and the references therein. The aim of the change-point identification task is to detect partitions in a sequence of observations, in order for the data in each block to be statistically “similar,” in other words, to be distributed according to a common probability distribution. The hidden Markov models (HMMs) and the dynamical Bayesian methods, which are discussed in [Chapter 17](#), come under this more general umbrella of problems. Our goal in this example is to demonstrate the use of Gibbs sampling in the context of the change-point detection task (see, e.g., [4]).

Let x_n be a discrete random variable that corresponds to the count of an event, for example, the number of requests for telephone calls within an interval of time, requests for individual documents on a web server, particle emissions in radioactive materials, number of accidents in a working environment, and so on. We adopt the Poisson process to model the distribution of x_n , that is,

$$P(x; \lambda) = \frac{(\lambda \tau)^x}{x!} e^{-\lambda \tau}. \quad (14.52)$$

Poisson processes have been widely used to model the number of events that take place in a time interval, τ . For our example, we have chosen $\tau = 1$. The parameter λ is known as the *intensity* of the process (see, e.g., [32]).

We assume that our observations, x_n , $n = 1, 2, \dots, N$, have been generated by two different Poisson processes, $P(x; \lambda_1)$ and $P(x; \lambda_2)$. Also, the change of the model has taken place suddenly at an unknown time instant, n_0 . Our goal is to estimate the posterior,

$$P(n_0 | \lambda_1, \lambda_2, \mathbf{x}_{1:N}).$$

Moreover, the exact values of λ_1 and λ_2 are not known. The only available information is that the Poisson process intensities, λ_i , $i = 1, 2$, are distributed according to a (prior) gamma distribution, that is,

$$p(\lambda) = \text{Gamma}(\lambda | a, b) = \frac{1}{\Gamma(a)} b^a \lambda^{a-1} \exp(-b\lambda),$$

for some known positive values a, b . We will finally assume that we have no prior information on when the time of change occurred; thus, the prior is chosen to be the uniform distribution, $P(n_0) = \frac{1}{N}$. Based on the previous assumptions, the corresponding joint distribution is given by,

$$p(n_0, \lambda_1, \lambda_2, \mathbf{x}_{1:N}) = p(\mathbf{x}_{1:N} | \lambda_1, \lambda_2, n_0) p(\lambda_1) p(\lambda_2) P(n_0)$$

or

$$p(n_0, \lambda_1, \lambda_2, \mathbf{x}_{1:N}) = \prod_{n=1}^{n_0} P(x_n | \lambda_1) \prod_{n=n_0+1}^N P(x_n | \lambda_2) p(\lambda_1) p(\lambda_2) P(n_0).$$

Taking the logarithm in order to get rid of the products, and integrating out respective variables, the following conditionals needed in Gibbs sampling are obtained ([Problem 14.15](#)):

$$p(\lambda_1|n_0, \lambda_2, \mathbf{x}_{1:N}) = \text{Gamma}(\lambda_1|a_1, b_1), \quad (14.53)$$

with

$$a_1 = a + \sum_{n=1}^{n_0} x_n, \quad b_1 = b + n_0,$$

$$p(\lambda_2|n_0, \lambda_1, \mathbf{x}_{1:N}) = \text{Gamma}(\lambda_2|a_2, b_2), \quad (14.54)$$

$$a_2 = a + \sum_{n=n_0+1}^N x_n, \quad b_2 = b + (N - n_0),$$

and

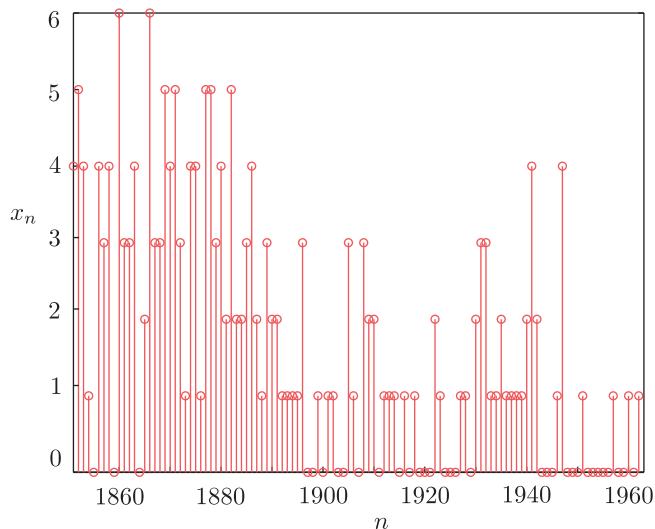
$$\begin{aligned} P(n_0|\lambda_1, \lambda_2, \mathbf{x}_{1:N}) &= \ln \lambda_1 \sum_{n=1}^{n_0} x_n - n_0 \lambda_1 + \ln \lambda_2 \sum_{n=n_0+1}^N x_n \\ &\quad - (N - n_0) \lambda_2, \quad n_0 = 1, 2, \dots, N. \end{aligned} \quad (14.55)$$

Note that the first two conditionals are gamma distributed and, as we said at the end of [Section 14.4](#), a number of different approaches are available for generating samples from it. The last distribution is a discrete one, and samples can be drawn as discussed in [Algorithm 14.1](#). We are now ready to apply Gibbs sampling.

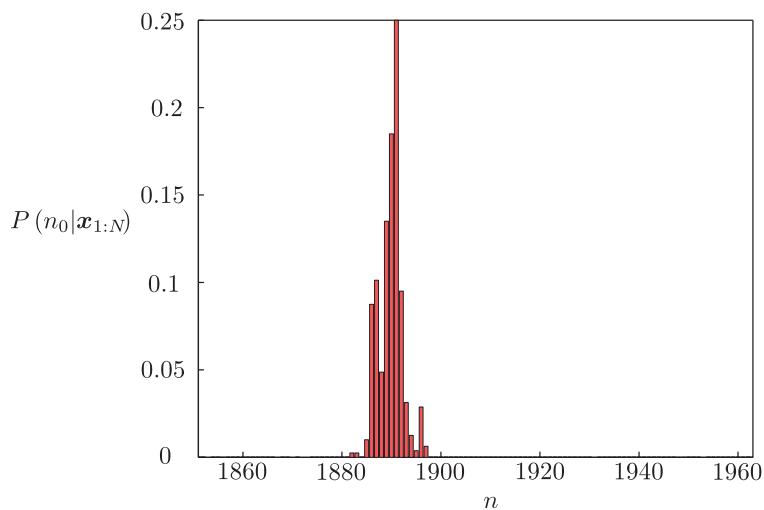
Algorithm 14.5 (Gibbs sampling for change-point detection).

- Having obtained $\mathbf{x}_{1:N} := \{x_1, \dots, x_N\}$, select a and b .
- Initialize $n_0^{(0)}$
- **For** $i = 1, 2, \dots$, **Do**
 - $\lambda_1^{(i)} \sim \text{Gamma}(\lambda|a + \sum_{n=1}^{n_0^{(i-1)}} x_n, b + n_0^{(i-1)})$
 - $\lambda_2^{(i)} \sim \text{Gamma}(\lambda|a + \sum_{n=n_0^{(i-1)}+1}^N x_n, b + (N - n_0^{(i-1)}))$
 - $n_0^{(i)} \sim P(n_0|\lambda_1^{(i)}, \lambda_2^{(i)}, \mathbf{x}_{1:N})$
- **End For**

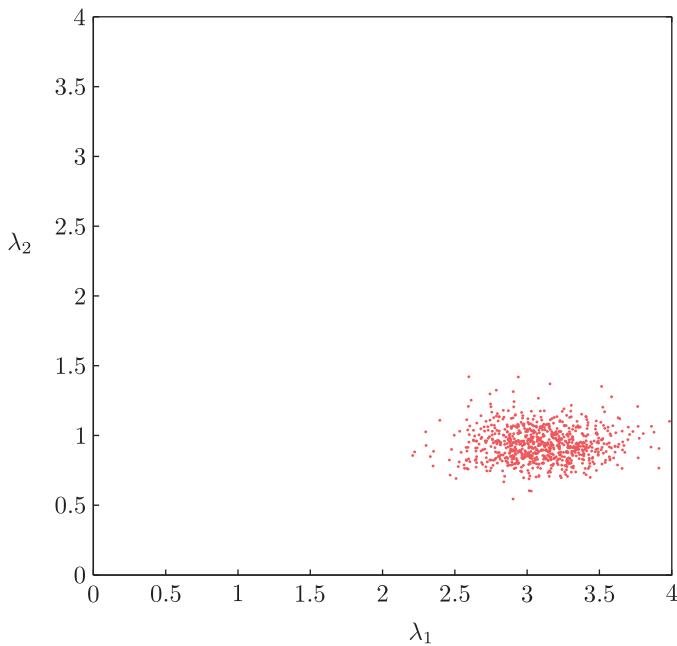
[Figure 14.12](#) shows the number of deadly accidents per year in the coal mines in England spanning the years 1851-1962. Looking at the graph, it is readily observed that the “front” part of the graph looks different from its “back” end, with a change around 1890-1900. As a matter of fact, in 1890, new health and safety regulations were introduced, following pressure from the coal miners’ unions. We will use the model explained before and draw samples according to [Algorithm 14.5](#) in order to determine the point, n_0 , where a change in the statistical distributions describing the data occurred [4]. The values of a and b were chosen equal to $a = 2$ and $b = 1$, although the obtained results are not sensitive in their choice. The burn-in phase was 200 samples. [Figure 14.13](#) shows the obtained histogram of the values of n_0 drawn by the algorithm, which clearly indicates a peak at the year 1890. [Figure 14.14](#) shows the plot of the points drawn for λ_1 and λ_2 . The plot clearly indicates that the intensity of the Poisson process dropped from $\lambda_1 = 3$ to $\lambda_2 = 1$ after the introduction of the safety regulations.

**FIGURE 14.12**

Number of deadly accidents per year in the coal mines in England over the period 1851-1962.

**FIGURE 14.13**

The histogram obtained from the values of n_0 generated by the algorithm, which approximates the posterior for n_0 , for the case study of Section 14.11. Observe that the histogram peaks at 1890, the year when the new regulations were introduced.

**FIGURE 14.14**

Case study of [Section 14.11](#): The cluster formed by the obtained values of λ_1, λ_2 .

PROBLEMS

- 14.1** Show that if $F_x(x)$ is the cumulative distribution function of a random variable x , then the random variable $u = F_x(x)$ follows the uniform distribution in $[0, 1]$.
14.2 Show that if u follows the uniform distribution and

$$x = F_x^{-1}(u) := g(u), \quad (14.56)$$

then indeed x is distributed according to $F_x(x) = \int_{-\infty}^x p(x) dx$.

- 14.3** Consider the random variables r and ϕ with exponential and uniform distributions

$$p_r(r) = \frac{1}{2} \exp\left(-\frac{r}{2}\right), \quad r \geq 0,$$

and

$$p_\phi(\phi) = \begin{cases} \frac{1}{2\pi} & 0 \leq \phi \leq 2\pi, \\ 0 & \text{otherwise,} \end{cases}$$

respectively. Show that the transformation

$$x = \sqrt{r} \cos \phi = g_x(r, \phi),$$

$$y = \sqrt{r} \sin \phi = g_y(r, \phi),$$

renders both \mathbf{x} and \mathbf{y} to follow the normalized Gaussian $\mathcal{N}(0, 1)$.

14.4 Show that if

$$p_{\mathbf{x}}(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \mathbf{0}, I),$$

then \mathbf{y} given by the transformation

$$\mathbf{y} = L\mathbf{x} + \boldsymbol{\mu}$$

is distributed according to

$$p_{\mathbf{y}}(\mathbf{y}) = \mathcal{N}(\mathbf{y} | \boldsymbol{\mu}, \Sigma),$$

where $\Sigma = LL^T$.

14.5 Consider two Gaussians

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \mathbf{0}, \sigma_p^2 I), \quad \sigma_p^2 = 0.1,$$

and

$$q(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \mathbf{0}, \sigma_q^2 I), \quad \sigma_q^2 = 0.11,$$

$\mathbf{x} \in \mathbb{R}^l$. In order to use $q(\mathbf{x})$ for drawing samples from $p(\mathbf{x})$ via the rejection sampling method, a constant c has to be computed so that

$$cq(\mathbf{x}) \geq p(\mathbf{x}).$$

Show that

$$c \geq \left(\frac{\sigma_q}{\sigma_p} \right)^l,$$

and compute the probability of accepting samples.

14.6 Show that using importance sampling leads to an unbiased estimator for the normalizing constant of the desired distribution,

$$p(\mathbf{x}) = \frac{1}{Z} \phi(\mathbf{x}).$$

However, the estimator of $\mathbb{E}[f(\mathbf{x})]$ for a function $f(\cdot)$ is a biased one.

14.7 Let $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \mathbf{0}, \sigma_1^2 I)$. Choose the proposal distribution for importance sampling as

$$q(\mathbf{x}) = \mathcal{N}(\mathbf{x} | \mathbf{0}, \sigma_2^2 I).$$

The weights are computed as

$$w(\mathbf{x}) = \frac{p(\mathbf{x})}{q(\mathbf{x})}.$$

If $w(\mathbf{0})$ is the weight at $\mathbf{x} = \mathbf{0}$, then the ratio $\frac{w(\mathbf{x})}{w(\mathbf{0})}$ is given by

$$\frac{w(\mathbf{x})}{w(\mathbf{0})} = \exp \frac{1}{2} \left(\frac{\sigma_1^2 - \sigma_2^2}{\sigma_1^2 \sigma_2^2} \sum_{i=1}^l x_i^2 \right).$$

Observe that even for a very good match between $q(\mathbf{x})$ and $p(\mathbf{x})$ ($\sigma_1^2 \simeq \sigma_2^2$), for large values of l , the values of the weights can change significantly due to the exponential dependence.

- 14.8 Show that a stochastic matrix P always has the value $\lambda = 1$ as its eigenvalue.
- 14.9 Show that if the eigenvalue of a transition matrix is not equal to one, its magnitude cannot be larger than one, that is, $|\lambda| \leq 1$.
- 14.10 Prove that if P is a stochastic matrix and $\lambda \neq 1$, then the elements of the corresponding eigenvector add to zero.
- 14.11 Prove the square root dependence of the distance traveled by a random walk, with infinite many integer states, on the time, n .
- 14.12 Prove, using the detailed balance condition, that the invariant distribution associated with the Markov chain implied by the Metropolis-Hastings algorithm is the desired distribution, $p(\mathbf{x})$.
- 14.13 Show that in Gibbs sampling, the desired joint distribution is invariant with respect to each one of the base transition pdfs.
- 14.14 Show that the acceptance rate for the Gibbs sampling is equal to one.
- 14.15 Derive the formulae for the conditional distributions of [Section 14.11](#).

MATLAB Exercise

- 14.16 Develop a MATLAB code for the Gibbs sampler. Then, use it to draw samples from the two-dimensional Gaussian distribution, with mean value and covariance matrix equal to

$$\boldsymbol{\mu} = [0, 0]^T, \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}.$$

Derive the conditional pdf of each one of the variables with respect to the other (use Appendix in [Section 12.9](#)). Then use the conditional pdfs to implement the Gibbs sampler. Plot the generated points in the two-dimensional space after 20, 50, 100, 300, and 1000 iterations. What do you observe concerning convergence?

REFERENCES

- [1] D. Barry, J.A. Hartigan, A Bayesian analysis for change point problems, *J. Am. Stat. Assoc.*, 88 (1993) 309-319.
- [2] N. Bhatnagar, A. Bogdanov, E. Mossel, The Computational Complexity of Estimating Convergence Time, 2010, arXiv:1007.0089v1 [cs.DS].
- [3] G. Celeux, J. Diebolt, The SEM algorithm: a probabilistic teacher derive from the EM algorithm for the mixture problem, *Comput. Stat. Q.* 2 (1985) 73-82.
- [4] A.T. Cemgil, A tutorial introduction to Monte Carlo methods, Markov chain Monte Carlo and particle filtering, in: R. Chellappa, S. Theodoridis (Eds.), Academic Press Library in Signal Processing, Vol. 1, Academic Press, San Diego, CA, (2014) 1065-1113.
- [5] M.H. Chen, Q.M. Shao, J.G. Ibrahim (Eds.), *Monte Carlo Methods in Bayesian Computation*, Springer, New York, 2001.
- [6] M.K. Cowles, B.P. Carlin, Markov chain Monte Carlo convergence diagnostics: a comparative review, *J. Am. Stat. Assoc.* 91 (1996) 883-904.
- [7] L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, New York, 1986.

- [8] A. Gelman, D.B. Rubin, Inference from iterative simulation using multiple sequences, *Stat. Sci.* 7 (1992) 457-511.
- [9] A.E. Gelfand, A.F.M. Smith, Sampling based approaches to calculating marginal densities, *J. Am. Stat. Assoc.* 85 (1990) 398-409.
- [10] S. Geman, D. Geman, Stochastic relaxation Gibbs distributions and the Bayesian restoration of images, *IEEE Trans. Pattern Anal. Mach. Intell.* 6 (1984) 721-741.
- [11] W.R. Gilks, P. Wild, Adaptive rejection sampling for Gibbs sampling, *Appl. Stat.* 41 (1992) 337-348.
- [12] W.R. Gilks, G.O. Roberts, E.I. George, Adaptive direction sampling, *Statistician*, 43 (1994) 179-189.
- [13] P.J. Green, Reversible jump Markov chain Monte Carlo computation and Bayesian model determination, *Biometrika* 82 (1995) 711-732.
- [14] W.K. Hastings, Monte Carlo sampling methods using Markov chains and their applications, *Biometrika* 57 (1970) 97-109.
- [15] D.M. Higdon, Auxiliary variable methods for Markov chain Monte Carlo with Applications, *J. Am. Stat. Assoc.* 93 (1994) 179-189.
- [16] C.A. Jensen, A. Kong, U. Kjaeruff, Blocking Gibbs sampling in very large probabilistic expert systems, *Int. J. Hum. Comput. Stud.* 42 (1995) 647-666.
- [17] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, *Science* 220 (1983) 671-680.
- [18] D.E. Knuth, *The Art of Computer Programming*, second ed., Addison Wesley, Reading, MA, 1981.
- [19] T.L. Lai, Sequential change point detection in quality control and dynamical systems, *J. R. Stat. Soc. B*, 57 (1995) 613-658.
- [20] D.H. Lehmer, Mathematical methods in large scale computing units, *Ann. Comput. Lab. Harvard Univ.* 26 (1951).
- [21] D.A. Levin, Y. Peres, E.L. Wilmer, *Markov Chains and Mixing Times*, American Mathematical Society, Providence, RI, 2008.
- [22] F. Liang, C. Liu, R.J. Carroll, *Advanced Markov Chain Monte Carlo Methods: Learning from Past Samples*, John Wiley, New York, 2010.
- [23] J.S. Liu, The collapsed Gibbs sampler in Bayesian computations with applications to a gene regulation problem, *J. Am. Stat. Assoc.* 89 (427) (1994) 958-966.
- [24] J.S. Liu, *Monte Carlo Strategies in Scientific Computing*, Springer, New York, 2001.
- [25] E. Marinari, G. Parisi, Simulated tempering: a new Monte Carlo scheme, *Europhys. Lett.* 19 (6) (1992) 451-458.
- [26] D.J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, Cambridge, 2003.
- [27] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, *J. Chem. Phys.* 21 (1953) 1087-1091.
- [28] N. Metropolis, The beginning of Monte Carlo methods, *Los Alamos Sci.* (1987) 125-130.
- [29] J. Moller, A.N. Pettitt, R. Reeves, K.K. Berthelsen, An efficient Markov chain Monte Carlo method for distributions with intractable normalising constants, *Biometrika* 93 (2006) 451-668.
- [30] R.M. Neal, Probabilistic inference using Markov chain Monte Carlo methods, Technical Report (GR-TR-93-1), Department of Computer Science, University of Toronto, Canada, 1993.
- [31] R.M. Neal, Slice sampling, *Ann. Stat.* 31 (2003) 705-767.
- [32] A. Papoulis, S.U. Pillai, *Probability, Random Variables and Stochastic Processes*, fourth ed., McGraw-Hill, New York, 2002.
- [33] P.H. Peskun, Optimum Monte Carlo sampling using Markov chains, *Biometrika*, 60 (1973) 607-612.
- [34] S.K. Park, K.W. Miller, Random number generations: Good ones are hard to find, *Commun. ACM* 31 (10) (1988) 1192-1201.

- [35] M. Plummer, N. Best, K. Cowles, CODA: output analysis and diagnostics for Markov chain Monte Carlo simulations, 2006, <http://cran.r-project.org>.
- [36] J. Reeves, J. Chen, X.L. Wang, R. Lund, Q.Q. Lu, A review and comparison of changepoint detection techniques for climate data, *J. Appl. Meteorol. Climatol.* 46 (2007) 900-915.
- [37] B. Ripley, *Stochastic Simulation*, John Wiley, New York, 1987.
- [38] C.P. Robert, G. Casella, *Monte Carlo Statistical Methods*, second ed., Springer, New York, 2004.
- [39] A. Sinclair, *Algorithms for Random Generation and Counting: A Markov chain Approach*, Birkhäuser, Boston, 1993.
- [40] L. Tierney, Markov chains for exploring posterior distribution, *Ann. Stat.* 22 (1994) 1701-1762.

PROBABILISTIC GRAPHICAL MODELS: PART I

15

CHAPTER OUTLINE

15.1 Introduction	755
15.2 The Need for Graphical Models	756
15.3 Bayesian Networks and the Markov Condition	758
15.3.1 Graphs: Basic Definitions	759
15.3.2 Some Hints on Causality	763
15.3.3 <i>D</i> -Separation	765
15.3.4 Sigmoidal Bayesian Networks	768
15.3.5 Linear Gaussian Models	769
15.3.6 Multiple-Cause Networks	770
15.3.7 I-Maps, Soundness, Faithfulness, and Completeness	771
15.4 Undirected Graphical Models	772
15.4.1 Independencies and I-Maps in Markov Random Fields	773
15.4.2 The Ising Model and Its Variants	775
15.4.3 Conditional Random Fields (CRFs)	777
15.5 Factor Graphs	778
15.5.1 Graphical Models for Error-Correcting Codes	780
15.6 Moralization of Directed Graphs	782
15.7 Exact Inference Methods: Message-Passing Algorithms	783
15.7.1 Exact Inference in Chains	783
15.7.2 Exact Inference in Trees	787
15.7.3 The Sum-Product Algorithm	788
15.7.4 The Max-Product and Max-Sum Algorithms	792
Problems	799
References	801

15.1 INTRODUCTION

In [Figure 13.2](#), we used a graphical description to indicate conditional dependencies among various parameters that control the “fusion” of the prior and conditional pdfs in a hierarchical manner. Our purpose there was more of a pedagogical nature; we could live without it. In this chapter, graphical

models emerge out of necessity. In many everyday machine learning applications involving multivariate statistical modeling, even simple inference tasks can easily become computationally intractable. Typical applications involve bioinformatics, speech recognition, machine vision, and text mining, to name but a few.

Graph theory has proved a powerful and elegant tool that has extensively been used in optimization and computational theory. A graph encodes dependencies among interacting variables and can be used to formalize the probabilistic structure that underlies our modeling assumptions. This can then be used to facilitate computation in a number of inference tasks, such as the calculation of marginals, modes, and conditional probabilities. Moreover, graphical models can be used as a vehicle to impose approximations onto the models when computational needs go beyond the available resources.

Early celebrated examples of the use of such models in learning tasks are the hidden Markov models, Kalman filtering, and error correcting coding, which have been popular since the early sixties.

This is the first of two chapters dedicated to probabilistic graphical models. This chapter focuses on the basic definitions and concepts, and most of its material is a must for a first reading on the topic. A number of basic graphical models are discussed, such as Bayesian networks (BNs) and Markov random fields. Exact inference is presented, and the elegant message-passing algorithm for inference on chains and trees is introduced.

15.2 THE NEED FOR GRAPHICAL MODELS

Let us consider a simplified example of a learning system in the context of a medical application. Such a system comprises a set of m diseases that correspond to hidden variables and a set of n symptoms (findings). The diseases are treated as random variables, d_1, d_2, \dots, d_m , and each of them can be absent or present and thus can be encoded by a zero or one, that is, $d_j \in \{0, 1\}$, $j = 1, 2, \dots, m$. The same applies to the symptoms, f_i , which can either be absent or present; hence, $f_i \in \{0, 1\}$, $i = 1, 2, \dots, n$. The symptoms comprise the observed variables.¹

The goal of the system is to predict a disease hypothesis, that is, the presence of a number of diseases, given the presence of a set of symptoms, which have been observed. During the training, which is based on experts' assessments, the system learns the prior probabilities $P(d_j)$, and the conditional probabilities $P_{ij} = P(f_i = 1 | d_j = 1)$, $i = 1, 2, \dots, n$, $j = 1, 2, \dots, m$. The latter comprise a table of nm entries. For a realistic system, these numbers can be very large. For example, in [41], m is of the order of 500-600 and n of the order of 4000. Let f be the vector that corresponds to a specific set of observations for the findings, indicating the presence or absence of the respective symptoms. Assuming that symptoms are *conditionally independent*, given any disease hypothesis, \mathbf{d} , then we can write

$$P(f|\mathbf{d}) = \prod_{i=1}^n P(f_i|\mathbf{d}). \quad (15.1)$$

Ideally, one should be able to obtain the conditional probabilities $P(f_i|\mathbf{d})$ for each disease hypothesis. However, for all possible 2^m combinations of \mathbf{d} , this should require a huge amount of training data, which is impossible to collect for any practical system. This is bypassed by adopting the following model,

¹ In a more realistic system, some of the findings may not be available, that is, they may be unobservable.

$$P(f_i = 0 | \mathbf{d}) = \prod_{j=1}^m (1 - P_{ij})^{d_j}, \quad (15.2)$$

where the exponent is set to zero, $d_j = 0$, when the disease is not related to the symptom. This is known as the *noisy-OR* model. That is, it is assumed that for a negative finding the individual causes are independent [37]. Obviously,

$$P(f_i = 1 | \mathbf{d}) = 1 - P(f_i = 0 | \mathbf{d}).$$

Let us now assume that we observe a set of findings, \mathbf{f} , and we want to infer $P(d_j | \mathbf{f})$ for some j . Then,

$$\begin{aligned} P(d_j = 1 | \mathbf{f}) &= \frac{P(\mathbf{f} | d_j = 1) P(d_j = 1)}{P(\mathbf{f})} \\ &= \frac{\sum_{\mathbf{d}: d_j = 1} P(\mathbf{f} | \mathbf{d}) P(\mathbf{d})}{\sum_{\mathbf{d}} P(\mathbf{f} | \mathbf{d}) P(\mathbf{d})}. \end{aligned} \quad (15.3)$$

The summation in the denominator involves 2^m terms. For $m \sim 500$, this is a formidable task that simply cannot be carried out in a realistic time.

The previous example indicates that once one gets involved with complex systems, even innocent-looking tasks turn out to be computationally intractable. Thus, one has either to be more clever in exploiting possible independencies in the data, which can reduce the required number of computations, or make certain assumptions/approximations. In this chapter, we will study both alternatives.

Before we proceed further, it is interesting to point out another source of computational obstacles besides the calculation of Eq. (15.3). In practice, it may be more convenient to perform addition instead of implementing multiplication; multiplying a large number of variables of small values such as probabilities may cause arithmetic accuracy problems. One way to bypass products is either via logarithmic or exponential operations, which transform products into summations. For example, Eq. (15.2) can be rewritten as

$$P(f_i = 0 | \mathbf{d}) = \exp \left(- \sum_{j=1}^m \theta_{ij} d_j \right), \quad (15.4)$$

where $\theta_{ij} := -\ln(1 - P_{ij})$ and

$$P(f_i = 1 | \mathbf{d}) = 1 - \exp \left(- \sum_{j=1}^m \theta_{ij} d_j \right). \quad (15.5)$$

Observe that the presence in Eq. (15.1) of terms corresponding to negative findings contributes linearly to the complexity (product of exponentials correspond to summations). However, this is not the case with the terms associated with positive findings. Take, for example, the extreme case where all findings are negative. Then,

$$\begin{aligned} P(\mathbf{f} = \mathbf{0} | \mathbf{d}) &= \prod_{i=1}^n \exp \left(- \sum_{j=1}^m \theta_{ij} d_j \right) \\ &= \exp \left(- \sum_{i=1}^n \left(\sum_{j=1}^m \theta_{ij} d_j \right) \right). \end{aligned} \quad (15.6)$$

Consider now $f_1 = 1$ and the rest to be $f_i = 0$, $i = 2, \dots, n$. Then,

$$P(f|\mathbf{d}) = \left(1 - \exp\left(-\sum_{j=1}^m \theta_{1j} d_j\right)\right) \exp\left(-\sum_{i=2}^n \left(\sum_{j=1}^m \theta_{ij} d_j\right)\right), \quad (15.7)$$

where now the number of exponents to be computed is two. It can easily be shown that the cross-product terms lead to an exponential computational growth [20] ([Problem 15.1](#)).

The path we will follow in order to derive efficient exact inference algorithms as well as to derive efficient approximation rules, when exact inference is not possible, will be via the use of graphical models.

15.3 BAYESIAN NETWORKS AND THE MARKOV CONDITION

Before we move on to definitions, let us first see how the existence of some structure in a joint distribution can simplify the task of marginalization. We will demonstrate it using discrete probabilities, where the use of counting can make things simpler.

Let us consider l discrete jointly distributed random variables. Applying the product rule of probability, we obtain

$$P(x_1, x_2, \dots, x_l) = P(x_l|x_{l-1}, x_{l-2}, \dots, x_1) P(x_{l-1}|x_{l-2}, \dots, x_1) \dots P(x_1). \quad (15.8)$$

Assume that each one of these variables takes values in the discrete set $\{1, 2, \dots, k\}$. In the general case, if we want to marginalize with respect to one of the variables, say x_1 , we must sum over the others, that is,

$$P(x_1) = \sum_{x_2} \dots \sum_{x_l} P(x_1, x_2, \dots, x_l),$$

where each one of the summations is over k possible values, which is equivalent to $\mathcal{O}(k^l)$ summations; for large values of k and/or l , this is a formidable and sometimes impossible task. Let us consider now one extreme case, where all the involved variables are mutually independent. Then the product rule becomes

$$P(x_1, x_2, \dots, x_l) = \prod_{i=1}^l P(x_i),$$

and marginalization turns out to be the trivial identity

$$P(x_1) = \left(\sum_{x_l} P(x_l) \sum_{x_{l-1}} P(x_{l-1}) \dots \sum_{x_2} P(x_2) \right) P(x_1), \quad (15.9)$$

because each summation is carried out independently, and of course results to one. In other words, exploiting the product rule and the statistical independence can bypass the obstacle of the exponential growth of the computational load. As a matter of fact, the previous full-independence assumptions give birth to the naive Bayes classifier, [Chapter 7](#).

In this chapter, we are going to study cases that lie between the previous two extremes. The general idea is to be able to express the joint probability distribution (probability density/mass function) in terms of *products* of factors, where each one of them depends on a *subset* of the involved variables. This can be expressed by writing the joint distribution as

$$p(x_1, x_2, \dots, x_l) = \prod_{i=1}^l p(x_i | \text{Pa}_i), \quad (15.10)$$

where Pa_i denotes the subset of variables associated with the random variable x_i . Take the following example,

$$p(x_1, x_2, x_3, x_4, x_5, x_6) = p(x_6 | x_4) p(x_5 | x_3, x_4) p(x_4 | x_1, x_2) p(x_3 | x_1) p(x_2) p(x_1). \quad (15.11)$$

Then $\text{Pa}_6 = \{x_4\}$, $\text{Pa}_5 = \{x_3, x_4\}$, $\text{Pa}_4 = \{x_1, x_2\}$, $\text{Pa}_3 = \{x_1\}$, $\text{Pa}_2 = \emptyset$, $\text{Pa}_1 = \emptyset$. The variables in the set, Pa_i , are defined as the *parents* of the respective, x_i , and from a statistical point of view this means that x_i is statistically independent of *all* the variables *given* the values of its parents. Every $p(x_i | \text{Pa}_i)$ expresses a *conditional independence* relationship and it imposes a *probabilistic structure* that underlies our multivariate set. It is such types of independencies that we will exploit in order to perform inference tasks at a lower computational cost.

15.3.1 GRAPHS: BASIC DEFINITIONS

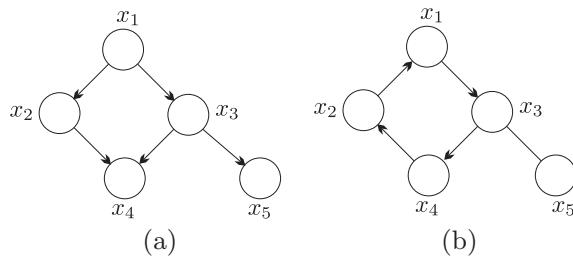
A graph $G = \{V, E\}$ is a collection of nodes/vertices $V = \{x_1, \dots, x_l\}$ and a collection of edges (arcs) $E \subset V \times V$. Each edge connects two vertices and it is denoted as a pair, $(x_i, x_j) \in E$. An edge can be either *directed*; we write $(x_i \rightarrow x_j)$ to indicate the direction or *undirected*, and in this case we simply write (x_i, x_j) . Suppose we have a set of nodes x_1, x_2, \dots, x_k , $k \geq 2$ and a corresponding set of edges $(x_{i-1}, x_i) \in E$ or $(x_{i-1} \rightarrow x_i) \in E$, $2 \leq i \leq k$; that is, the edges connect pairs of nodes in *sequence* and they can be either directed or not. This sequence of edges is called a *path* from x_1 to x_k . If there is at least one directed edge, the path is called *directed*. A *cycle* is a path from a node to itself. A *chain* or a *trail* is a path that can be “run” either from x_1 to x_k or from x_k to x_1 ; that is, all directed edges are replaced by undirected ones.

A *directed* graph comprises directed edges only and it is called a *directed acyclic graph* (DAG) if it contains *no cycles*. Given a DAG, a node in it, x_i , is called *parent* of x_j if there is a (directed) edge from x_i to x_j , and we call x_j the *child* of x_i . A node x_j is called a *descendant* of x_i and x_i an *ancestor* of x_j if there is a path from x_i to x_j . A node x_j is called *nondescendant* of x_i if it is not a descendant of x_i . A graph is said to be *fully connected* or *complete* if there is an edge between every pair of nodes.

Figure 15.1 illustrates the previous definitions.

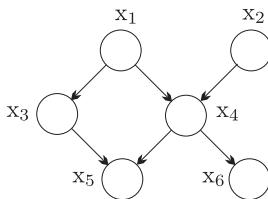
Definition 15.1. A *Bayesian network structure* is a DAG whose nodes represent random variables, x_1, \dots, x_l , and every variable (node), x_i , is *conditionally independent* of the set of *all* its nondescendants, given the set of all its *parents*. Sometimes this is also known as the *Markov condition*.

If we denote the set of the nondescendants of a node x_i as ND_i , the Markov condition can be written as [12] $x_i \perp ND_i | \text{Pa}_i, \forall i = 1, 2, \dots, l$. Sometimes, the conditional independencies are also known as *local independencies*. Stated differently, a BN graphical structure is a convenient way to encode conditional independencies. **Figure 15.2** shows the DAG that expresses the conditional independencies

**FIGURE 15.1**

(a) This is a DAG because there are no cycles. x_1 is a parent of both x_2 and x_3 . x_4 and x_5 are children of x_3 . x_1 , x_2 , and x_3 are ancestors of x_4 , while x_4 and x_5 are descendants of x_1 . x_5 is a nondescendant of x_2 and x_4 .

(b) This is not a DAG and the sequence $(x_2, x_1, x_3, x_4, x_2)$ comprises a cycle. The edge (x_3, x_5) is undirected. The sequence of nodes (x_1, x_3, x_5) forms a directed path, and the sequence (x_1, x_2, x_4, x_3) forms a chain, once directed edges are replaced by undirected ones.

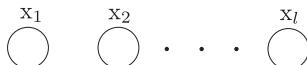
**FIGURE 15.2**

A Bayesian network corresponding to the pdf in Eq. (15.11). Observe that x_5 is conditionally independent of x_1 and x_2 given the values of x_3 and x_4 .

used in Eq. (15.11), in order to express the joint distribution as a product of factors. Conditional independence among random variables, that is, $x \perp\!\!\!\perp y|z$ or, equivalently, $p(x|y, z) = p(x|z)$, means that once we know the value of z , observing the value of y gives no additional information about x (note that the previous makes sense only if $p(y, z) > 0$). For example, the probability of children having a good education varies whether they grow up in a poor or a rich (low and high Gross National Product (GNP)) country. The probability of someone getting a high-paying job depends on her/his level of education. The probability of someone getting a high-paying job is independent of the country in which he or she was born and raised, given the level of her/his education.

Theorem 15.1. *Let G be a Bayesian network structure and p be the joint probability distribution of the random variables associated with the graph. Then p is equal to the product of the conditional distributions of all the nodes given the values of their parents, and we say that p factorizes over G .*

The proof of the theorem is done by induction, [Problem 15.2](#). Moreover, the reverse of this theorem is also true. The previous theorem assumed a distribution and built the BN based on the underlying conditional independencies. The next theorem deals with the reverse procedure. One builds a graph based on a set of conditional distributions—one for each node of the network.

**FIGURE 15.3**

BN for independent variables. No edges are present because every variable is independent of all the others and no parents can be identified.

Theorem 15.2. *Let G be a DAG and associate a conditional probability for each node, given the values of its parents. Then the product of these conditional probabilities yields a joint probability of the variables. Moreover, the Markov condition is satisfied.*

The proof of this theorem is given in [Problem 15.4](#). Note that in this theorem, we used the term probability and not distribution. The reason is that the theorem is not true for every form of conditional densities (pdfs) [14]. However, it holds true for a number of widely used pdfs, such as the Gaussians. This theorem is very useful because, often in practice, this is the way we construct a probabilistic graphical model—building it hierarchically, using reasoning on the corresponding physical process that we want to model, and encoding conditional independencies in the graph.

[Figure 15.3](#) shows the BN structure describing a set of mutually independent variables (naive Bayes assumption).

Definition 15.2. A Bayesian network (BN) is a pair (G, p) , where the distribution, p , factorizes over the DAG, G , in terms of a set of conditional probability distributions, associated with the nodes of G .

In other words, a Bayesian network is associated with a specific distribution. In contrast, a Bayesian network structure refers to any distribution that satisfies the Markov condition as expressed by the network structure.

Example 15.1. Consider the following simplified study relating the GNP of a country to the level of education and the type of a job an adult gets later in her/his professional life. Variable x_1 is binary with two values HGP and LGP, corresponding to countries with high and low GNP, respectively. Variable x_2 gets three values, NE, LE, and HE, corresponding to no education, low-level, and high-level education, respectively. Finally, variable x_3 gets also three possible values, UN, LP, HP corresponding to unemployed, low-paying and high-paying jobs, respectively. Using a large enough sample of data, the following probabilities are learned:

1. Marginal Probabilities:

$$P(x_1 = LGP) = 0.8, \quad P(x_1 = HGP) = 0.2.$$

2. Conditional Probabilities:

$$P(x_2 = NE|x_1 = LGP) = 0.1, \quad P(x_2 = LE|x_1 = LGP) = 0.7,$$

$$P(x_2 = HE|x_1 = LGP) = 0.2,$$

$$P(x_2 = NE|x_1 = HGP) = 0.05, \quad P(x_2 = LE|x_1 = HGP) = 0.2,$$

$$P(x_2 = HE|x_1 = HGP) = 0.75.$$

$$P(x_3 = UN|x_2 = NE) = 0.15, \quad P(x_3 = LP|x_2 = NE) = 0.8,$$

$$P(x_3 = HP|x_2 = NE) = 0.05.$$

$$P(x_3 = UN|x_2 = LE) = 0.10, P(x_3 = LP|x_2 = LE) = 0.85,$$

$$P(x_3 = HP|x_2 = LE) = 0.05.$$

$$P(x_3 = UN|x_2 = HE) = 0.05, P(x_3 = LP|x_2 = HE) = 0.15,$$

$$P(x_3 = HP|x_2 = HE) = 0.8.$$

Note that although these values are not the result of a specific experiment, they are in line with the general trend provided by more professional studies, which involve many more random variables. However, for pedagogical reasons we must keep the example simple.

The first observation is that even for this simplistic example involving only three variables, one has to obtain seventeen probability values. This verifies the high computational load that may be required with such tasks.

[Figure 15.4](#) shows the BN that captures the previously stated conditional probabilities. Note that the Markov condition renders x_3 independent of x_1 , given the value of x_2 . Indeed, the job that one finds is independent of the GNP of the country, given her/his education level. We will verify that by playing with the laws of probability for the previously defined values.

According to [Theorem 15.2](#), the joint probability of an event is given by the product

$$P(x_1, x_2, x_3) = P(x_3|x_2)P(x_2|x_1)P(x_1). \quad (15.12)$$

In other words, the probability of someone coming from a rich country, having a good education, and getting a high-paying job will be equal to $(0.8)(0.75)(0.2) = 0.12$; similarly, the probability of somebody coming from a poor country with low-level education to get a low-paying job is 0.476.

As a next step, we will verify the Markov condition, implied by the Bayesian network structure, using the probability values given before. That is, we will verify that using conditional probabilities to build the network, these probabilities basically encode *conditional independencies*, as [Theorem 15.2](#) suggests. Let us consider,

$$\begin{aligned} P(x_3 = HP|x_2 = HE, x_1 = HGP) &= \frac{P(x_3 = HP, x_2 = HE, x_1 = HGP)}{P(x_2 = HE, x_1 = HGP)} \\ &= \frac{0.12}{P(x_2 = HE, x_1 = HGP)}. \end{aligned}$$

Also,

$$\begin{aligned} P(x_2 = HE, x_1 = HGP) &= P(x_2 = HE|x_1 = HGP)P(x_1 = HGP) \\ &= 0.75 \times 0.2 = 0.15, \end{aligned}$$

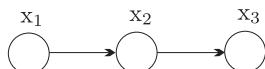


FIGURE 15.4

BN for [Example 15.1](#). Note that $x_3 \perp x_1 | x_2$.

which finally results to

$$\begin{aligned} P(x_3 = HP|x_2 = HE, x_1 = HGP) &= 0.8 \\ &= P(x_3 = HP|x_2 = HE), \end{aligned}$$

which verifies the claim. The reader can check that this is true for all possible combinations of values.

15.3.2 SOME HINTS ON CAUSALITY

The existence of directed links in a Bayesian network *does not* necessarily reflect a cause-effect relationship from a parent to a child node.² It is a well-known fact in statistics that correlation between two variables does not always establish a causal relationship between them. For example, their correlation may be due to the fact that they both relate to a latent (unknown) variable. A typical example is the discussion related to whether smoking causes cancer or they are both due to an unobserved genotype that causes cancer and at the same time a craving for nicotine; this has been the defense line of the tobacco companies.

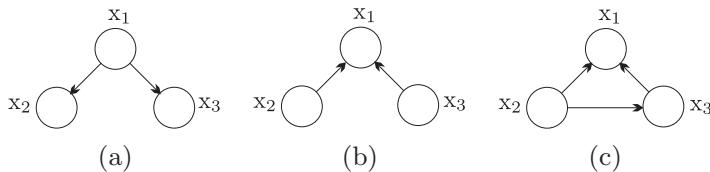
Let us return to [Example 15.1](#). Although GNP and quality of education are correlated, one cannot say that GNP is a cause of the educational system. No doubt there is a multiplicity of reasons, such as the political system, the social structure, the economic system, historical reasons, and tradition, all of which need to be taken into consideration. As a matter of fact, the structure of the graph relating the three variables in the example could be reversed. We could collect data in the other way around; obtain the probabilities $P(x_3 = UN)$, $P(x_3 = LP)$, $P(x_3 = HP)$, and then the conditional probabilities $P(x_2|x_3)$ (e.g., $P(x_2 = HE|x_3 = UN)$) and finally $P(x_1|x_2)$ (e.g., $P(x_1 = HGP|x_2 = HE)$). In principle, such data can also be collected from a sample of people. In such a case, the resulting Bayesian network would comprise again three nodes as in [Figure 15.4](#), but with the direction of the arrows reversed. This is also reasonable because the probability of someone coming from a rich or a poor country is independent of her/his job, given the level of education. Moreover, both models should result in the same joint-probability distribution for any joint event. Thus, if the direction of the arrows were to indicate causality, then this time, it would be that the educational system has a cause-effect relationship on the GNP. This, for the same reasons stated before, cannot be justified. Having said all that, it does not necessarily mean that cause-effect relationships are either absent in a BN or it is not important to know them. On the contrary, in many cases, there is good reason to strive to unveil the underlying cause-effect relationships while building a BN.

Let us elaborate a bit more on this and see why exploiting any underlying cause-effect relationships can be to our benefit. Take, for example, the BN in [Figure 15.5](#) relating the presence or absence of a disease with the findings from two medical tests. Let x_1 indicate the presence or absence of a disease and x_2, x_3 the discrete outcomes that can result from the two tests, respectively.

The BN in [Figure 15.5a](#) complies with our common sense reasoning that x_1 (disease) causes x_2 and x_3 (tests). However, this is not possible to deduct by *simply* looking at the available probabilities. This is because the probability laws are *symmetric*. Even if x_1 is the cause, we can still compute $P(x_1|x_2)$ once $P(x_1, x_2, x_3)$ and $P(x_2)$ are available, that is,

$$P(x_1|x_2) = \frac{P(x_1, x_2)}{P(x_2)} = \frac{\sum_{x_3} P(x_1, x_2, x_3)}{P(x_2)}.$$

² This topic will not be pursued any further; its purpose is to make the reader aware of the issue. It can be bypassed in a first reading.

**FIGURE 15.5**

Three possible graphs relating a disease x_1 , to the results of two tests, x_2, x_3 . (a) The dependencies in this graph comply with common sense. (b) This graph renders x_2, x_3 statistically independent, which is not reasonable. (c) Training this graph needs an extra probability value compared to that in (a).

Previously, in order to say that x_1 causes x_2 and x_3 , we used some *extra* information/knowledge, which we called common sense reasoning. Note that in this case, training requires the knowledge of the values of three probabilities, namely, $P(x_1)$, $P(x_2|x_1)$, $P(x_3|x_1)$. Let us now assume that we choose the graph model in Figure 15.5b. This time, ignoring the cause-effect relationship has resulted in the wrong model. This model renders x_2 and x_3 independent, which, obviously, cannot be the case. These should only be *conditionally* independent given x_1 . The only sensible way to keep x_2 and x_3 as parents of x_1 is to add an extra link, as shown in Figure 15.5c, which establishes a relation between the two. However, to train such a network, besides the values of the three probabilities, $P(x_2)$, $P(x_3)$, and $P(x_1|x_2, x_3)$, one needs to know the values for an extra one, $P(x_3|x_2)$. Thus, when building a BN, it is always good to know any underlying cause-effect directions. Moreover, there are other reasons, too. For example, this may be related to the *interventions*, which are actions that change the state of a variable in order to study the respective impact on other variables; because a change propagates in the causal direction, such a study is only possible if the network has been structured in a cause-effect hierarchy. For example, in biology, there is a strong interest in understanding which genes affect activation levels of other genes, and in predicting the effects of turning certain genes on or off.

The notion of causality is not an easy one, and philosophers have been arguing about it for centuries. Although our intention here is by no means to touch this issue, it is interesting to quote two well-known philosophers.

According to David Hume, causality is not a property of the real world but a concept of the mind that helps us explain our perception of the world. Hume (1711–1776) was a Scottish philosopher best known for his philosophical empiricism and skepticism. His most well-known work is the “Treatise of Human Nature,” and in contrast to the rationalistic philosophy school, he advocated that human nature is mainly governed by desire and not reason.

According to Bertrand Russell, the law of causality has nothing to do with the laws of physics, which are symmetrical (recall our statement before concerning conditional probabilities) and indicate no cause-effect relationship. For example, Newton’s gravity law can be expressed in any of the following forms,

$$B = mg \text{ or } g = \frac{B}{m} \text{ or } m = \frac{B}{g},$$

and looking only at them, no cause-effect relationship can be deduced. Bertrand Russell (1872–1970) was a British philosopher, mathematician, and logician. He is considered one of the founders of analytic

philosophy. In *Principia Mathematica*, co-authored with A.N. Whitehead, they made an attempt to ground mathematics on mathematical logic. He was also an antiwar activist and a liberal.

The previously stated provocative arguments have been inspired by Judea Pearl's book [38], and we provided them in order to persuade the reader to read this book; he or she can only become wiser. Pearl has made a number of significant contributions to the field and was the recipient of the Turing award in 2011.

Although one cannot deduce causality by looking only at the laws of physics or probabilities, ways of identifying it have been developed. One way is to carry out *controlled* experiments; one can change the values of the variable and study the effect of the change on another. However, this has to be done in a controlled way in order to guarantee that the caused effects are not due to other related factors.

Besides experimentation, there has been a major effort to discover causal relationships from nonexperimental evidence. In modern applications, such as microarray measurements for gene expressions or fMRI brain imaging, the number of the involved variables can easily reach the order of a few thousand. Performing experiments for such tasks is out of the question. In [38], the notion of causality is related to that of the minimality in the structure of the obtained possible DAGs. Such a view ties causality with Occam's razor. More recently, inferring causality was attempted by comparing the conditional distributions of variables given their direct causes, for all hypothetical causal directions, and choosing the most plausible. The method builds upon some smoothness arguments that underlie the conditional distributions of the effect given the causes, compared to the marginal distributions of the effect/cause [43]. In [24], an interesting alternative for inferring causality is built upon arguments from Kolmogorov's complexity theory; causality is verified by comparing shortest description lengths of strings associated with the involved distributions. For further information, the interested reader may consult, for example, [42] and the references therein.

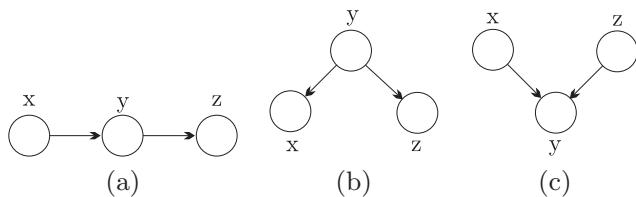
15.3.3 D-SEPARATION

Dependencies and independencies among a set of random variables play a key role in understanding their statistical behavior. Moreover, as we have already commented, they can be exploited to substantially reduce the computational load for solving inference tasks.

By the definition and the properties of a Bayesian network structure, G , we know that certain independencies hold and are readily observed via the parent-child links. The question that now is raised is whether there are additional independencies that the structure of the graph imposes on any joint probability distribution that factorizes over G . Unveiling extra independencies offers the designer more freedom to deal with computational complexity issues more aggressively.

We will attack the task of searching for conditional independencies across a network by observing whether probabilistic evidence, that becomes available at a node, x , can propagate and influence our certainty about another node, y .

Serial or head-to-tail connection. This type of node connection is shown in Figure 15.6a. Evidence on x will influence the certainty about y , which in turn will influence that of z . This is also true for the reverse direction, starting from z and propagating to x . However, if the state of y is known, then x and z become (conditionally) independent. In this case, we say that y *blocks* the path from x to z and vice versa. When the state at a node is fixed/known, we say that the node is *instantiated*.

**FIGURE 15.6**

Three different types of connections: (a) serial, (b) diverging, and (c) converging.

Diverging or tail-to-tail connection. In this type of connection, shown in Figure 15.6b, evidence can propagate from y to x and from y to z, and also from x to z and from z to x via y, unless y is instantiated. In the latter case, y *blocks* the path from x to z and vice versa. That is, x and z become independent given the value of y. For example, if y represents “flu,” x “runny nose,” and z “sneezing,” then if we do not know whether someone has the flu, then a runny nose is evidence that can change our certainty about her/him having the flu; this in turn changes our belief about sneezing. However, if we know that someone has the flu, seeing the nose running gives no extra information about sneezing.

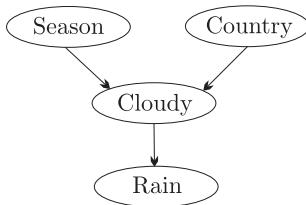
Converging or head-to-head connection or v-structure. This type of connection is slightly more subtle than the previous two cases, and it is shown in Figure 15.6c. Evidence from x does not propagate to z and thus cannot change our certainty about it. Knowing something about x tells us nothing about z. For example, let z denote either of two countries (e.g., England and Greece), x “season,” and y “cloudy weather.” Obviously, knowing the season says nothing about a country. However, having some evidence about cloudy weather y, then knowing that it is summer provides information that can change our certainty about the country. This is in accordance with our intuition. Knowing that it is summer and that the weather is cloudy *explains away* that the country is Greece. This is the reason we sometimes refer to this type of reasoning as *explaining away*. Explaining away is an instance of a general reasoning pattern called *intercausal reasoning*, where different causes of the same effect can interact; this is a very common pattern of reasoning in humans.

For this particular type of connection, explaining away is also achieved by evidence that is provided by *any one* of the descendants of y. Figure 15.7 illustrates the case via an example. Having evidence about the rain will also establish a path so that evidence about the season (country), x (z), changes our certainty about the country (season), z (x).

To recapitulate, let us stress the delicate point here. For the first two cases, head-to-tail and tail-to-tail, the path is blocked if node y is instantiated, that is, when its state is disclosed to us. However, in the head-to-head connection, the path between x and z “*opens*” when probabilistic evidence becomes available, either at y or *at any one* of its descendants.

Definition 15.3. Let G be a BN structure, and let x_1, \dots, x_k comprise a chain of nodes. Let Z be a subset of *observed* variables. The chain, x_1, \dots, x_k , is said to be *active given* the set, Z , if

- whenever a converging connection, $x_{i-1} \rightarrow x_i \leftarrow x_{i+1}$, is present in the chain, then either x_i or one of its descendants is in Z .
- no other node in the chain is in Z .

**FIGURE 15.7**

Having some evidence about either the weather being cloudy or rainy establishes the path for information flow between the nodes “season” and “country.”

In other words, in an active chain, probabilistic evidence can flow from x_1 to x_k and vice versa because no nodes (links), which can block this information flow, are present.

Definition 15.4. Let G be a BN structure and let X, Y, Z be three mutually disjoint sets of nodes in G . We say that X and Y are *d-separated* given Z if there is *no* active chain between *any* node $x \in X$ and $y \in Y$ given Z . If these are not *d-separated*, we say that they are *d-connected*.

In other words, if two variables x and y are *d-separated* by a third one z then observing the state of z , blocks any evidence propagation from x to y and vice versa. That is, *d-separation implies conditional independence*. Moreover, the following very important theorem holds.

Theorem 15.3. *Let the pair (G, p) be a Bayesian network. For every three mutually disjoint subsets of nodes X, Y, Z , whenever X and Y are *d-separated*, given Z , then for every pair $(x, y) \in X \times Y$, x and y are conditionally independent in p given Z .*

The proof of the theorem was given in [45]. In other words, this theorem guarantees that *d-separation implies conditional independence* on any probability distribution that factorizes over G . Note that, unfortunately, the opposite is not true. There may be conditional independencies that cannot be identified by *d-separation* (e.g., Problem 15.5). However, for most practical applications, the reverse is also true. The number of distributions that do not comply with the reverse statement of the theorem is infinitesimally small (see, e.g., [25]). Identification of all *d-separations* in a graph can be carried out via a number of efficient algorithms (e.g., [25, 32]).

Example 15.2. Consider the DAG G of Figure 15.8, connecting two nodes x, y . It is obvious that these nodes are not *d-separated* and comprise an active chain. Consider the following probability distribution, which factorizes over G , with

$$P(y = 0|x = 0) = 0.2, \quad P(y = 1|x = 0) = 0.8,$$

$$P(y = 0|x = 1) = 0.2, \quad P(y = 1|x = 1) = 0.8.$$

It can easily be checked out that $P(y|x) = P(y)$ (independent of the values of $P(x = 1)$, $P(x = 0)$) and the variables x and y are independent; this cannot be predicted by observing the *d-separations*. Note, however, that if we slightly perturb the values of the conditional probabilities, then the resulting distribution has as many independencies as those predicted by the *d-separations*; that is, in this case, none. As a matter of fact, this is a more general result. If we have a distribution that has independencies that are not predicted by *d-separations*, a small perturbation will almost always eliminate them (e.g., [25]).

**FIGURE 15.8**

This DAG involves no nodes that are d -separated.

Example 15.3. Consider the DAG shown in Figure 15.9. The red nodes indicate that the respective random variables have been observed; that is, these nodes have been instantiated. Node x_5 is d -connected to x_1, x_2, x_6 . In contrast, node x_9 is d -separated from all the rest. Indeed, evidence starting from x_1 is blocked by x_3 . However, it propagates via x_4 (instantiated and converging connection) to x_2, x_6 and then to x_5 (x_7 is instantiated and converging connection). In contrast, any flow of evidence toward x_9 is blocked by the instantiation of x_7 . It is interesting to note that, although all neighbors of x_5 have been instantiated, still it remains d -connected with other nodes.

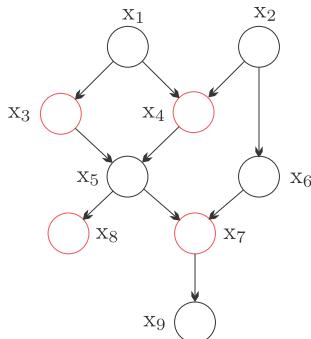
Definition 15.5. The *Markov blanket* of a node is the set of nodes comprising (a) its parents, (b) its children, and (c) the nodes sharing a child with this node. Once all the nodes in the blanket of a node are instantiated, then the node becomes d -separated from the rest of the network (Problem 15.7).

For example, in Figure 15.9, the Markov blanket of x_5 comprises the nodes x_3, x_4, x_8, x_7 , and x_6 . Note that if all these nodes are instantiated, then x_5 becomes d -separated from the rest of the nodes.

In the sequel, we give some examples of machine learning tasks, which can be cast in terms of a Bayesian graphical representation. As we will discuss, for many practical cases, the involved conditional probability distributions are expressed in terms of a set of parameters.

15.3.4 SIGMOIDAL BAYESIAN NETWORKS

We have already seen that when the involved random variables are discrete, the conditional probabilities $P(x_i|Pa_i)$, $i = 1, \dots, l$, associated with the nodes of a Bayesian graph structure have to be learned from the training data. If the number of possible states and/or the number of the variables in Pa_i is large

**FIGURE 15.9**

Red nodes are instantiated. Node x_5 is d -connected to x_1, x_2, x_6 and node x_9 is d -separated from all the nonobserved variables.

enough, this amounts to a large number of probabilities that have to be learned; thus, a large number of training points is required in order to obtain good estimates. This can be alleviated by expressing the conditional probabilities in a parametric form, that is,

$$P(x_i|\text{Pa}_i) = P(x_i|\text{Pa}_i; \theta_i), \quad i = 1, 2, \dots, l. \quad (15.13)$$

In case of binary valued variables, a common functional form is to view P as a logistic regression model; we used this model in the context of relevance vector machines in [Chapter 13](#). Adopting this model, we have

$$P(x_i = 1|\text{Pa}_i; \theta_i) = \sigma(t_i) = \frac{1}{1 + \exp(-t_i)}, \quad (15.14)$$

$$t_i := \theta_{i0} + \sum_{k:x_k \in \text{Pa}_i} \theta_{ik} x_k. \quad (15.15)$$

This reduces the number of parameter vectors to be learned to $O(l)$. The exact number of parameters depends on the size of the parent sets. Assuming the maximum number of parents for a node to be K , then the unknown number of parameters to be learned from the training data is less than or equal to lK . Taking into account the binary nature of the variables, we can write that

$$P(x_i|\text{Pa}_i; \theta_i) = x_i \sigma(t_i) + (1 - x_i)(1 - \sigma(t_i)), \quad (15.16)$$

where t_i is given in Eq. (15.15).

Such models are also known as *sigmoidal Bayesian networks*, and they have been proposed as one type of neural network ([Chapter 18](#)) (e.g., [33]). [Figure 15.10](#) presents the graphical structure of such a network. The network can be treated as a BN structure by associating a binary variable at each node and interpreting nodes' activations as probabilities, as dictated by Eq. (15.16). Performing inference and training of the parameters in such networks is not an easy task. We have to resort to approximations. We will come to this in [Section 16.3](#).

15.3.5 LINEAR GAUSSIAN MODELS

The computational advantages of the Gaussian pdf have recurrently been exploited in this book. We will now see the advantage gains in the framework of graphical models when the conditional pdf at every node, given the values of its parents, is expressed in a Gaussian form. Let

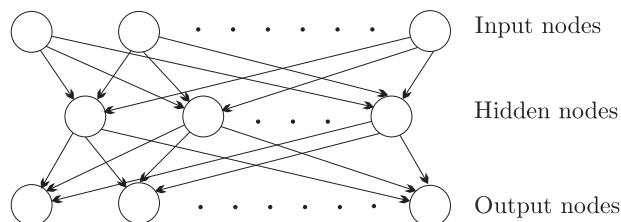


FIGURE 15.10

A sigmoidal Bayesian network.

$$p(x_i|\text{Pa}_i) = \mathcal{N}\left(x_i \left| \sum_{k:x_k \in \text{Pa}_i} \theta_{ik}x_k + \theta_{i0}, \sigma_i^2\right.\right), \quad (15.17)$$

where σ_i^2 is the respective variance and θ_{i0} is the bias term. From the properties of a Bayesian network, the joint pdf will be given by the product of the conditional probabilities (Theorem 15.2, which is valid for Gaussians), and the respective logarithm is given by

$$\ln p(\mathbf{x}) = \sum_{i=1}^l \ln p(x_i|\text{Pa}_i) = -\sum_{i=1}^l \frac{1}{2\sigma_i^2} \left(x_i - \sum_{k:x_k \in \text{Pa}_i} \theta_{ik}x_k - \theta_{i0} \right)^2 + \text{constant}. \quad (15.18)$$

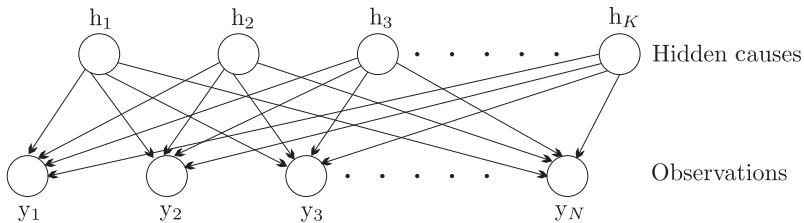
This is of a quadratic form, hence it is also of a Gaussian nature. The mean values and the covariance matrices for each one of the variables can be computed *recursively* in a straightforward way (Problem 15.8).

Note the computational elegance of such a Bayesian network. In order to obtain the joint pdf, one has only to sum up all the exponents, that is, an operation of linear complexity. Moreover, concerning training, one could readily think of a way to learn the unknown parameters; adopting the maximum likelihood method (although it may not be, necessarily, the best method), optimization with respect to the unknown parameters is a straightforward task. In contrast, one cannot make similar comments for the training of the sigmoidal Bayesian network. Unfortunately, products of sigmoid functions do not lead to an easy computational procedure. In such cases, one has to resort to approximations. For example, one way is to employ the variational bound approximation, as discussed in Chapter 13, in order to enforce, locally, a Gaussian functional form. We will discuss this technique in Section 16.3.1.

15.3.6 MULTIPLE-CAUSE NETWORKS

In the beginning of this chapter, we started with an example from the field of medical informatics. We were given a set of diseases and a set of symptoms/findings. The conditional probabilities for each symptom being absent, given the presence of a disease (Eq. (15.2)) were assumed known. We can consider the diseases as hidden causes (h) and the symptoms as observed variables (y), in a learning task. This can be represented in terms of a Bayesian network structure as in Figure 15.11. For the previous medical example, the variables h correspond to d (diseases) and the observed variables, y , to the findings, f .

However, the Bayesian structure given in Figure 15.11 can serve the needs of a number of inference and pattern recognition tasks, and sometimes it is referred to as a *multiple-cause network*, for obvious reasons. For example, in a machine vision application, the hidden causes, h_1, h_2, \dots, h_k , may refer to the presence or absence of an object, and $y_n, n = 1, \dots, N$, may correspond to the values of the observed pixels in an image [15]. The hidden variables can be binary (presence or absence of the respective object) and the conditional pdf can be formulated into a parameterized form, that is, $p(y_n|h; \theta)$. The specific form of the pdf captures the way objects interact as well as the effects of the noise. Note that in this case, the Bayesian network has a mixed set of variables, the observations are continuous, and the hidden causes binary. We will return to this type of Bayesian structure when discussing approximate inference methods in Section 16.3.

**FIGURE 15.11**

The general structure of a multiple-cause Bayesian network. The top-level nodes correspond to the hidden causes and the bottom ones to the observations.

15.3.7 I-MAPS, SOUNDNESS, FAITHFULNESS, AND COMPLETENESS

We have seen a number of definitions and theorems referring to the notion of conditional independence in graphs and probability distributions. Before we proceed further, it will be instructive to summarize what has been said and provide some definitions that will dress up our findings in a more formal language. This will prove useful for subsequent generalizations.

We have seen that a Bayesian network is a DAG that encodes a number of conditional independencies. Some of them are local ones, defined by the parent-child links, and some of them are of a more global nature and are the result of d -separations. Given a DAG, G , we denote as $I(G)$ the set of all independencies that correspond to d -separations. Also, let p be a probability distribution over a set of random variables, x_1, \dots, x_l . We denote as $I(p)$ the set of all independence assertions of the type $x_i \perp x_j | Z$ that hold true for the distribution p .

Let G be a DAG and p a distribution that factorizes over G ; in other words, it satisfies the local independencies as suggested by G . Then, we have seen (Theorem 15.3) that

$$I(G) \subseteq I(p). \quad (15.19)$$

We say that G is an *I-map* (independence map) for p . This property is sometimes referred to as *soundness*.

Definition 15.6. A distribution p is *faithful* to a graph G , if *any* independence in p is reflected in the d -separation properties of the graph.

In other words, the graph can represent all (and only) the conditional independence properties of the distribution. In such a case we write $I(p) = I(G)$. If equality is valid, we say that the graph, G , is a *perfect map* for p . Unfortunately, this is *not* valid for any distribution, p , that factorizes over G . However, for most practical purposes, $I(G) = I(P)$, which is true for *almost all* distributions that factorize over G .

Although $I(p) = I(G)$ is not valid for all distributions that factorize over G , the following two properties are always valid for any Bayesian network structure (e.g., [25]).

- If $x \perp y | Z$ for *all* distributions p that factorize over G , then x and y are d -separated given Z .
- If x and y are d -connected given Z , then there will be some distribution that factorizes over G where x and y are *dependent*.

A final definition concerns minimality.

Definition 15.7. A graph, G , is said to be *minimal I-map* for a set of independencies if the removal of any of its edges renders it *not* to be an I-map.

Note that a minimal I-map is not necessarily a perfect map. In the same way that there exist algorithms to find the set of d -separations, there exist algorithms to find perfect and minimal I-maps for a distribution (e.g., [25]).

15.4 UNDIRECTED GRAPHICAL MODELS

Bayesian structures and networks are not the only way to encode independencies in distributions. As a matter of fact, the directionality assigned to the edges of a DAG, while being advantageous and useful in some cases, becomes a disadvantage in others. A typical example is that of four variables, x_1, x_2, x_3, x_4 . There is no directed graph that can encode the following conditional independencies simultaneously: $x_1 \perp x_4 | \{x_2, x_3\}$ and $x_2 \perp x_3 | \{x_1, x_4\}$. Figure 15.12 shows the possible DAGs; notice that both fail to capture the desired independencies.

In 15.12a, $x_1 \perp x_4 | \{x_2, x_3\}$ because both paths, which connect x_1 and x_4 , are blocked. However, x_2 and x_3 are d -connected given x_1 and x_4 (Why?). In 15.12b, $x_2 \perp x_3 | \{x_1, x_4\}$ because the diverging links are blocked. However, we have violation of the other independence. (Why?)

Such situations can be overcome by resorting to undirected graphs. We will also see that this type of graphical modeling leads to a simplification concerning our search for conditional independencies.

Undirected graphical models or *Markov networks* have their roots in *Markov random fields* (MRF) in statistical physics. As was the case with the Bayesian models, each node of the graph is associated with a random variable. Edges connecting nodes are undirected, giving no preference to either of the two directions. Local interactions among connected nodes are expressed via functions of the involved variables, but they do not necessarily express probabilities. One can view these local functional interactions as a way to encode information related to the affinity/similarity among the involved variables. These local functions are known as *potential functions* or *compatibility functions* or *factors*, and they are *nonnegative*, usually positive, functions of their arguments. Moreover, as we will soon see, the global description of such a model is the result of the product of these local potential functions; this is in analogy to what holds true for the Bayesian networks.

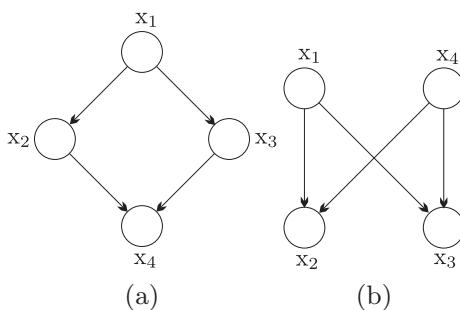


FIGURE 15.12

None of these DAGs can capture the two independencies: $x_1 \perp x_4 | \{x_2, x_3\}$ and $x_2 \perp x_3 | \{x_1, x_4\}$.

Following a path similar to that used for the directed graphs, we will begin with the factorization properties of a distribution over an MRF and then move on to study conditional independencies. Let x_1, \dots, x_l be a set of random variables that are grouped in K groups, $\mathbf{x}_1, \dots, \mathbf{x}_K$; each random vector, \mathbf{x}_k , $k = 1, 2, \dots, K$, involves a subset of the random variables, x_i , $i = 1, 2, \dots, l$.

Definition 15.8. A distribution is called a *Gibbs* distribution if it can be factorized in terms of a set of potential functions, ψ_1, \dots, ψ_K , such as

$$p(x_1, \dots, x_l) = \frac{1}{Z} \prod_{k=1}^K \psi_k(\mathbf{x}_k). \quad (15.20)$$

The constant Z is known as the *partition* function and it is the normalizing constant to guarantee that $p(x_1, \dots, x_l)$ is a probability distribution. Hence,

$$Z = \int \cdots \int \prod_{k=1}^K \psi_k(\mathbf{x}_k) dx_1, \dots, dx_l, \quad (15.21)$$

which becomes a summation for the case of probabilities.

Note that nobody can prohibit us from assigning conditional probability distributions as potential functions and making (15.20) identical to Eq. (15.10); in this case, normalization is not explicitly required because each one of the conditional distributions is normalized. However, MRFs can deal with more general cases.

Definition 15.9. We say that a Gibbs distribution, p , factorizes over an MRF, H , if *each* group of the variables, \mathbf{x}_k , $k = 1, 2, \dots, K$, involved in the K factors of the distribution p , forms a *complete subgraph* of H . Every complete subgraph of an MRF is known as a *clique*, and the corresponding factors of the Gibbs distribution are known as *clique potentials*.

Figure 15.13a shows an MRF and two cliques. Note that the set of nodes $\{x_1, x_3, x_4\}$ does not comprise a clique because the respective subgraph is not fully connected. The same applies to the set $\{x_1, x_2, x_3, x_4\}$. In contrast, the sets $\{x_1, x_2, x_3\}$ and $\{x_3, x_4\}$ form cliques. The fact that all variables in a group, \mathbf{x}_k , that are involved in the respective factor $\psi_k(\mathbf{x}_k)$ form a clique means that *all* these variables mutually interact, and the factor is a measure of such an interaction/dependence.

A clique is called *maximal* if we cannot include any other node from the graph in the set without its ceasing to be a clique. For example, both cliques in Figure 15.13a are maximal cliques. On the other hand, the clique in Figure 15.13b formed by $\{x_1, x_2, x_3\}$ is not maximum because bringing x_4 into the new set $\{x_1, x_2, x_3, x_4\}$ is also a clique. The same holds true for the clique formed by $\{x_2, x_3, x_4\}$.

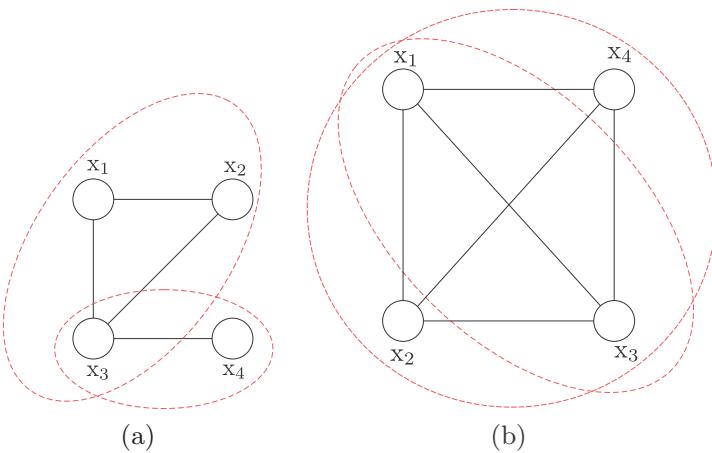
15.4.1 INDEPENDENCIES AND I-MAPS IN MARKOV RANDOM FIELDS

We will now state the equivalent theorem of d -separation, which was established for Bayesian network structures; recall the respective definition in Section 15.3.3, via the notion of an active chain.

Definition 15.10. Let H be an MRF and let x_1, x_2, \dots, x_k comprise a path.³ If Z is a set of observed variables/nodes, the path is said to be *active given Z* if none of x_1, x_2, \dots, x_k , is in Z .

Given three disjoint sets, X, Y, Z , we say that the nodes of X are *separated* by the nodes of Y , given Z , if there is no active path between X and Y , given Z . Note that the previous definition is much

³ Because edges are undirected, the notions of “chain” and “path” become identical.

**FIGURE 15.13**

(a) There are two cliques encircled by the red lines. (b) There are as many possible cliques as the combinations of the points in pairs, in triples, and so on. Considering all points together also forms a clique, and this is a maximal clique.

simpler compared to the respective definition given for the Bayesian network structures. According to the current definition, for a set X to be separated from a set Y given a third set Z , it suffices that *all* possible paths from X to Y pass via Z . Figure 15.14 illustrates the geometry. In 15.14a, there is no active path connecting the nodes in X from the nodes in Y given the nodes in Z . In 15.14b, there exist active paths connecting X and Y given Z .

Let us now denote by $I(H)$ the set of all possible statements of the type “ X separated by Y given Z .” This is in analogy to the set of all possible d -separations associated with a Bayesian network structure. The following theorem (soundness) holds true (Problem 15.10).

Theorem 15.4. *Let p be a Gibbs distribution that factorizes over an MRF H . Then, this is an I-map for p , that is,*

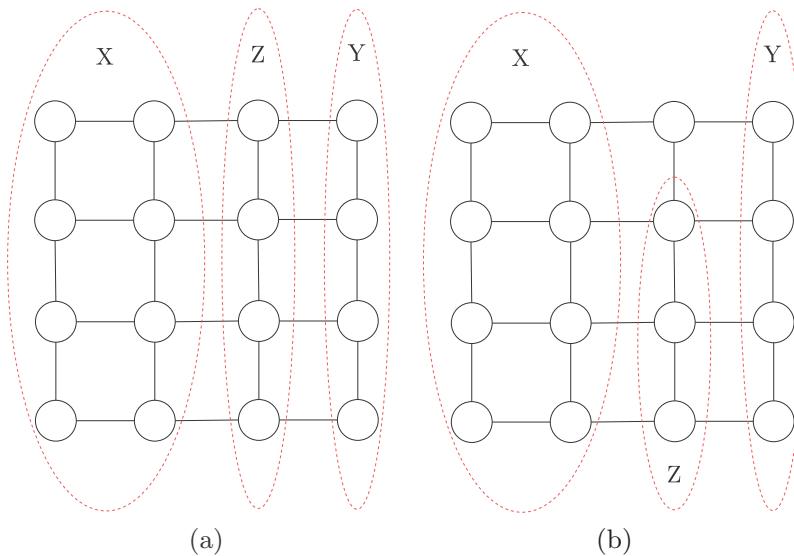
$$I(H) \subseteq I(p). \quad (15.22)$$

This is the counterpart of Theorem 15.3 in its “I-map formulation” as introduced in Section 15.3.7. Moreover, given that an MRF, H , is an I-map for a distribution, p , then p factorizes over H . Note that this holds true for Bayesian network structures; indeed, if $I(G) \subseteq I(p)$, then p factorizes over G (Problem 15.12). However, for MRFs, it is only true for strictly positive Gibbs distributions and it is given by the following *Hammersley-Clifford theorem*.

Theorem 15.5. *Let H be an MRF over a set of random variables, x_1, \dots, x_l , described by a probability distribution, $p > 0$. If H is an I-map for p , then p is a Gibbs distribution that factorizes over H .*

For a proof of this theorem, the interested reader is referred to the original paper [22] and also [5].

Our final touch on independencies in the context of MRFs concerns the notion of completeness. As was the case with the Bayesian networks, if p factorizes over an MRF, this does not necessarily establish completeness, although it is true for almost all practical cases. However, the weaker version

**FIGURE 15.14**

(a) The nodes of X and Y are separated by the nodes of Z . (b) There exist active paths that connect the nodes of X with the nodes of Y , given Z .

holds. That is, if x and y are two nodes in an MRF, which are *not* separated given a set Z , then there exists a Gibbs distribution, p , which factorizes over H and according to which x and y are dependent, given the variables in Z (see, e.g., [25]).

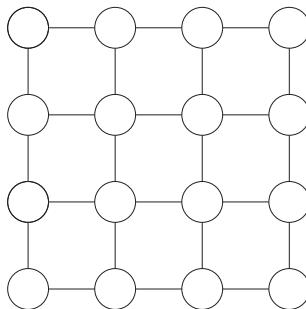
15.4.2 THE ISING MODEL AND ITS VARIANTS

The origin of the theory on Markov random fields is traced back to the discipline of statistical physics, and since then it has extensively been used in a number of different disciplines, including machine learning. In particular, in image processing and machine vision, MRFs have been established as a major tool in tasks such as de-noising, image segmentation, and stereo reconstruction (see, e.g., [29]). The goal of this section is to state a basic and rather primitive model, which, however, demonstrates the way information is captured and subsequently processed by such models.

Assume that each random variable takes binary values in $\{-1, 1\}$ and that the joint probability distribution is given by the following model,

$$p(x_1, \dots, x_l) := p(\mathbf{x}) = \frac{1}{Z} \exp \left(- \sum_i \left(\sum_{j>i} \theta_{ij} x_i x_j + \theta_{i0} x_i \right) \right), \quad (15.23)$$

where $\theta_{ij} = 0$ if the respective nodes are not connected. It is readily seen that this model is the result of the product of potential functions (factors), each one of an exponential form, defined on cliques of size two. Also, $\theta_{ij} = \theta_{ji}$ and we sum such as $i < j$ in order to avoid duplication. This model was

**FIGURE 15.15**

The graph of an MRF with pairwise dependencies among the nodes.

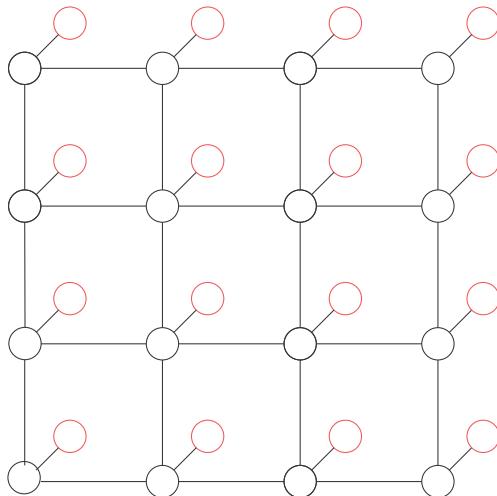
originally used by Ising in 1924 in his doctoral thesis to model phase transition phenomena in magnetic materials. The ± 1 of each node in the lattice models the two possible spin directions of the respective atoms. If $\theta_{ij} > 0$, interacting atoms tend to align spins in the same direction in order to decrease energy (ferromagnetism). The opposite is true if $\theta_{ij} < 0$. The corresponding graph is given in [Figure 15.15](#).

This basic model has been exploited in computer vision and image processing for tasks such as image de-noising, image segmentation, and scene analysis. Let us take, as an example, a binarized image and let x_i denote the noiseless pixel values (± 1). Let y_i be the observed noisy pixels whose values have been corrupted by noise and have changed polarity; see [Figure 15.16](#) for the respective graph. The task is to obtain the noiseless pixel values. One can rephrase the model in Eq. (15.23) to the needs of this task and rewrite it as [5, 21],

$$P(\mathbf{x}|\mathbf{y}) = \frac{1}{Z} \exp \left(\sum_i \left(\alpha \sum_{j>i} x_i x_j + \beta x_i y_i \right) \right), \quad (15.24)$$

where we have used only two parameters, α , β . Moreover, the summation $\sum_{j>i}$ involves only *neighboring* pixels. The goal now becomes that of estimating the pixel values, x_i , by maximizing the conditional (on the observations) probability. The adopted model is justified by the following two facts: (a) for low enough noise levels, most of the pixels will have the same polarity as the respective observations; this is encouraged by the presence of the product $x_i y_i$, where similar signs contribute to higher probability values; and (b) neighboring pixels are encouraged to have the same polarity because we know that real-world images tend to be smooth, except at the points that lie close to the edges in the image. Sometimes, a term $c x_i$, for an appropriately chosen value of c , is also present if we want to penalize either of the two polarities. The max-product or max-sum algorithms, to be discussed later in this chapter, are possible algorithmic alternatives for the maximization of the joint probability given in Eq. (15.24). However, these are not the only algorithmic possibilities to perform the optimization task. A number of alternative schemes that deal with inference in MRFs have been developed and studied. Some of them are suboptimal, yet they enjoy computational efficiency. Some classical references on the use of MRFs in image processing are [7–9, 47]. A number of variants of the basic Ising model result if one writes it as

$$P(\mathbf{x}) = \frac{1}{Z} \exp \left(- \sum_i \left(\sum_{j>i} f_{ij}(x_i x_j) + f_i(x_i) \right) \right) \quad (15.25)$$

**FIGURE 15.16**

A pairwise MRF as in [Figure 15.15](#), but now the observed values associated with each node are separately denoted as red nodes. For the image de-noising task, black nodes correspond to the noiseless pixel values (hidden variables) and red nodes to the observed pixel values.

and uses different functional forms for $f_{ij}(\cdot, \cdot)$ and $f_i(\cdot)$, and also by allowing the variables to take more than two values. This is sometimes known as the *Potts* model. In general, MRF models of the general form of Eq. (15.23) are also known as *pairwise MRFs undirected graphs* because the dependence among nodes is expressed in terms of products of pairs of variables. Further information on the applications of MRFs in image processing can be found in, for example, [\[29, 40\]](#).

Another name for Eq. (15.23) is Boltzmann distribution, where, usually, the variables take values in $\{0, 1\}$. Such a distribution has been used in *Boltzmann machines*, [\[23\]](#). Boltzmann machines can be seen as the stochastic counterpart of Hopfield networks; the latter have been proposed to act as *associative memories* as well as a way to attack combinatoric optimization problems (e.g., [\[31\]](#)). The interest in Boltzmann machines has been revived in the context of deep learning, and we will discuss them in more detail in [Chapter 18](#).

15.4.3 CONDITIONAL RANDOM FIELDS (CRFs)

All the graphical models (directed and undirected) that have been discussed so far evolve around the joint distribution of the involved random variables and its factorization on a corresponding graph. More recently, there is a trend to focus on the conditional distribution of some of the variables given the rest. The focus on the joint pdf originates from our interest in developing generative learning models.

However, this may not always be the most efficient way to deal with learning tasks, and we have already talked in [Chapters 3 and 7](#) about the discriminative learning alternative. Let us assume that from the set of the jointly distributed variables, some correspond to output target variables, whose variables are to be inferred when the rest are observed. For example, the target variables may correspond to the labels in a classification task and the rest to the (input) features.

Let us denote the former set by the vector \mathbf{y} and the latter by \mathbf{x} . Instead of focusing on the joint distribution $p(\mathbf{x}, \mathbf{y})$, it may be more sensible to focus on $p(\mathbf{y}|\mathbf{x})$. In [\[27\]](#), graphical models were adopted to encode the conditional distribution, $p(\mathbf{y}|\mathbf{x})$.

A *conditional random Markov field* is an undirected graph, H , whose nodes correspond to the joint set of random variables, (\mathbf{x}, \mathbf{y}) , but we now assume that the conditional distribution is factorized, that is,

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{k=1}^K \psi_k(\mathbf{x}_k, \mathbf{y}_k), \quad (15.26)$$

where $\{\mathbf{x}_k, \mathbf{y}_k\} \subseteq \{\mathbf{x}, \mathbf{y}\}$, $k = 1, 2, \dots, K$, and

$$Z(\mathbf{x}) = \int p(\mathbf{y}|\mathbf{x}) d\mathbf{y}, \quad (15.27)$$

where for discrete distributions the integral becomes summation. To stress the difference with Eq. [\(15.20\)](#), note that there it is the joint distribution of all the involved variables that is factorized. As a result, and observing Eqs. [\(15.20\)](#) and [\(15.26\)](#), it turns out that the normalization constant is now a function of \mathbf{x} . This seemingly minor difference can offer a number of advantages in practice. Avoiding the explicit modeling of $p(\mathbf{x})$, we have the benefit of using as inputs variables with complex dependencies, because we do not care to model them. This has led CRFs to be applied in a number of applications such as text mining, bioinformatics, and computer vision. Although we are not going to get involved with CRFs from now on, it suffices to say that the efficient inference techniques, which will be discussed in subsequent sections, can also be adapted, with only minor modifications, to the case of CRFs. For a tutorial on CRFs, including a number of variants and techniques concerning inference and learning, the interested reader is referred to [\[44\]](#).

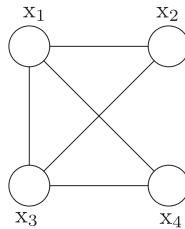
15.5 FACTOR GRAPHS

In contrast to a Bayesian network, an MRF does not necessarily indicate the specific form of factorization of the corresponding Gibbs distribution. Looking at a Bayesian network, the factorization evolves along the conditional distributions allocated in each node. Let us look at the MRF of [Figure 15.17](#). The corresponding Gibbs distribution could be written as

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \psi_1(x_1, x_2) \psi_2(x_1, x_3) \psi_3(x_3, x_2) \psi_4(x_3, x_4) \psi_5(x_1, x_4), \quad (15.28)$$

or

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \psi_1(x_1, x_2, x_3) \psi_2(x_1, x_3, x_4). \quad (15.29)$$

**FIGURE 15.17**

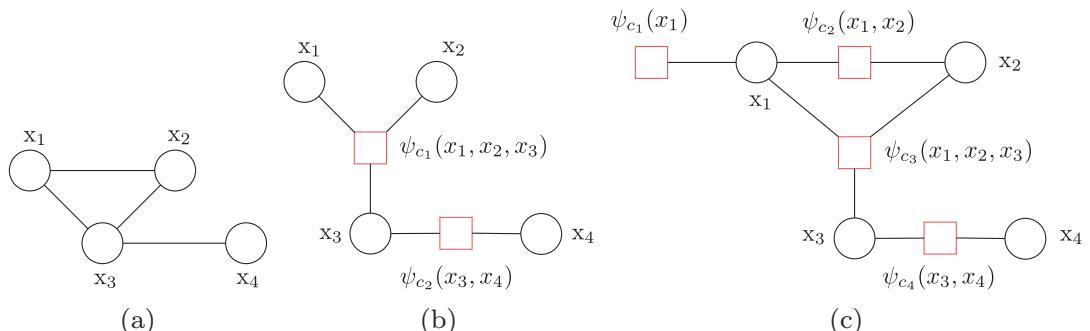
The Gibbs distribution can be written as a product of factors involving the cliques (x_1, x_2) , (x_1, x_3) , (x_3, x_4) , (x_3, x_2) , (x_1, x_4) or of (x_1, x_2, x_3) , (x_1, x_3, x_4) .

As an extreme case, if all the points of an MRF form a maximal clique, as is the case in Figure 15.13b, we could include only a single product term. Note that aiming at maximal cliques reduces the number of factors, but at the same time the complexity is increased; for example, this can amount to an exponential explosion in the number of terms that have to be learned in the case of discrete variables. At the same time, using large cliques hides modeling details. On the other hand, smaller cliques allow us to be more explicit and detailed in our description.

Factor graphs provide us with the means of making the decomposition of a probability distribution into a product of factors more explicit. A factor graph is an undirected bipartite graph involving two types of nodes (thus the term bipartite): one that corresponds to the random variables, denoted by circles; and one to the potential functions, denoted by squares. Edges exist only between two different types of nodes, that is, between “potential function” nodes and “variable” nodes [15, 16, 26].

Figure 15.18a is an MRF for four variables. The respective factor graph in Figure 15.18b corresponds to the product

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \psi_{c_1}(x_1, x_2, x_3) \psi_{c_2}(x_3, x_4), \quad (15.30)$$

**FIGURE 15.18**

(a) An MRF and (b), (c) possible equivalent factor graphs at different fine-grained factorization in terms of product factors.

and the one in Figure 15.18c to

$$p(x_1, x_2, x_3, x_4) = \frac{1}{Z} \psi_{c_1}(x_1) \psi_{c_2}(x_1, x_2) \psi_{c_3}(x_1, x_2, x_3) \psi_{c_4}(x_3, x_4). \quad (15.31)$$

As an example, if the potential functions were chosen to express “interactions” among variables using probabilistic information, the involved functions in Figure 15.18 may be chosen as

$$\psi_{c_1}(x_1, x_2, x_3) = p(x_3|x_1, x_2)p(x_2|x_1)p(x_1), \quad (15.32)$$

and

$$\psi_{c_2}(x_3, x_4) = p(x_4|x_3). \quad (15.33)$$

For the case of Figure 15.18c,

$$\psi_{c_1}(x_1) = p(x_1), \quad \psi_{c_2}(x_1, x_2) = p(x_2|x_1)$$

$$\psi_{c_3}(x_1, x_2, x_3) = p(x_3|x_1, x_2), \quad \psi_{c_4}(x_3, x_4) = p(x_4|x_3).$$

For such an example, in both cases, it is readily seen that $Z = 1$. We will soon see that factor graphs turn out to be very useful for inference computations.

Remarks 15.1.

- A variant of the factor graphs has been more recently introduced, known as *normal factor graphs* (NFG). In an NFG, edges represent variables and vertices represent factors. Moreover, latent and observable variables (internal and external) are distinguished by being represented by edges of degree 2 and degree 1, respectively. Such models can lead to simplified learning algorithms and can nicely unify a number of previously proposed models (see, e.g., [2, 3, 18, 19, 30, 34, 35]).

15.5.1 GRAPHICAL MODELS FOR ERROR-CORRECTING CODES

Graphical models are extensively used for representing a class of error-correcting codes. In the *block parity check* codes (e.g., [31]), one sends k information bits (0, 1 for a binary code) in a block of N bits, $N > k$; thus, redundancy is introduced into the system to cope with the effects of noise in the transmission channel. The extra bits are known as parity-check bits. For each code, a parity-check matrix, H , is defined; in order to be a valid one, for each code word, x , it must satisfy the parity check constraint (modulo-2 operations), $Hx = \mathbf{0}$. Take as an example the case of $k = 3$ and $N = 6$. The code comprises 2^3 (2^k in general) code words, each of them of length $N = 6$ bits. For the parity-check matrix,

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix},$$

the eight code words that satisfy the parity check constraint are 000000, 001011, 010101, 011110, 100110, 101101, 110011, and 111000. In each one of the eight words, the first three are the information bits and the remaining ones the parity-check bits, which are uniquely determined in order to satisfy the parity-check constraint. Each one of the three parity-check constraints can be expressed via a function, that is,

$$\psi_1(x_1, x_2, x_4) = \delta(x_1 \oplus x_2 \oplus x_4),$$

$$\psi_2(x_1, x_3, x_5) = \delta(x_1 \oplus x_3 \oplus x_5),$$

$$\psi_3(x_2, x_3, x_6) = \delta(x_2 \oplus x_3 \oplus x_6),$$

where $\delta(\cdot)$ is equal to one or zero, depending on whether its argument is one or zero, respectively, and \oplus denotes the modulo-2 addition. The code words are transmitted to a noisy memoryless binary symmetric channel, where each transmitted bit, x_i , may be flipped over and be received as, y_i , according to the following rule:

$$P(y = 0|x = 1) = p, \quad P(y = 1|x = 1) = 1 - p,$$

$$P(y = 1|x = 0) = p, \quad P(y = 0|x = 0) = 1 - p.$$

Upon reception of the observation sequence, y_i , $i = 1, 2, \dots, N$, one has to decide the value, x_i , that was transmitted. Because the channel has been assumed memoryless, every bit is affected by the noise independently of the other bits, and the overall posterior probability of each codeword is proportional to

$$\prod_{i=1}^N P(x_i|y_i).$$

In order to guarantee that only valid codewords are considered, and assuming *equiprobable* information bits, we write the joint probability as

$$P(x, y) = \frac{1}{Z} \psi_1(x_1, x_2, x_4) \psi_2(x_1, x_3, x_5) \psi_3(x_2, x_3, x_6) \prod_{i=1}^N P(y_i|x_i),$$

where the parity-check constraints have been taken into account. The respective factor model is shown in Figure 15.19, where

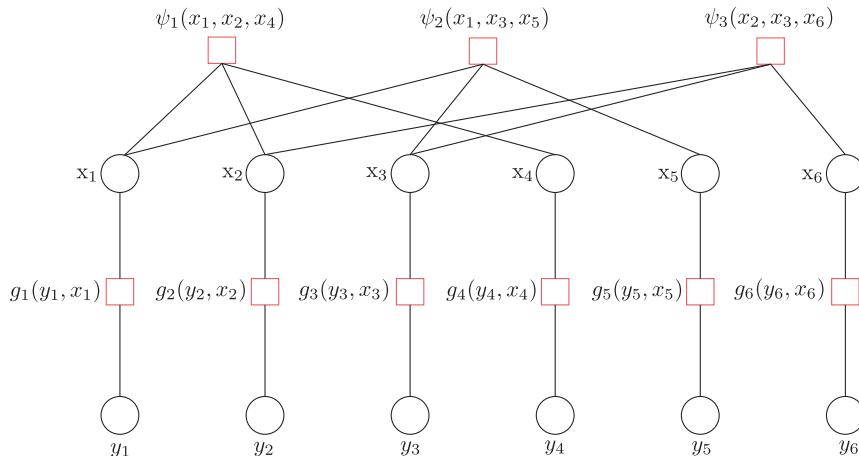


FIGURE 15.19

Factor graph for a (3,3) parity-check code.

$$g_i(y_i, x_i) = P(y_i|x_i).$$

The task of decoding is to derive an efficient inference scheme to compute the posteriors, and based on that to decide in favor of 1 or 0.

15.6 MORALIZATION OF DIRECTED GRAPHS

At a number of points we have already made bridges between Bayesian networks and MRFs. In this section, we will formalize the bridge and see how one can convert a Bayesian network to an MRF and discuss the subsequent effects of such a conversion on the implied conditional independencies.

We can trust common sense to drive us to construct such a conversion. Because the conditional distributions will play the role of the potential functions (factors), one has to make sure that edges do exist among all the involved variables in each one of these factors. Because edges from the parents to children exist, we have to (a) retain these edges and make them undirected and (b) add edges between nodes that are parents of a common child. This is shown in [Figure 15.20](#). In [Figure 15.20a](#), a DAG is shown, which is converted to the MRF of [Figure 15.20b](#) by adding undirected edges between x_1, x_2 (parents of x_3) and x_3, x_6 (parents of x_5). The procedure is known as *moralization* and the resulting undirected graph as a *moral graph*. The terminology stems from the fact that “parents are forced to be married.” This conversion will be very useful soon, when an inference algorithm will be stated that covers both Bayesian networks and MRFs in a unifying framework.

The obvious question that is now raised is how moralization affects independencies. It turns out that if H is the resulting moral graph, then $I(H) \subseteq I(G)$ ([Problem 15.11](#)). In other words, the moral graph can guarantee a smaller number of independencies compared to the original BN via its set of d -separations. This is natural because one adds extra links. For example, in [Figure 15.20a](#), x_1, x_2 in the converging node $x_1 \rightarrow x_3 \leftarrow x_2$ are marginally independent, not given x_3 . However, in the resulting moral graph in [15.20b](#), this independence is lost. It can be shown that moralization adds the fewest extra links and hence retains the maximum number of independence, see for example, [25].

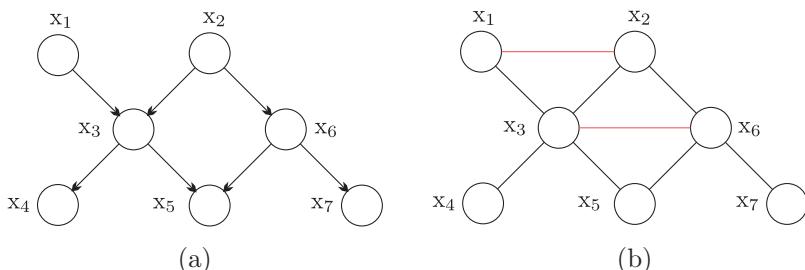


FIGURE 15.20

(a) A DAG and (b) the resulting MRF after applying moralization on the DAG. Directed edges become undirected and new edges, shown in red, are added to “marry” parents with common child-nodes.

15.7 EXACT INFERENCE METHODS: MESSAGE-PASSING ALGORITHMS

This section deals with efficient techniques for inference on undirected graphical models. So, even if our starting point was a BN, we assume that it was converted to an undirected one prior to the inference task. We will begin with the simplest case of graphical models—graphs comprising a chain of nodes. This will help the reader to grasp the basic notions behind exact inference schemes. It is interesting to note that, in general, the inference task in graphical models is an NP-hard one [10]. Moreover, it has been shown that for general Bayesian networks, approximate inference to a desired number of digits of precision is also an NP-hard task, [11]; that is, the time required has an exponential dependence on the number of digits of accuracy. However, as we will see in a number of cases that are commonly encountered in practice, the exponential growth in computational time can be bypassed by exploiting the underlying independencies and factorization properties of the associated distributions.

The inference tasks of interest are (a) computing the likelihood, (b) computing the marginals of the involved variables, (c) computing conditional probability distributions, and (d) finding modes of distributions.

15.7.1 EXACT INFERENCE IN CHAINS

Let us consider the chain graph of Figure 15.21 and focus our interest on computing marginals. The naive approach, which overlooks factorization and independencies, would be to compute first the joint distribution. Let us concentrate on discrete variables and assume that each one of them, l in total, has K states. Then, in order to compute the marginal of, say, x_j , we have to obtain the sum

$$\begin{aligned} P(x_j) &:= \sum_{x_1} \cdots \sum_{x_{j-1}} \sum_{x_{j+1}} \cdots \sum_{x_l} P(x_1, \dots, x_l) \\ &= \sum_{x_i : i \neq j} P(x_1, \dots, x_l). \end{aligned} \quad (15.34)$$

Each summation is over K values, hence, the number of the required computations amounts to $\mathcal{O}(K^l)$. Let us now bring factorization into the game and concentrate on computing $P(x_1)$. Assume that the joint probability factorizes over the graph, hence, we can write

$$P(\mathbf{x}) := P(x_1, x_2, \dots, x_l) = \frac{1}{Z} \prod_{i=1}^{l-1} \psi_{i,i+1}(x_i, x_{i+1}),$$

(15.35)

and



FIGURE 15.21

An undirected chain graph with l nodes. There are $l - 1$ cliques consisting of pairs of nodes.

$$P(x_1) = \frac{1}{Z} \sum_{x_i: i \neq 1} \prod_{i=1}^{l-1} \psi_{i,i+1}(x_i, x_{i+1}). \quad (15.36)$$

Note that the only term that depends on x_l is $\psi_{l-1,l}(x_{l-1}, x_l)$. Let us start by summing with respect to this last term, which leaves unaffected all the preceding factors in the sequence of products in Eq. (15.36),

$$P(x_1) = \frac{1}{Z} \sum_{x_i: i \neq 1, l} \prod_{i=1}^{l-2} \psi_{i,i+1}(x_i, x_{i+1}) \sum_{x_l} \psi_{l-1,l}(x_{l-1}, x_l), \quad (15.37)$$

where we exploited the basic property of arithmetic

$$\sum_i \alpha \beta_i = \alpha \sum_i \beta_i. \quad (15.38)$$

- Define:

$$\sum_{x_l} \psi_{l-1,l}(x_{l-1}, x_l) := \mu_b(x_{l-1}).$$

Because the possible values of the pair (x_{l-1}, x_l) comprise a table with K^2 elements, the summation involves K^2 terms and $\mu_b(x_{l-1})$ consists of K possible values.

- After marginalizing out x_l , the only factor in the product that depends on x_{l-1} is

$$\psi_{l-2,l-1}(x_{l-2}, x_{l-1}) \mu_b(x_{l-1}).$$

Then, in a similar way as before we obtain

$$P(x_1) = \frac{1}{Z} \sum_{x_i: i \neq 1, l-1, l} \prod_{i=1}^{l-3} \psi_{i,i+1}(x_i, x_{i+1}) \sum_{x_{l-1}} \psi_{l-2,l-1}(x_{l-2}, x_{l-1}) \mu_b(x_{l-1}),$$

where this summation also involves K^2 terms.

We are now ready to define the general recursion as

$$\mu_b(x_i) := \sum_{x_{i+1}} \psi(x_i, x_{i+1}) \mu_b(x_{i+1}), \quad (15.39)$$

$$\mu_b(x_l) = 1,$$

whose repeated application leads to

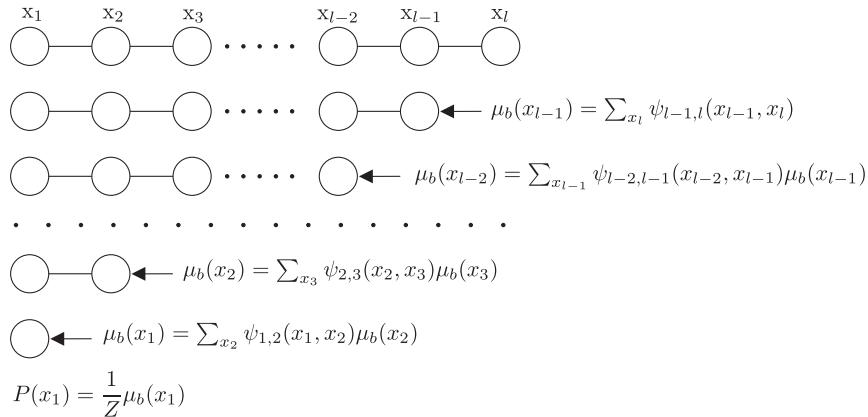
$$\mu_b(x_1) = \sum_{x_1} \psi_{1,2}(x_1, x_2) \mu_b(x_2),$$

and finally

$$P(x_1) = \frac{1}{Z} \mu_b(x_1). \quad (15.40)$$

The series of recursions is illustrated in [Figure 15.22](#).

We can think that every node, x_i , (a) receives a message from its right, $\mu_b(x_i)$, which for our case comprises K values, (b) performs locally sum-multiply operations and a new message $\mu_b(x_{i-1})$ is

**FIGURE 15.22**

To compute $P(x_1)$, starting from the last node, x_l , each node (a) receives a message; (b) processes it locally via sum and product operations, which produces a new message; and (c) the latter is passed backward, to the node on its left. We have assumed that $\mu_b(x_l) = 1$.

computed, which (c) is passed to its left, to node x_{l-1} . The subscript “*b*,” in μ_b , denotes “backward” to remind us of the flow of the message-passing activity from right to left.

If we wanted to compute $P(x_l)$, we would adopt the same reasoning but start summation from x_1 . In this case, message-passing takes place forward (from left to right) and messages are defined as

$$\mu_f(x_{i+1}) := \sum_{x_i} \psi_{i,i+1}(x_i, x_{i+1})\mu_f(x_i), \quad i = 1, \dots, l-1,$$

$$\mu_f(x_1) = 1,$$

(15.41)

where “*f*” has been used to denote “forward” flow. The procedure is shown in [Figure 15.23](#).

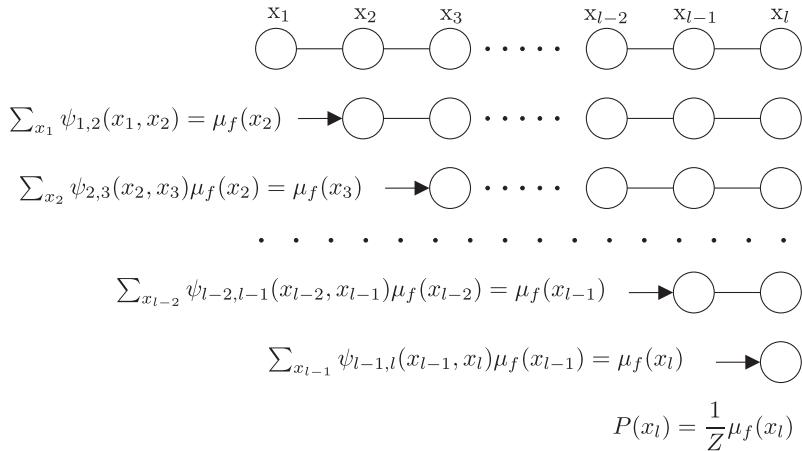
The term $\mu_b(x_j)$ is the result of summing the products over $x_{j+1}, x_{j+2}, \dots, x_l$, and the term $\mu_f(x_j)$ over x_1, x_2, \dots, x_{j-1} . At each iteration step, one variable is *eliminated* by summing up over all its possible values. It can be easily shown, following similar arguments, that the marginal at any point, x_j , $2 \leq j \leq l-1$ is obtained by ([Problem 15.13](#))

$$P(x_j) = \frac{1}{Z} \mu_f(x_j)\mu_b(x_j), \quad j = 2, 3, \dots, l-1. \quad (15.42)$$

The idea is to perform one forward and one backward message-passing operation, store the values, and then compute any one of the marginals of interest. The total cost will be $\mathcal{O}(2K^2l)$, instead of K^l of the naive approach.

We still have to compute the normalizing constant Z . This is readily obtained by summing up both sides of Eq. (15.42), which requires $\mathcal{O}(K)$ operations,

$$Z = \sum_{x_j=1}^K \mu_f(x_j)\mu_b(x_j). \quad (15.43)$$

**FIGURE 15.23**

To compute $P(x_l)$, message-passing takes place in the forward direction, from left to right. As opposed to Figure 15.22, messages are denoted as μ_f , to remind us of the forward flow.

So far, we have considered the computation of marginal probabilities. Let us now turn our attention to their conditional counterparts. We start with the simplest case, for example, to compute $P(x_j|x_k = \hat{x}_k)$, $k \neq j$. That is, we assume that variable x_k has been observed and its value is \hat{x}_k . The first step in computing the conditional is to recover the joint $P(x_j, x_k = \hat{x}_k)$. This is a normalized version of the respective conditional, which can then be obtained as

$$P(x_j|x_k = \hat{x}_k) = \frac{P(x_j, x_k = \hat{x}_k)}{P(\hat{x}_k)}. \quad (15.44)$$

The only difference in computing $P(x_j, x_k = \hat{x}_k)$ compared to the previous computations of the marginals is that now, in order to obtain the messages, we *do not sum* with respect to x_k . We just clump the respected potential function to its value \hat{x}_k . That is, the computations

$$\begin{aligned} \mu_b(x_{k-1}) &= \sum_{x_k} \psi_{k-1,k}(x_{k-1}, x_k) \mu_b(x_k), \\ \mu_f(x_{k+1}) &= \sum_{x_k} \psi_{k,k+1}(x_k, x_{k+1}) \mu_f(x_k), \end{aligned}$$

are replaced by

$$\begin{aligned} \mu_b(x_{k-1}) &= \psi_{k-1,k}(x_{k-1}, \hat{x}_k) \mu_b(\hat{x}_k), \\ \mu_f(x_{k+1}) &= \psi_{k,k+1}(\hat{x}_k, x_{k+1}) \mu_f(\hat{x}_k). \end{aligned}$$

In other words, x_k is considered a delta function at the instantiated value. Once $P(x_j, x_k = \hat{x}_k)$ has been obtained, normalization is straightforward and is locally performed at the j th node. The procedure can be generalized when more than one variable is observed.

15.7.2 EXACT INFERENCE IN TREES

Having gained experience and learned the basic secrets in developing efficient inference algorithms for chains, we turn our attention to the more general case involving *tree-structured* undirected graphical models.

A tree is a graph in which there is a single path between any two nodes of the graph; thus, there are no cycles in the graph. Figures 15.24a and b are two examples of trees. Note that in a directed tree, any node has only a single parent. A tree can be directed or undirected. Furthermore, because in a directed one there are no children with two parents, the moralization step, which converts a directed graph to an undirected one, adds no extra links. The only change consists of making the edges undirected.

There is an important property of the trees, which will prove very important for our current needs. Let us denote a tree graph as T , which is a collection of vertices/nodes V and edges E , which link the nodes, that is, $T = \{V, E\}$. Consider any node $x \in V$ and consider the set of all its neighbors, that is, all nodes that share an edge with x . Denote this set as

$$\mathcal{N}(x) = \{y \in V : (x, y) \in E\}.$$

Looking at Figure 15.24b, we have that $\mathcal{N}(x) = \{y, u, z, v\}$. Then, for each element $r \in \mathcal{N}(x)$, define the subgraph $T_r = \{V_r, E_r\}$ such that any node in this subgraph can be reached from r via paths that do not pass through x . In Figure 15.24b, the respective subgraphs, each associated with one element in $\{y, u, z, v\}$, are encircled by dotted lines. By the definition of a tree, it can easily be deduced that each one of these subgraphs is also a tree. Moreover, these subgraphs are *disjoint*. In other words, each one of the neighboring nodes of a node, for example, x , can be viewed as a *root* of a subtree, and these subtrees are mutually disjoint, having no common nodes. This property will allow us to break a large problem into a number of smaller ones. Moreover, each one of the smaller problems can be further divided in

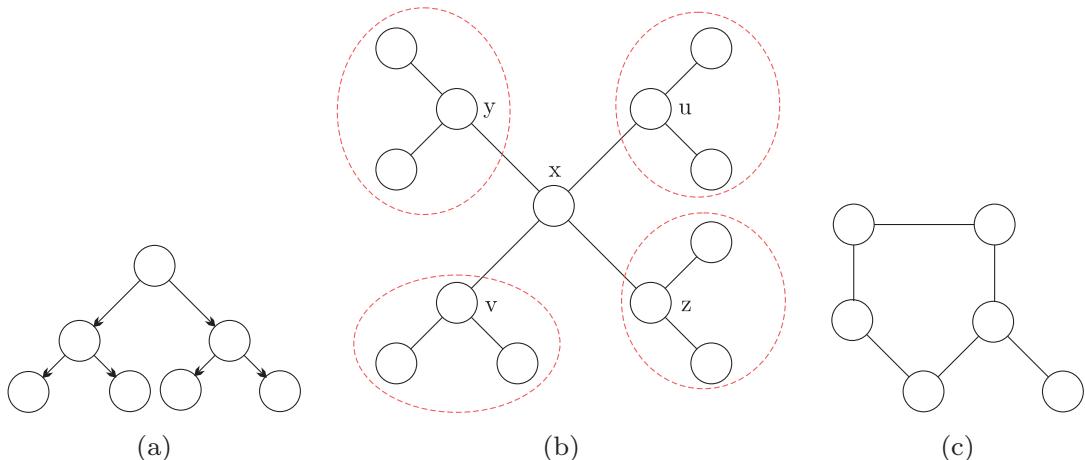
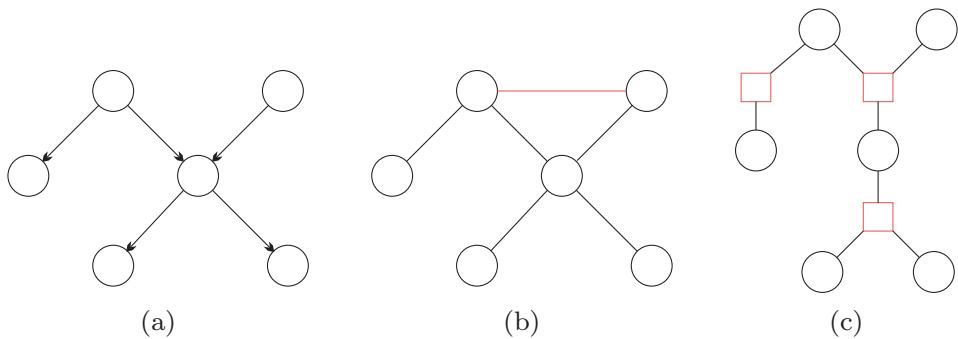


FIGURE 15.24

Examples of (a) directed and (b) undirected trees. Note that in the directed one, any node has a single parent. In both cases, there is only a single chain that connects any two nodes. (c) The graph is not a tree because there is a cycle.

**FIGURE 15.25**

(a) Although there are no cycles, one of the nodes has two parents, hence the graph is a polytree. (b) The resulting structure after moralization has a cycle. (c) A factor graph for the polytree in (a).

the same way, being itself a tree. We have now all the basic ingredients to derive an efficient scheme for inference on trees (recall that such a breaking of a large problem into a sequence of smaller ones was at the heart of the message-passing algorithm for chains). However, let us first bring the notion of factor graphs into the scene. The reason is that using factor graphs allows us to deal with some more general graph structures, such as *polytrees*.

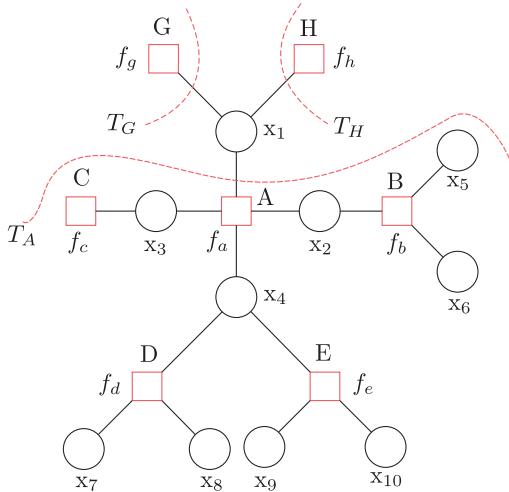
A directed polytree is a graph in which, although there are no cycles, a child may have more than one parent. Figure 15.25a shows an example of a polytree. The unpleasant situation results after the moralization step because marrying the parents results in cycles, and we *cannot* derive *exact* inference algorithms in graphs with cycles. However, if one converts the original directed polytree into a factor graph, the resulting bipartite entity has a tree structure, with no cycles involved. Thus, everything we said before about tree structures applies to these factor graphs.

15.7.3 THE SUM-PRODUCT ALGORITHM

We will develop the algorithm in a “bottom-up” approach, via the use of an example. Once the rationale is understood, the generalization can readily be obtained. Let us consider the factor tree of Figure 15.26. The factor nodes are denoted by capital letters and squares, and each one is associated with a potential function. The rest are variable nodes, denoted by circles. Assume that we want to compute the marginal $P(x_1)$. Node x_1 , being a variable node, is connected to factor nodes only. We split the graph in as many (tree) subgraphs as the factor nodes connected to x_1 (three in our case). In the figure, each one of these subgraphs is encircled, having as roots the nodes A, H, and G, respectively. Recall that the joint $P(\mathbf{x})$ is given as the product of all the potential functions, each one associated with one factor node, divided by the normalizing constant, Z. Focusing on the node of interest, x_1 , this product can be written as

$$P(\mathbf{x}) = \frac{1}{Z} \psi_A(x_1, \mathbf{x}_A) \psi_H(x_1, \mathbf{x}_H) \psi_G(x_1, \mathbf{x}_G), \quad (15.45)$$

where \mathbf{x}_A denotes the vector corresponding to all the variables in T_A ; the vectors \mathbf{x}_H and \mathbf{x}_G are similarly defined. The function $\psi_A(x_1, \mathbf{x}_A)$ is the product of all the potential functions associated with the factor

**FIGURE 15.26**

The tree is subdivided into three subtrees, each one having as its root one of the factor nodes connected to x_1 . This is the node whose marginal is computed in the text. Messages are initiated from the leaf nodes toward x_1 . Once messages arrive at x_1 , a new propagation of messages starts, this time from x_1 to the leaves.

nodes in T_A , and $\psi_H(x_1, \mathbf{x}_H)$, $\psi_G(x_1, \mathbf{x}_G)$ are defined in an analogous way. Then, the marginal of interest is given by

$$P(x_1) = \frac{1}{Z} \sum_{\mathbf{x}_A \in V_A} \sum_{\mathbf{x}_H \in V_H} \sum_{\mathbf{x}_G \in V_G} \psi_A(x_1, \mathbf{x}_A) \psi_H(x_1, \mathbf{x}_H) \psi_G(x_1, \mathbf{x}_G). \quad (15.46)$$

We will concentrate on the subtree with root A, denoted as $T_A := \{V_A, E_A\}$, where V_A stands for the nodes in T_A and E_A for the respective set of edges. Because the three subtrees are disjoint, we can split the previous expression in Eq. (15.46) into

$$P(x_1) = \frac{1}{Z} \sum_{\mathbf{x}_A \in V_A} \psi_A(x_1, \mathbf{x}_A) \sum_{\mathbf{x}_H \in V_H} \psi_H(x_1, \mathbf{x}_H) \sum_{\mathbf{x}_G \in V_G} \psi_G(x_1, \mathbf{x}_G). \quad (15.47)$$

Note that $x_1 \notin V_A \cup V_H \cup V_G$. Having reserved the symbol $\psi_A(\cdot, \cdot)$ to denote the product of all the potentials in the subtree T_A (and similarly for T_H , T_G), let us denote the individual potential functions, for each one of the factor nodes, via the symbol f , as shown in Figure 15.26. Thus, we can now write

$$\begin{aligned} \sum_{\mathbf{x}_A \in V_A} \psi_A(x_1, \mathbf{x}_A) &= \sum_{\mathbf{x}_A \in V_A} f_a(x_1, x_2, x_3, x_4) f_c(x_3) f_b(x_2, x_5, x_6) \\ &\quad \times f_d(x_4, x_7, x_8) f_e(x_4, x_9, x_{10}) \\ &= \sum_{x_2} \sum_{x_3} \sum_{x_4} f_a(x_1, x_2, x_3, x_4) f_c(x_3) \sum_{x_7} \sum_{x_8} f_d(x_4, x_7, x_8) \\ &\quad \times \sum_{x_9} \sum_{x_{10}} f_e(x_4, x_9, x_{10}) \sum_{x_6} \sum_{x_5} f_b(x_2, x_5, x_6). \end{aligned} \quad (15.48)$$

Recall from our treatment of the chain graph that messages were nothing but locally computed summations over products. Having this experience, let us define as

$$\mu_{f_b \rightarrow x_2}(x_2) = \sum_{x_6} \sum_{x_5} f_b(x_2, x_5, x_6),$$

$$\mu_{f_e \rightarrow x_4}(x_4) = \sum_{x_9} \sum_{x_{10}} f_e(x_4, x_9, x_{10}),$$

$$\mu_{f_d \rightarrow x_4}(x_4) = \sum_{x_7} \sum_{x_8} f_d(x_4, x_7, x_8),$$

$$\mu_{f_c \rightarrow x_3}(x_3) = f_c(x_3),$$

$$\mu_{x_4 \rightarrow f_a}(x_4) = \mu_{f_d \rightarrow x_4}(x_4) \mu_{f_e \rightarrow x_4}(x_4),$$

$$\mu_{x_2 \rightarrow f_a}(x_2) = \mu_{f_b \rightarrow x_2}(x_2),$$

$$\mu_{x_3 \rightarrow f_a}(x_3) = \mu_{f_c \rightarrow x_3}(x_3),$$

and

$$\mu_{f_a \rightarrow x_1}(x_1) = \sum_{x_2} \sum_{x_3} \sum_{x_4} f_a(x_1, x_2, x_3, x_4) \mu_{x_2 \rightarrow f_a}(x_2) \mu_{x_3 \rightarrow f_a}(x_3) \mu_{x_4 \rightarrow f_a}(x_4).$$

Observe that we were led to define two types of messages; one type is passed from variable nodes to factor nodes and the other one for messages passed from factor to variable nodes.

- Variable node to factor node messages (Figure 15.27a):

$$\mu_{x \rightarrow f}(x) = \prod_{s: f_s \in \mathcal{N}(x) \setminus f} \mu_{f_s \rightarrow x}(x).$$

(15.49)

We use $\mathcal{N}(x)$ to denote the set of the nodes with which a variable node, x , is connected. $\mathcal{N}(x) \setminus f$ refers to all nodes *excluding* the factor node, f ; note that all these nodes are factor nodes. In other

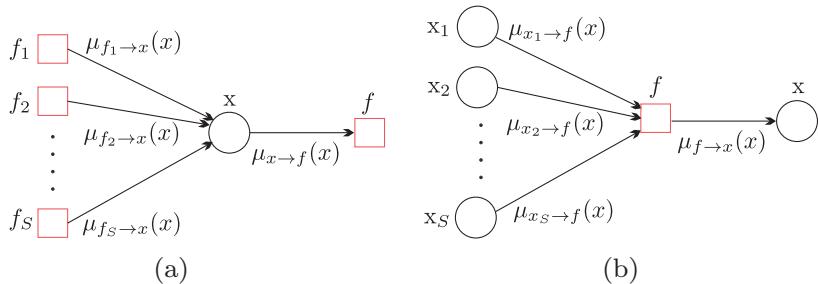


FIGURE 15.27

(a) Variable x is connected to S factor nodes, besides f ; that is, $\mathcal{N}(x) \setminus f = \{f_1, f_2, \dots, f_S\}$. The output message from x to f is the product of the incoming messages. The arrows indicate directions of flow of the message propagation. (b) The factor node f is connected to S node variables, besides x ; that is, $\mathcal{N}(f) \setminus x = \{x_1, x_2, \dots, x_S\}$.

words, the action of a variable node, as far as message-passing is concerned, is to multiply incoming messages. Obviously, if it is connected only to one factor node (except f), then such a variable node passes what it receives, without any computation. This is, for example, the case of $\mu_{x_2 \rightarrow f_a}$, as previously defined.

- Factor node to variable node messages (Figure 15.27b):

$$\boxed{\mu_{f \rightarrow x}(x) = \sum_{x_i \in \mathcal{N}(f) \setminus x} f(\mathbf{x}^f) \prod_{i: x_i \in \mathcal{N}(f) \setminus x} \mu_{x_i \rightarrow f}(x_i)}, \quad (15.50)$$

where $\mathcal{N}(f)$ denotes the set of the (variable) nodes connected to f and $\mathcal{N}(f) \setminus x$ the corresponding set if we exclude node x . The vector \mathbf{x}^f comprises all the variables involved as arguments in f , that is, all the variables/nodes in $\mathcal{N}(f)$.

If a node is a leaf, we adopt the following convention. If it is a variable node, x , connected to a factor node, f , then

$$\mu_{x \rightarrow f}(x) = 1. \quad (15.51)$$

If it is a factor node, f , connected to variable node, x , then

$$\mu_{f \rightarrow x}(x) = f(x). \quad (15.52)$$

Adopting the previously stated definitions, Eq. (15.48) is now written as

$$\begin{aligned} & \sum_{x_A \in V_A} \psi_A(x_1, \mathbf{x}_A) \\ &= \sum_{x_2} \sum_{x_3} \sum_{x_4} f_a(x_1, x_2, x_3, x_4) \mu_{x_2 \rightarrow f_a}(x_2) \mu_{x_3 \rightarrow f_a}(x_3) \mu_{x_4 \rightarrow f_a}(x_4) \\ &= \mu_{f_a \rightarrow x_1}(x_1). \end{aligned} \quad (15.53)$$

Working similarly for the other two subtrees, T_G and T_H , we finally obtain

$$P(x_1) = \frac{1}{Z} \mu_{f_a \rightarrow x_1}(x_1) \mu_{f_g \rightarrow x_1}(x_1) \mu_{f_h \rightarrow x_1}(x_1). \quad (15.54)$$

Note that each summation can be viewed as a step that “removes” a variable and produces a message. This is the reason that sometimes this procedure is called *variable elimination*. We are now ready to summarize the steps of the algorithm.

The Algorithmic Steps

1. Pick the variable, x , whose marginal, $P(x)$, will be computed.
2. Divide the tree into as many subtrees as the number of factor nodes to which the variable node, x , is connected.
3. For each one of these subtrees, identify the leaf nodes.
4. Start message-passing, toward x , by initializing the leaf nodes according to Eqs. (15.51) and (15.52), by utilizing Eqs. (15.49) and (15.50).
5. Compute the marginal according to Eq. (15.54), or in general

$$P(x) = \frac{1}{Z} \prod_{s: f_s \in \mathcal{N}(x)} \mu_{f_s \rightarrow x}(x). \quad (15.55)$$

The normalizing constant, Z , can be obtained by adding both sides of Eq. (15.55) over all possible values of x . As was the case with the chain graphs, if a variable is observed, then we replace the summation over this variable, in the places where this is required, by a single term evaluated on the observed value.

Although so far we have considered discrete variables, everything that has been said also applies to continuous variables by substituting summations by integrals. For such integrations, Gaussian models turn out to be very convenient.

Remarks 15.2.

- Thus far, we have concentrated on the computation of the marginal for a single variable, x . If the marginal of another variable is required, the obvious way would be for the whole process to be repeated. However, as we already commented in subsection 15.7.1, such an approach is computationally wasteful because many of the computations are common and can be shared for the evaluation of the marginals of the various nodes. Note that in order to compute the marginal at any variable node, one needs all the messages from the factor nodes, to which the specific variable node is connected, to be available (15.55). Assume now that we pick one node, say x_1 , and compute the marginal, once all the required messages have “arrived.” Then, this node initiates a new message-propagation phase, this time toward the leaves. It is not difficult to see (Problem 15.15) that this process, once completed, will make available to every node all the required messages for the computation of the respective marginals. In other words, this two stage message-passing procedure, in two opposite-flow directions, suffices to provide all the necessary information for the computation of the marginals at every node. The total number of messages passed is just twice the number of edges in the graph. Similar to the case of chain graphs, in order to compute conditional probabilities, say $P(x_i|x_k = \hat{x}_k)$, node x_k has to be instantiated. Running the sum-product algorithm will provide the joint probability $P(x_i, x_k = \hat{x}_k)$, from which the respective conditional is obtained after normalization; this is performed locally at the respective variable node.
- The (joint) marginal probability of all the variables, x_1, x_2, \dots, x_S , associated with a factor node, f , is given by (Problem 15.16),

$$P(x_1, \dots, x_S) = \frac{1}{Z} f(x_1, \dots, x_S) \prod_{s=1}^S \mu_{x_s \rightarrow f}(x_s). \quad (15.56)$$

- Earlier versions of the sum-product algorithm, known as *belief propagation*, were first developed in the context of singly connected graphs independently in [28, 36, 37]. However, the problem of variable elimination has an older history and has been discovered in different communities (e.g., [4, 6, 39]). Sometimes, the general sum-product algorithm, as described before, is also called *the generalized forward-backward algorithm*.

15.7.4 THE MAX-PRODUCT AND MAX-SUM ALGORITHMS

Let us now turn our attention from marginals to modes of distributions. That is, given a distribution, $P(\mathbf{x})$, that factorizes over a tree (factor) graph, the task is to compute efficiently the quantity

$$\max_{\mathbf{x}} P(\mathbf{x}).$$

We will focus on discrete variables. Following similar arguments as before, one can readily write the counterpart of Eq. (15.46), for the case of Figure 15.26, as

$$\max_{\mathbf{x}} P(\mathbf{x}) = \frac{1}{Z} \max_{x_1} \max_{x_A \in V_A} \max_{x_H \in V_H} \max_{x_G \in V_G} \psi_A(x_1, \mathbf{x}_A) \psi_H(x_1, \mathbf{x}_H) \psi_G(x_1, \mathbf{x}_G). \quad (15.57)$$

Exploiting the property of the max operator, that is,

$$\max_{b,c} (ab, ac) = a \max_{b,c} (b, c), \quad a \geq 0,$$

we can rewrite Eq. (15.57) as

$$\max_{\mathbf{x}} P(\mathbf{x}) = \frac{1}{Z} \max_{x_1} \max_{x_A \in V_A} \psi_A(x_1, \mathbf{x}_A) \max_{x_H \in V_H} \psi_H(x_1, \mathbf{x}_H) \max_{x_G \in V_G} \psi_G(x_1, \mathbf{x}_G).$$

Following similar arguments as for the sum-product rule, we arrive at the counterpart of Eq. (15.48), that is,

$$\begin{aligned} \max_{x_1} \max_{x_A \in V_A} \psi_A(x_1, \mathbf{x}_A) &= \max_{x_1} \max_{x_2, x_3, x_4} f_a(x_1, x_2, x_3, x_4) f_c(x_3) \max_{x_7, x_8} f_d(x_4, x_7, x_8) \\ &\times \max_{x_9, x_{10}} f_e(x_4, x_9, x_{10}) \max_{x_5, x_6} f_b(x_2, x_6, x_5). \end{aligned} \quad (15.58)$$

Equation (15.58) suggests that everything that was said before for the sum-product message-passing algorithm holds true here, provided we replace summations with the max operations, the definitions of the messages passed between nodes change to

$$\mu_{x \rightarrow f}(x) = \prod_{s: f_s \in \mathcal{N}(x) \setminus f} \mu_{f_s \rightarrow x}(x), \quad (15.59)$$

and

$$\mu_{f \rightarrow x}(x) = \max_{x_i: x_i \in \mathcal{N}(f) \setminus x} f(x^f) \prod_{i: x_i \in \mathcal{N}(f) \setminus x} \mu_{x_i \rightarrow f}(x_i),$$

(15.60)

with the same definition of symbols as for Eq. (15.50). Then, the mode of $P(\mathbf{x})$ is given by

$$\max_{\mathbf{x}} P(\mathbf{x}) = \frac{1}{Z} \max_{x_1} \mu_{f_a \rightarrow x_1}(x_1) \mu_{f_g \rightarrow x_1}(x_1) \mu_{f_h \rightarrow x_1}(x_1), \quad (15.61)$$

or in general

$$\max_{\mathbf{x}} P(\mathbf{x}) = \frac{1}{Z} \max_{\mathbf{x}} \prod_{s: f_s \in \mathcal{N}(\mathbf{x})} \mu_{f_s \rightarrow x}(x), \quad (15.62)$$

where x is the node chosen to play the role of the root, toward which the flow of the messages is directed, starting from the leaves. The resulting scheme is known as the *max-product algorithm*.

In practice, an alternative formulation of the previously stated max-product algorithm is usually adopted. Often, the involved potential functions are probabilities (by absorbing the normalization constant) and their magnitude is less than one; however, if a large number of product terms is involved it may lead to arithmetic inaccuracies. A way to bypass this is to involve the logarithmic function, which transforms products into summations. This is justified by the fact that the logarithmic function is monotonic and increasing, hence it does not affect the point \mathbf{x} at which a maximum occurs, that is,

$$\mathbf{x}_* := \arg \max_{\mathbf{x}} P(\mathbf{x}) = \arg \max_{\mathbf{x}} \ln P(\mathbf{x}). \quad (15.63)$$

Under this formulation, the following *max-sum* version of the algorithm results. It is straightforward to see that Eqs. (15.59) and (15.60) now take the form of

$$\mu_{x \rightarrow f}(x) = \sum_{s: f_s \in \mathcal{N}(x) \setminus f} \mu_{f_s \rightarrow x}(x), \quad (15.64)$$

$$\mu_{f \rightarrow x}(x) = \max_{x_i: x_i \in \mathcal{N}(f) \setminus x} \left\{ \ln f(x^f) + \sum_{i: x_i \in \mathcal{N}(f) \setminus x} \mu_{x_i \rightarrow f}(x_i) \right\}. \quad (15.65)$$

In place of Eqs. (15.51) and (15.52) for the initial messages, sent by the leaf nodes, we now define

$$\mu_{x \rightarrow f}(x) = 0, \text{ and } \mu_{f \rightarrow x}(x) = \ln f(x). \quad (15.66)$$

Note that after one pass of the message flow, the maximum value of $P(\mathbf{x})$ has been obtained. However, one is also interested in knowing the corresponding value \mathbf{x}_* , for which the maximum occurs, that is,

$$\mathbf{x}_* = \arg \max_{\mathbf{x}} P(\mathbf{x}).$$

This is achieved by a reverse message-passing process, which is slightly different than what we have discussed so far, and it is known as back-tracking.

Back-tracking: Assume that x_1 is the chosen node to play the role of the root, where the flow of messages “converge.” From Eq. (15.61), we get

$$x_{1*} = \arg \max_{x_1} \mu_{f_a \rightarrow x_1}(x_1) \mu_{f_g \rightarrow x_1}(x_1) \mu_{f_h \rightarrow x_1}(x_1). \quad (15.67)$$

A new message-passing flow now starts, and the root node, x_1 , passes the obtained optimal value to the factor nodes, to which it is connected. Let us follow this message-passing flow within the nodes of the subtree T_A .

- Node A: It receives x_{1*} from node x_1 .
 - Selection of the optimal values: Recall that

$$\begin{aligned} \mu_{f_a \rightarrow x_1}(x_1) &= \max_{x_2, x_3, x_4} f_a(x_1, x_2, x_3, x_4) \mu_{x_4 \rightarrow f_a}(x_4) \\ &\quad \times \mu_{x_3 \rightarrow f_a}(x_3) \mu_{x_2 \rightarrow f_a}(x_2). \end{aligned}$$

Thus, for different values of x_1 , different optimal values for (x_2, x_3, x_4) will result. For example, assume that in our discrete variable setting, each variable can take one out of four possible values, that is, $x \in \{1, 2, 3, 4\}$. Then, if $x_{1*} = 2$, say that the resulting optimal values are $(x_{2*}, x_{3*}, x_{4*}) = (1, 1, 3)$. On the other hand, if $x_{1*} = 4$, then maximization may result to, let us say, $(x_{2*}, x_{3*}, x_{4*}) = (2, 3, 4)$. However, having obtained a *specific* value for x_{1*} via the maximization at node x_1 , we choose the triplet (x_{2*}, x_{3*}, x_{4*}) such as

$$\begin{aligned} (x_{2*}, x_{3*}, x_{4*}) &= \arg \max_{x_2, x_3, x_4} f_a(x_{1*}, x_2, x_3, x_4) \mu_{x_4 \rightarrow f_a}(x_4) \\ &\quad \times \mu_{x_3 \rightarrow f_a}(x_3) \mu_{x_2 \rightarrow f_a}(x_2). \end{aligned} \quad (15.68)$$

Hence, during the first pass, the obtained optimal values have to be stored to be used during the second (backward) pass.

- Message-passing: Node A passes x_{4*} to node x_4 , x_{2*} to node x_2 and x_{3*} to node x_3 .

- Node x_4 passes x_{4*} to nodes D and E.
- Node D
 - Selection of the optimal values: Select (x_{7*}, x_{8*}) such as

$$(x_{7*}, x_{8*}) = \arg \max_{x_7, x_8} f_d(x_{4*}, x_7, x_8) \mu_{x_7 \rightarrow f_d}(x_7) \mu_{x_8 \rightarrow f_d}(x_8).$$

- Message Passing: Node D passes (x_{7*}, x_{8*}) to nodes x_7, x_8 , respectively.

This type of flow spreads toward all the leaves and finally,

$$\mathbf{x}_* = \arg \max_{\mathbf{x}} P(\mathbf{x}) \quad (15.69)$$

is obtained. One may wonder why not use a similar two-stage message passing as we did with the sum-product rule, and recover x_{i*} , for each node i . This would be possible if there were a guarantee for a unique optimum, \mathbf{x}_* . If this is not the case and we have two optimal values, say, \mathbf{x}_*^1 and \mathbf{x}_*^2 , which result from Eq. (15.69), then we run the danger of failing to obtain them. To see this, let us take an example of four variables, x_1, x_2, x_3, x_4 , each taking values in the discrete set $\{1, 2, 3, 4\}$. Assume that $P(\mathbf{x})$ does not have a unique maximum and the two combinations for optimality are

$$(x_{1*}, x_{2*}, x_{3*}, x_{4*}) = (1, 1, 2, 3), \quad (15.70)$$

and

$$(x_{1*}, x_{2*}, x_{3*}, x_{4*}) = (1, 2, 2, 4). \quad (15.71)$$

Both of them are acceptable because they correspond to $\max P(x_1, x_2, x_3, x_4)$. The back-tracking procedure guarantees to give either of the two. In contrast, using two-stage message passing may result to a combination of values, for example,

$$(x_{1*}, x_{2*}, x_{3*}, x_{4*}) = (1, 1, 2, 4), \quad (15.72)$$

which does not correspond to the maximum. Note that this result is correct in its own rationale. It provides, for every node, a value for which an optimum may result. Indeed, searching for a maximum of $P(\mathbf{x})$, node x_2 , can take either the value of 1 or 2. However, what we want to find is the correct *combination* for all nodes. This is guaranteed by the back-tracking procedure.

Remarks 15.3.

The max-product (max-sum) algorithm is a generalization of the celebrated Viterbi algorithm [46], which has extensively been used in communications [17] and speech recognition [39]. The algorithm has been generalized to arbitrary commutative semirings on tree-structured graphs (e.g., [1, 13]).

Example 15.4. Consider the Bayesian network of Figure 15.28a. The involved variables are binary, $(0, 1)$, and the respective probabilities are

$$\begin{aligned} P(x = 1) &= 0.7, \quad P(x = 0) = 0.3, \\ P(w = 1) &= 0.8, \quad P(w = 0) = 0.2, \\ P(y = 1|x = 0) &= 0.8, \quad P(y = 0|x = 0) = 0.2, \\ P(y = 1|x = 1) &= 0.6, \quad P(y = 0|x = 1) = 0.4, \\ P(z = 1|y = 0) &= 0.7, \quad P(z = 0|y = 0) = 0.3, \end{aligned}$$

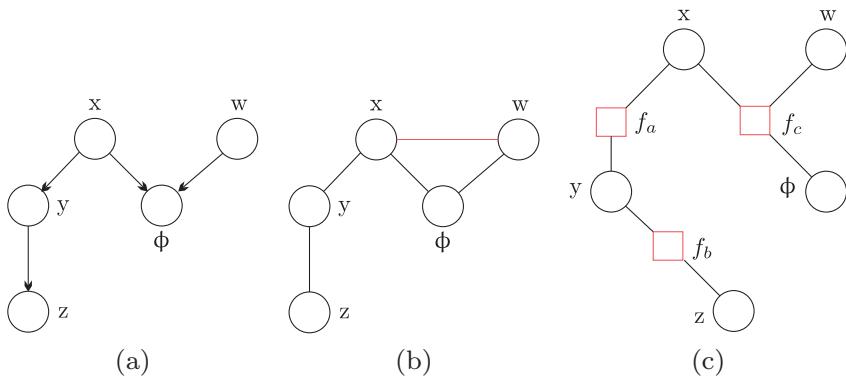


FIGURE 15.28

(a) The Bayesian network of [Example 15.4](#); (b) its moralized version, where the two parents of ϕ have been connected; and (c) a possible factor graph.

$$\begin{aligned}
P(z=1|y=1) &= 0.9, \quad P(z=0|y=1) = 0.1, \\
P(\phi=1|x=0, w=0) &= 0.25, \quad P(\phi=0|x=0, w=0) = 0.75, \\
P(\phi=1|x=1, w=0) &= 0.3, \quad P(\phi=0|x=1, w=0) = 0.7, \\
P(\phi=1|x=0, w=1) &= 0.2, \quad P(\phi=0|x=0, w=1) = 0.8, \\
P(\phi=1|x=1, w=1) &= 0.4, \quad P(\phi=0|x=1, w=1) = 0.6.
\end{aligned}$$

Compute the combination, $x_*, y_*, z_*, \phi_*, w_*$, which results in the maximum of the joint probability,

$$\begin{aligned} P(x, y, z, \phi, w) &= P(z|y, x, \phi, w)P(y|x, \phi, w)P(\phi|x, w)P(x|w)P(w) \\ &= P(z|y)P(y|x)P(\phi|x, w)P(x)P(w), \end{aligned}$$

which is the factorization imposed by the Bayesian network.

In order to apply the max-product rule, we first moralize the graph and then form a factor graph version, as shown in Figures 15.28b, c, respectively. The factor nodes realize the following potential (factor) functions.

$$\begin{aligned}f_a(x, y) &= P(y|x)P(x), \\f_b(y, z) &= P(z|y), \\f_c(\phi, x, w) &= P(\phi|x, w)P(w),\end{aligned}$$

and obviously

$$P(x, y, z, \phi, w) = f_a(x, y)f_b(y, z)f_c(\phi, x, w).$$

Note that in this case, the normalizing constant, $Z = 1$. Thus, the values these factor functions take, according to their previous definitions, are

$$f_a(x, y) : \begin{cases} f_a(1, 1) = 0.42 \\ f_a(1, 0) = 0.28 \\ f_a(0, 1) = 0.24 \\ f_a(0, 0) = 0.06 \end{cases}, \quad f_b(y, z) : \begin{cases} f_b(1, 1) = 0.9 \\ f_b(1, 0) = 0.1 \\ f_b(0, 1) = 0.7 \\ f_b(0, 0) = 0.3 \end{cases}$$

$$f_c(\phi, x, w) : \begin{cases} f_c(1, 1, 1) = 0.32 \\ f_c(1, 1, 0) = 0.06 \\ f_c(1, 0, 1) = 0.48 \\ f_c(1, 0, 0) = 0.14 \\ f_c(0, 1, 1) = 0.16 \\ f_c(0, 1, 0) = 0.05 \\ f_c(0, 0, 1) = 0.64 \\ f_c(0, 0, 0) = 0.15 \end{cases}$$

Note that the number of possible values of a factor explodes by increasing the number of the involved variables.

Application of the max-product algorithm: Choose as root the node x . Then the nodes z , ϕ , and w become the leaves.

- Initialization:

$$\mu_{z \rightarrow f_b}(z) = 1, \quad \mu_{\phi \rightarrow f_c}(\phi) = 1, \quad \mu_{w \rightarrow f_c}(w) = 1.$$

- Begin message-passing:

- $f_b \rightarrow y$:

$$\mu_{f_b \rightarrow y}(y) = \max_z f_b(y, z) \mu_{z \rightarrow f_b}(z),$$

or

$$\mu_{f_b \rightarrow y}(1) = 0.9, \quad \mu_{f_b \rightarrow y}(0) = 0.7,$$

where the first one occurs at $z = 1$ and the second one at $z = 0$.

- $y \rightarrow f_a$:

$$\mu_{y \rightarrow f_a}(y) = \mu_{f_b \rightarrow y}(y),$$

or

$$\mu_{y \rightarrow f_a}(1) = 0.9, \quad \mu_{y \rightarrow f_a}(0) = 0.7.$$

- $f_a \rightarrow x$:

$$\mu_{f_a \rightarrow x}(x) = \max_y f_a(x, y) \mu_{y \rightarrow f_a}(y),$$

or

$$\mu_{f_a \rightarrow x}(1) = 0.42 \cdot 0.9 = 0.378,$$

which occurs for $y = 1$. Note that for $y = 0$, the value for $\mu_{f_a \rightarrow x}(1)$ would be $0.7 \cdot 0.28 = 0.196$, which is smaller than 0.378. Also,

$$\mu_{f_a \rightarrow x}(0) = 0.24 \cdot 0.9 = 0.216,$$

which also occurs for $y = 1$.

- $f_c \rightarrow x$:

$$\mu_{f_c \rightarrow x}(x) = \max_{w,\phi} f_c(\phi, x, w) \mu_{w \rightarrow f_c}(w) \mu_{\phi \rightarrow f_c}(\phi),$$

or

$$\mu_{f_c \rightarrow x}(1) = 0.48,$$

which occurs for $\phi = 0$ and $w = 1$, and

$$\mu_{f_c \rightarrow x}(0) = 0.64,$$

which occurs for $\phi = 0$ and $w = 1$.

- Obtain the optimal value:

$$x_* = \arg \max \mu_{f_a \rightarrow x}(x) \mu_{f_c \rightarrow x}(x),$$

or

$$x_* = 1,$$

and the corresponding maximum value is

$$\max P(x, y, z, w, \phi) = 0.378 \cdot 0.48 = 0.1814.$$

- Back-tracking:

- Node f_c :

$$\max_{w,\phi} f_c(1, \phi, w) \mu_{w \rightarrow f_c}(w) \mu_{\phi \rightarrow f_c}(\phi),$$

which has occurred for

$$\phi_* = 0 \text{ and } w_* = 1.$$

- Node f_a :

$$\max_y f_a(1, y) \mu_{y \rightarrow f_a}(y),$$

which has occurred for

$$y_* = 1.$$

- Node f_b :

$$\max_z f_b(1, z) \mu_{z \rightarrow f_b}(z),$$

which has occurred for

$$z_* = 1.$$

Thus, the optimizing combination is

$$(x_*, y_*, z_*, \phi_*, w_*) = (1, 1, 1, 0, 1).$$

PROBLEMS

- 15.1** Show that in the product

$$\prod_{i=1}^n (1 - x_i),$$

the number of cross-product terms, x_1, x_2, \dots, x_k , $1 \leq k \leq n$, for all possible combinations of x_1, \dots, x_n is equal to $2^n - n - 1$.

- 15.2** Prove that if a probability distribution p satisfies the Markov condition, as implied by a BN, then p is given as the product of the conditional distributions given the values of the parents.
- 15.3** Show that if a probability distribution factorizes according to a Bayesian network structure, then it satisfies the Markov condition.
- 15.4** Consider a DAG and associate each node with a random variable. Define for each node the conditional probability of the respective variable given the values of its parents. Show that the product of the conditional probabilities yields a valid joint probability and that the Markov condition is satisfied.
- 15.5** Consider the graph in [Figure 15.29](#). Random variable x has two possible outcomes, with probabilities $P(x_1) = 0.3$ and $P(x_2) = 0.7$. Variable y has three possible outcomes, with conditional probabilities

$$\begin{aligned} P(y_1|x_1) &= 0.3, \quad P(y_2|x_1) = 0.2, \quad P(y_3|x_1) = 0.5, \\ P(y_1|x_2) &= 0.1, \quad P(y_2|x_2) = 0.4, \quad P(y_3|x_2) = 0.5. \end{aligned}$$

Finally, the conditional probabilities for z are

$$\begin{aligned} P(z_1|y_1) &= 0.2, \quad P(z_2|y_1) = 0.8, \\ P(z_1|y_2) &= 0.2, \quad P(z_2|y_2) = 0.8, \\ P(z_1|y_3) &= 0.4, \quad P(z_2|y_3) = 0.6. \end{aligned}$$

Show that this probability distribution, which factorizes over the graph, renders x and z independent. However, x and z in the graph are not d -separated because y is not instantiated.

- 15.6** Consider the DAG in [Figure 15.30](#). Detect the d -separations and d -connections in the graph.

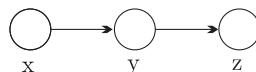
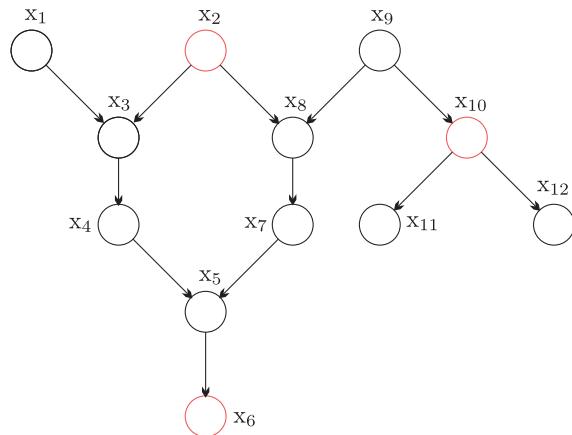
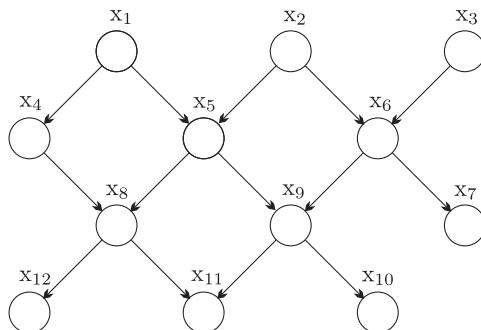


FIGURE 15.29

Graphical Model for [Problem 15.5](#).

**FIGURE 15.30**

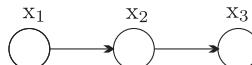
DAG for [Problem 15.6](#). Nodes in red have been instantiated.

**FIGURE 15.31**

The graph structure for [Problem 15.7](#).

- 15.7** Consider the DAG of [Figure 15.31](#). Detect the blanket of node x_5 and verify that if all the nodes in the blanket are instantiated, then the node becomes d -separated from the rest of the nodes in the graph.
- 15.8** In a linear Gaussian Bayesian network model, derive the mean values and the respective covariance matrices for each one of the variables in a recursive manner.
- 15.9** Assuming the variables associated with the nodes of the Bayesian structure of [Figure 15.32](#) to be Gaussian, find the respective mean values and covariances.
- 15.10** Prove that if p is a Gibbs distribution that factorizes over an MRF H , then H is an I-map for p .
- 15.11** Show that if H is the moral graph that results from moralization of a BN structure, then

$$I(H) \subseteq I(G).$$

**FIGURE 15.32**

Network for Problem 15.9.

- 15.12** Consider a Bayesian network structure and a probability distribution p . Then show that if $I(G) \subseteq I(p)$, then p factorizes over G .
- 15.13** Show that in an undirected chain graphical model, the marginal probability $P(x_j)$ of a node, x_j , is given by

$$P(x_j) = \frac{1}{Z} \mu_f(x_j) \mu_b(x_j),$$

where $\mu_f(x_j)$ and $\mu_b(x_j)$ are the received by the node forward and backward messages.

- 15.14** Show that the joint distribution of two neighboring nodes in an undirected chain graphical model is given by

$$P(x_j, x_{j+1}) = \frac{1}{Z} \mu_f(x_j) \psi_{j,j+1}(x_j, x_{j+1}) \mu_b(x_{j+1}).$$

- 15.15** Using Figure 15.26, prove that if there is a second message passing, starting from x , toward the leaves, then any node will have the available information for the computation of the respective marginals.
- 15.16** Consider the tree graph of Figure 15.26. Compute the marginal probability $P(x_1, x_2, x_3, x_4)$.
- 15.17** Repeat the message-passing procedure to find the optimal combination of variables for Example 15.4 using the logarithmic version and the max-sum algorithm.

REFERENCES

- [1] S.M. Ajji, R.J. McEliece, The generalized distributive law, *IEEE Trans. Inform. Theory* 46 (2000) 325-343.
- [2] A. Al-Bashabsheh, Y. Mao, Normal factor graphs and holographic transformations, *IEEE Trans. Inform. Theory* 57 (February (2)) (2011) 752-763.
- [3] A. Al-Bashabsheh, Y. Mao, Normal Factor Graphs as Probabilistic Models, 2012, arXiv:1209.3300v1 [cs.IT] 14 September 2012.
- [4] U. Bertele, F. Brioschi, *Nonserial Dynamic Programming*, Academic Press, Boston, 1972.
- [5] J. Besag, Spatial interaction and the statistical analysis of lattice systems, *J. R. Stat. Soc. B* 36 (2) (1974) 192-236.
- [6] C.E. Cannings, A. Thompson, M.H. Skolnick, The recursive derivation of likelihoods on complex pedigrees, *Adv. Appl. Probab.* 8 (4) (1976) 622-625.
- [7] R. Chellappa, R.L. Kashyap, Digital image restoration using spatial interaction models, *IEEE Trans. Acoust. Speech Signal Process.* 30 (1982) 461-472.
- [8] R. Chellappa, S. Chatterjee, Classification of textures using Gaussian Markov random field models, *IEEE Trans. Acoust. Speech Signal Process.* 33 (1985) 959-963.
- [9] R. Chellappa, A.K. Jain (Eds.), *Markov Random Fields: Theory and Applications*, Academic Press, Boston, 1993.

- [10] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, *Artif. Intell.* 42 (1990) 393-405.
- [11] P. Dagum, M. Luby, Approximating probabilistic inference in Bayesian belief networks is NP-hard, *Artif. Intell.* 60 (1993) 141-153.
- [12] A.P. Dawid, Conditional independence in statistical theory, *J. R. Stat. Soc. B* 41 (1978) 1-31.
- [13] A.P. Dawid, Applications of a general propagation algorithm for probabilistic expert systems, *Stat. Comput.* 2 (1992) 25-36.
- [14] A.P. Dawid, M. Studeny, Conditional products: an alternative approach to conditional independence, in: D. Heckerman, J. Whittaker (Eds.), *Artificial Intelligence and Statistics*, Morgan-Kaufmann, San Mateo, 1999.
- [15] B.J. Frey, *Graphical Models for Machine Learning and Digital Communications*, MIT Press, Cambridge, MA, 1998.
- [16] B.J. Frey, F.R. Kschischang, H.A. Loeliger, N. Wiberg, Factor graphs and algorithms, *Proceedings of the 35th Alerton Conference on Communication, Control and Computing*, 1999.
- [17] G.D. Forney Jr., The Viterbi algorithm, *Proc. IEEE* 61 (1973) 268-277.
- [18] G.D. Forney Jr., Codes on graphs: normal realizations, *IEEE Trans. Inform. Theory* 47 (2001) 520-548.
- [19] G.D. Forney Jr., Codes on graphs: duality and MacWilliams identities, *IEEE Trans. Inform. Theory* 57 (3) (2011) 1382-1397.
- [20] D. Geiger, T. Verma, J. Pearl, d-Separation: From theorems to algorithms, in: M. Henrion, R.D. Shachter, L.N. Kanal, J.F. Lemmer (Eds.), *Proceedings 5th Annual Conference on Uncertainty in Artificial Intelligence*, 1990.
- [21] S. Geman, D. Geman, Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images, *IEEE Trans. Pattern Anal. Mach. Intell.* 6 (1) (1984) 721-741.
- [22] J.M. Hammersley, P. Clifford, Markov fields on finite graphs and lattices, Unpublished manuscript available the web, 1971.
- [23] G.E. Hinton, T. Sejnowski, Learning and relearning in Boltzmann machines, in: D.E. Rumelhart, J.L. McClelland (Eds.), *Parallel Distributed Processing*, vol. 1, MIT Press, Cambridge, MA, 1986.
- [24] D. Janzing, B. Schölkopf, Causal inference using the algorithmic Markov condition, *IEEE Trans. Inform. Theory* 56 (2010) 5168-5194.
- [25] D. Koller, N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press, Cambridge, MA, 2009.
- [26] F.R. Kschischang, B.J. Frey, H.A. Loeliger, Factor graphs and the sum-product algorithm, *IEEE Trans. Inform. Theory* 47(2) (2001) 498-519.
- [27] J. Lafferty, A. McCallum, F. Pereira, Conditional random fields: probabilistic models for segmenting and labeling sequence data, in: *International Conference on Machine Learning*, 2001, pp. 282-289.
- [28] S.L. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, *J. R. Stat. Soc. B* 50 (1988) 157-224.
- [29] S.Z. Li, *Markov Random Field Modeling in Image Analysis*, Springer-Verlag, New York, 2009.
- [30] H.A. Loeliger, J. Dauwels, J. Hu, S. Korl, L. Ping, F.R. Kschischang, The factor graph approach to model-based signal processing, *Proc. IEEE*, 95 (6) (2007) 1295-1322.
- [31] D.J.C. MacKay, *Information Inference and Learning Algorithms*, Cambridge University Press, Cambridge, 2003.
- [32] R.E. Neapolitan, *Learning Bayesian Networks*, Prentice Hall, Upper Saddle River, NJ, 2004.
- [33] R.M. Neal, Connectionist learning of belief networks, *Artif. Intell.* 56 (1992) 71-113.
- [34] F.A.N. Palmieri, Learning nonlinear functions with factor graphs, *IEEE Trans. Signal Process.* 61 (12) (2013) 4360-4371.
- [35] F.A.N. Palmieri, A Comparison of algorithms for learning hidden variables in normal graphs, 2013, arXiv: 1308.5576v1 [stat.ML] 26 August 2013.

- [36] J. Pearl, Fusion, propagation, and structuring in belief networks, *Artif. Intell.* 29 (1986) 241-288.
- [37] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan-Kaufmann, San Mateo, 1988.
- [38] J. Pearl, Causality, Reasoning and Inference, second ed., Cambridge University Press, Cambridge, 2012.
- [39] L. Rabiner, A tutorial on hidden Markov models and selected applications in speech processing, *Proc. IEEE* 77 (1989) 257-286.
- [40] U. Schmidt, Learning and Evaluating Markov Random Fields for Natural Images, Master's Thesis, Department of Computer Science, Technische Universität Darmstadt, Germany, 2010.
- [41] M.A. Shwe, G.F. Cooper, An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief network, *Comput. Biomed. Res.* 24 (1991) 453-475.
- [42] P. Spirtes, Introduction to causal inference, *J. Mach. Learn. Res.* 11 (2010) 1643-1662.
- [43] X. Sun, D. Janzing, B. Schölkopf, Causal inference by choosing graphs with most plausible Markov kernels, in: Proceedings, 9th International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, 2006, pp. 1-11.
- [44] C. Sutton, A. McCallum, An introduction to conditional random fields, 2010, arXiv:1011.4088v1 [stat.ML] 17 November 2010.
- [45] T. Verma, J. Pearl, Causal networks: semantics and expressiveness, in: R.D. Schachter, T.S. Levitt, L.N. Kanal, J.F. Lemmer (Eds.), *Proceedings of the 4th Conference on Uncertainty in Artificial Intelligence*, North-Holland, 1990.
- [46] A.J. Viterbi, Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, *IEEE Trans. Inform. Theory* IT-13 (1967) 260-269.
- [47] J.W. Woods, Two-dimensional discrete Markovian fields, *IEEE Trans. Inform. Theory*, 18 (2) (1972) 232-240.

This page intentionally left blank

PROBABILISTIC GRAPHICAL MODELS: PART II

16

CHAPTER OUTLINE

16.1 Introduction	805
16.2 Triangulated Graphs and Junction Trees	806
16.2.1 Constructing a Join Tree	809
16.2.2 Message-Passing in Junction Trees	811
16.3 Approximate Inference Methods	814
16.3.1 Variational Methods: Local Approximation	814
<i>Multiple-Cause Networks and the Noisy-OR Model</i>	815
<i>The Boltzmann Machine</i>	817
16.3.2 Block Methods for Variational Approximation	819
<i>The Mean Field Approximation and the Boltzmann Machine</i>	820
16.3.3 Loopy Belief Propagation	823
16.4 Dynamic Graphical Models	826
16.5 Hidden Markov Models	828
16.5.1 Inference	831
16.5.2 Learning the Parameters in an HMM	835
16.5.3 Discriminative Learning	838
16.6 BEYOND HMMs: A DISCUSSION	839
16.6.1 Factorial Hidden Markov Models	839
16.6.2 Time-Varying Dynamic Bayesian Networks	842
16.7 Learning Graphical Models	843
16.7.1 Parameter Estimation	843
16.7.2 Learning the Structure	847
Problems	848
References	850

16.1 INTRODUCTION

This is the follow-up to [Chapter 15](#) and it builds upon the notions and models introduced there. The emphasis of this chapter is on more advanced topics for probabilistic graphical models. It wraps up the topic of exact inference in the context of junction trees and then moves on to introduce approximate

inference techniques. This establishes a bridge with [Chapter 13](#). Then, dynamic Bayesian networks are introduced with an emphasis on hidden Markov models (HMM). Inference and training of HMMs is seen as a special case of the message-passing algorithm and the EM scheme discussed in [Chapter 12](#). Finally, the more general concept of training graphical models is briefly discussed.

16.2 TRIANGULATED GRAPHS AND JUNCTION TREES

In [Chapter 15](#), we discussed three efficient schemes for exact inference in graphical entities of a tree structure. Our focus in this section is to present a methodology that can transform an arbitrary graph into an equivalent one having a tree structure. Thus, in principle, such a procedure offers the means for exact inference in arbitrary graphs. This transformation of an arbitrary graph to a tree involves a number of stages. Our goal is to present these stages and explain the procedure more via examples and less via formal mathematical proofs. A more detailed treatment can be obtained from more specialized sources, for example, [32, 45].

We assume that our graph is undirected. Thus, if the original graph was a directed one, then it is assumed that the moralization step has previously been applied.

Definition 16.1. An undirected graph is said to be *triangulated* if and only if for *every* cycle of length greater than three the graph possesses a *chord*. A chord is an edge joining two *nonconsecutive* nodes in the cycle.

In other words, in a triangulated graph, the largest “minimal cycle” is a triangle. [Figure 16.1a](#) shows a graph with a cycle of length $n = 4$ and [Figures 16.1b](#) and [c](#) show two triangulated versions; note that the process of triangulation does not lead to unique answers. [Figure 16.2a](#) is an example of a graph with a cycle of $n = 5$ nodes. [Figure 16.2b](#), although it has an extra edge joining two nonconsecutive nodes, is not triangulated. This is because there still remains a chordless cycle of four nodes ($x_2 - x_3 - x_4 - x_5$). [Figure 16.2c](#) is a triangulated version. There are no cycles of length $n > 3$ without a chord. Note that by joining nonconsecutive edges in order to triangulate a graph, we divide it into cliques ([Section 15.4](#)); we will appreciate this very soon. [Figure 16.1b, c](#) comprise two (three-node) cliques and [Figure 16.2 c](#) comprises three cliques. This is not the case with [Figure 16.2b](#), where the subgraph (x_2, x_3, x_4, x_5) is not a clique.

Let us now see how the previous definition relates to the task of variable elimination, which underlies the message-passing philosophy. In our discussion on such algorithmic schemes, we “just” picked a node and marginalized out the respective variable (e.g., in the sum-product algorithm); as a matter of

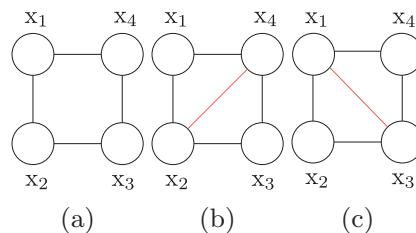
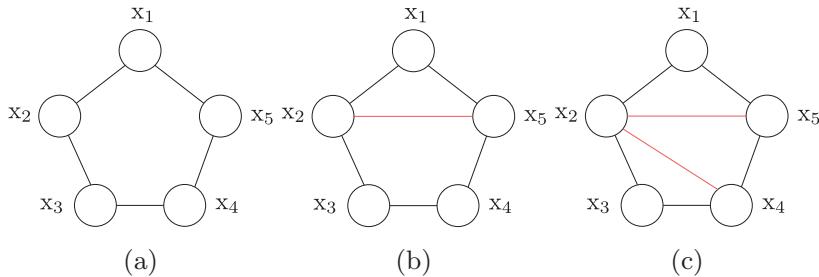
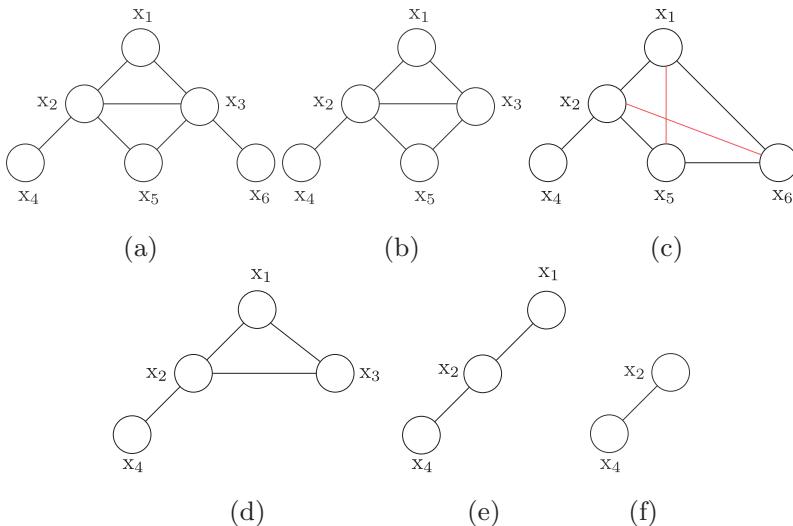


FIGURE 16.1

(a) A graph with a cycle of length $n = 4$. (b), (c) Two possible triangulated versions.


FIGURE 16.2

- (a) A graph of cycle of length $n = 5$. (b) Adding one edge still leaves a cycle of length $n = 4$ chordless.
 (c) A triangulated version; there are no cycles of length $n > 3$ without a chord.


FIGURE 16.3

- (a) An undirected graph with potential (factor) functions $\psi_1(x_1)$, $\psi_2(x_1, x_2)$, $\psi_3(x_1, x_3)$, $\psi_4(x_2, x_4)$, $\psi_5(x_2, x_3, x_5)$, $\psi_6(x_3, x_6)$. (b) The graph resulting after the elimination of x_6 . (c) The graph that would have resulted if the first node to be eliminated were x_3 . Observe the fill-in edges denoted by red. (d), (e), (f) are the graphs that would result if the elimination process had continued from the topology shown in (b) and sequentially removing the nodes: x_5 , x_3 , and finally x_1 .

fact, this is not quite true. The message-passing was initialized at the leaves of the tree graphs; this was done on purpose, although not explicitly stated there. We will soon realize why.

Consider Figure 16.3 and let

$$P(\mathbf{x}) := \psi_1(x_1)\psi_2(x_1, x_2)\psi_3(x_1, x_3)\psi_4(x_2, x_4)\psi_5(x_2, x_3, x_5)\psi_6(x_3, x_6), \quad (16.1)$$

assuming that $Z = 1$.

Let us eliminate x_6 first, that is,

$$\begin{aligned}\sum_{x_6} P(\mathbf{x}) &= \psi^{(1)}(x_1, x_2, x_3, x_4, x_5) \sum_{x_6} \psi_6(x_3, x_6) \\ &= \psi^{(1)}(x_1, x_2, x_3, x_4, x_5) \psi^{(3)}(x_3),\end{aligned}\quad (16.2)$$

where the definitions of $\psi^{(1)}$ and $\psi^{(3)}$ are self-explained, by comparing Eqs. (16.1) and (16.2). The result of elimination is equivalent with a new graph, shown in Figure 16.3b with $P(\mathbf{x})$ given as the product of the same potential functions as before with the exception of ψ_3 , which is now replaced by the product $\psi_3(x_1, x_3)\psi^{(3)}(x_3)$. Basically, $\psi^{(3)}(\cdot)$ is the message passed to x_3 .

In contrast, let us now start by eliminating x_3 first. Then, we have

$$\begin{aligned}\sum_{x_3} P(\mathbf{x}) &= \psi^{(2)}(x_1, x_2, x_4) \sum_{x_3} \psi_3(x_1, x_3) \psi_5(x_2, x_3, x_5) \psi_6(x_3, x_6) \\ &= \psi^{(2)}(x_1, x_2, x_4) \tilde{\psi}^{(3)}(x_1, x_2, x_5, x_6).\end{aligned}$$

Note that this summation is more difficult to perform. It involves four variables (x_1, x_2, x_5, x_6) besides x_3 , which requires many more combination terms be computed than before. Figure 16.3c shows the resulting equivalent graph, after eliminating x_3 . Due to the resulting factor $\tilde{\psi}^{(3)}(x_1, x_2, x_5, x_6)$ new connections implicitly appear, known as *fill-in edges*. This is not a desired situation, as it introduces factors depending on new combinations of variables. Moreover, the new factor depends on four variables, and we know that the larger the number of variables, or the *domain* of the factor as we say, the larger the number of terms involved in the summations, which increases the computational load.

Thus, the choice of the sequence of elimination is very important and far from innocent. For example, for the case of Figure 16.3a an elimination sequence that does not introduce fill-ins is the following: $x_6, x_5, x_3, x_1, x_2, x_4$. For such an elimination sequence, every time a variable is eliminated, the new graph results from the previous one by just removing one node. This is shown by the sequence of graphs in Figures 16.3a, b, d, e, f, for the case of the previously given elimination sequence. An elimination sequence that *does not* introduce fill-ins is known as a *perfect elimination sequence*.

Proposition 16.1. An undirected graph is triangulated if and only if it has a perfect elimination sequence, for example, [32].

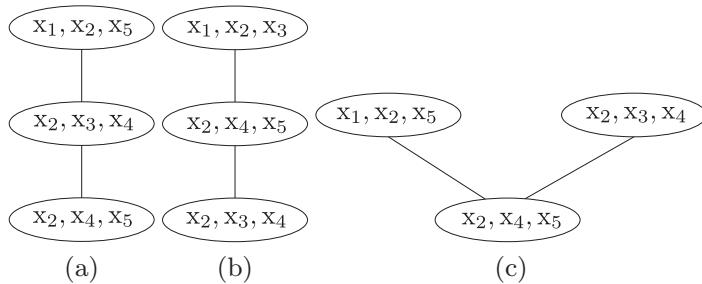
Definition 16.2. A tree, T , is said to be a *join tree* if (a) its nodes correspond to the cliques of an (undirected) graph, G , and (b) the intersection of *any* two nodes, $U \cap V$ is contained in every node in the *unique* path between U and V . The latter property is also known as the *running intersection property*.

Moreover, if a probability distribution p factorizes over G so that each of the product factors (potential functions) is attached to a clique (i.e., depends only on variables associated with the nodes in the clique) then the join tree is said to be a *junction tree* for p [7].

Example 16.1. Consider the triangulated graph of Figure 16.2c. It comprises three cliques, namely (x_1, x_2, x_5) , (x_2, x_3, x_4) , and (x_2, x_4, x_5) . Associating each clique with a node of a tree, Figure 16.4 presents three possibilities. The trees in Figure 16.4a, b are not join trees. Indeed, the intersection $\{x_1, x_2, x_5\} \cap \{x_2, x_4, x_5\} = \{x_2, x_5\}$ does not appear in node (x_2, x_3, x_4) . Similar arguments hold true for the case of Figure 16.4b. In contrast, the tree in Figure 16.4c is a join tree, because the intersection $\{x_1, x_2, x_5\} \cap \{x_2, x_3, x_4\} = \{x_2\}$ is contained in (x_2, x_4, x_5) . If, now, we have a distribution such as

$$p(\mathbf{x}) = \psi_1(x_1, x_2, x_5)\psi_2(x_2, x_3, x_4)\psi_3(x_2, x_4, x_5)$$

the graph in Figure 16.4c is a junction tree for $p(\mathbf{x})$

**FIGURE 16.4**

The graphs resulting from Figure 16.2c and shown in (a) and (b) are not join trees. (c) This is a join tree, because the node in the path from (x_1, x_2, x_5) to (x_2, x_3, x_4) contains their intersection, x_2 .

We are now ready to state the basic theorem of this section; the one that will allow us to transform an arbitrary graph into a graph of a tree structure.

Theorem 16.1. *An undirected graph is triangulated if and only if its cliques can be organized into a join tree (Problem 16.1).*

Once a triangulated graph, which is associated with a factorized probability distribution, $p(\mathbf{x})$, has been transformed into a junction tree, then any of the message-passing algorithms, described in Chapter 15, can be adopted to perform exact inference.

16.2.1 CONSTRUCTING A JOIN TREE

Starting from a triangulated graph, the following algorithmic steps construct a join tree ([32]):

- Select a node in a maximal clique of the triangulated graph, which is not shared by other cliques. Eliminate this node and keep removing nodes from the clique, as long as they are *not* shared by other cliques. Denote the set of the remaining nodes of this clique as S_i , where i is the number of the nodes eliminated so far. This set is called a *separator*. Use V_i to denote the set of all the nodes in the clique, prior to the elimination process.
- Select another maximal clique and repeat the process with the index counting the node elimination starting from i .
- Continue the process until all cliques have been eliminated. Once the previous peeling off procedure has been completed, join together the parts that have resulted, so that each separator, S_i , is joined to V_i on one of its sides and to a clique node (set) V_j , ($j > i$), such that $S_i \subset V_j$. This is in line with the running intersection property. It can be shown that the resulting graph is a join tree (part of proof in Problem 16.1).

An alternative algorithmic path to construct a join tree, once the cliques have been formed, is the following: Build an undirected graph having as nodes the maximal cliques of the triangulated graph. For each pair of linked nodes, V_i, V_j , assign a weight, w_{ij} , on the respective edge equal to the cardinality of $V_i \cap V_j$. Then run the *maximal spanning tree* algorithm (e.g., [43]) to identify a tree in this graph such as the sum of weights is maximal [41]. It turns out that such a procedure guarantees the running intersection property.

Example 16.2. Consider the graph of [Figure 16.5](#), which is described in the seminal paper [44]. Smoking can cause lung cancer or bronchitis. A recent visit to Asia increases the probability of tuberculosis. Both, tuberculosis and cancer can result in a positive X-ray finding. Also, all three diseases can cause difficulty in breathing (dyspnea). In the context of the current example, we are not interested in the values of the respective probability table, and our goal is to construct a join tree, following the previous algorithm. [Figure 16.6](#) shows a triangulated graph that corresponds to [Figure 16.5](#).

The elimination sequence of the nodes in the triangulated graph is graphically illustrated in [Figure 16.7](#). First, node A is eliminated from the clique (A, T) and the respective separator set comprises T . Because only one node can be eliminated ($i = 1$), we indicate the separator as S_1 . Next, node T is eliminated from the clique (T, L, E) and the S_2 ($i = i + 1$) separator comprises L, E . The process

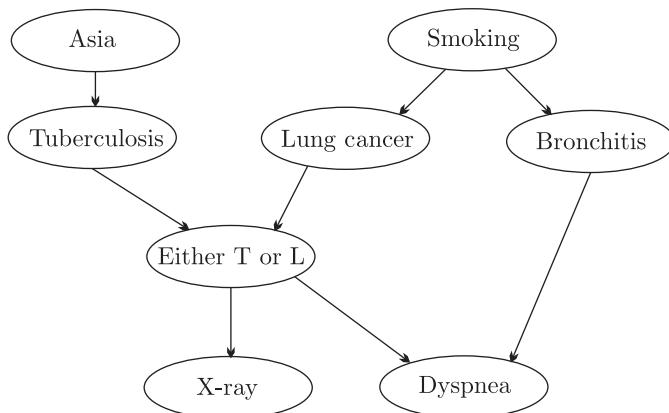


FIGURE 16.5

The Bayesian network structure of the example given in [44].

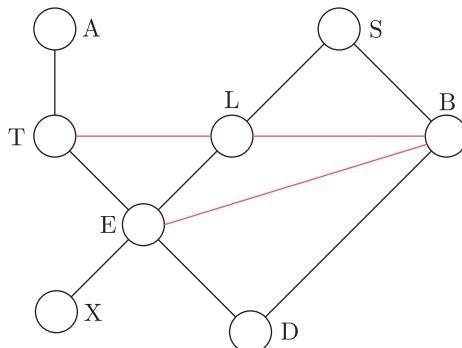
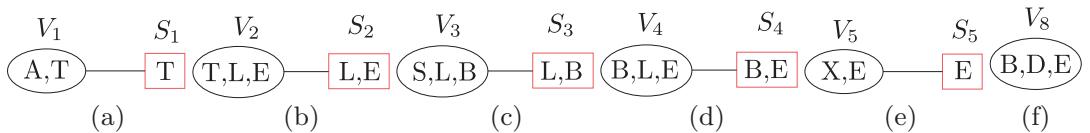
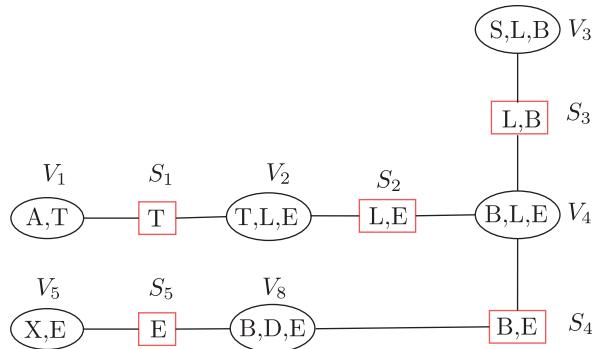


FIGURE 16.6

The graph resulting from the Bayesian network structure of [Figure 16.5](#), after having been moralized and triangulated. The inserted edges are drawn in red.


FIGURE 16.7

The sequence of elimination of nodes from the respective cliques of [Figure 16.6](#) and the resulting separators.


FIGURE 16.8

The resulting join tree from the graph of [Figure 16.6](#). A separator, S_i , is linked to a clique V_j , ($j > i$) and so that $S_i \subset V_j$.

continues till clique (B, D, E) is the only remaining one. It is denoted as V_8 , as all three nodes can be eliminated sequentially (hence, $8 = 5 + 3$); there is no other neighboring clique. [Figure 16.8](#) shows the resulting junction tree. Verify the running intersection property.

16.2.2 MESSAGE-PASSING IN JUNCTION TREES

By its definition, a junction tree is a join tree where we have associated a factor, say ψ_c , of a probability distribution, p , with each one of the cliques. Each factor can be considered as the product of all potential functions, which are defined in terms of the variables associated with the nodes of the corresponding clique; hence, the domain of each one of these potential functions is a subset of the variables-nodes comprising the clique. Then, focusing on the discrete probability case, we can write

$$P(x) = \frac{1}{Z} \prod_c \psi_c(x_c), \quad (16.3)$$

where c runs over the cliques and x_c denotes the variables comprising the respective clique. Because a junction tree is a graph with a tree structure, exact inference can take place in the same way as we have already discussed in [Section 15.7](#), via a message-passing rationale. A two-way message-passing is also required here. There are, however, some small differences. In the case of the factor graphs, which

we have considered previously, the exchanged messages were functions of one variable. This is not necessarily the case here. Moreover, in this case, after the bidirectional flow of the messages has been completed, what is recovered from each node of the junction tree is the *joint probability of the variables associated with the clique*, $P(\mathbf{x}_c)$. The computation of the marginal probabilities for individual variables requires extra summations in order to marginalize with respect to the rest.

Note that in the message-passing, the following take place:

- A separator receives messages and passes their product to one of its connected cliques, depending on the direction of the message-passing flow, that is,

$$\mu_{S \rightarrow V}(\mathbf{x}_S) = \prod_{v \in \mathcal{N}(S) \setminus V} \mu_{v \rightarrow S}(\mathbf{x}_S), \quad (16.4)$$

where $\mathcal{N}(S)$ is the index set of the clique nodes connected to S , and $\mathcal{N}(S) \setminus V$ is this set excluding the index for clique node V . Note that the message is a function of the variables comprising the separator.

- Each clique node performs marginalization and passes the message to each one of its connected separators, depending on the direction of the flow. Let V be a clique node and \mathbf{x}_V the vector of the involved variables in it, and let S be a separator node connected to it. The message passed to S is given by

$$\mu_{V \rightarrow S}(\mathbf{x}_S) = \sum_{\mathbf{x}_V \setminus \mathbf{x}_S} \psi_V(\mathbf{x}_V) \prod_{s \in \mathcal{N}(V) \setminus S} \mu_{s \rightarrow V}(\mathbf{x}_s). \quad (16.5)$$

By \mathbf{x}_S we denote the variables in the separator S . Obviously, $\mathbf{x}_S \subset \mathbf{x}_V$ and $\mathbf{x}_V \setminus \mathbf{x}_S$ denotes all variables in \mathbf{x}_V excluding those in \mathbf{x}_S . $\mathcal{N}(V)$ is the index set of all separators connected to V and \mathbf{x}_s , the set of variables in the respective separator ($\mathbf{x}_s \subset \mathbf{x}_V$, $s \in \mathcal{N}(V)$). $\mathcal{N}(V) \setminus S$ denotes the index set of all separators connected to V excluding S . This is basically the counterpart of Eq. (15.50). Figure 16.9 shows the respective configuration.

Once the two-way message-passing has been completed, marginals in the clique as well as the separator nodes are computed as (Problem 16.3)

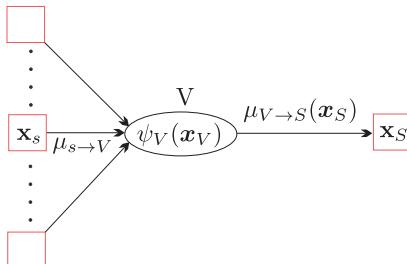


FIGURE 16.9

Clique node V “collects” all incoming messages from the separators it is connected with (except S); then, it outputs a message to S , after the marginalization performed on the product of $\psi_V(\mathbf{x}_V)$ with the incoming messages.

- Clique nodes:

$$P(\mathbf{x}_V) = \frac{1}{Z} \psi_V(\mathbf{x}_V) \prod_{s \in \mathcal{N}(V)} \mu_{s \rightarrow V}(\mathbf{x}_s) \quad (16.6)$$

- Separator nodes: Each separator is connected only to clique nodes. After the two-way message-passing, every separator has received messages from both flow directions. Then, it is shown that

$$P(\mathbf{x}_S) = \frac{1}{Z} \prod_{v \in \mathcal{N}(S)} \mu_{v \rightarrow S}(\mathbf{x}_S). \quad (16.7)$$

An important by-product of the previous message-passing algorithm in junction trees concerns the joint distribution of all the involved variables, which turns out to be independent of Z , and it is given by (Problem 16.4)

$$P(\mathbf{x}) = \frac{\prod_v P_v(\mathbf{x}_v)}{\prod_s [P_s(\mathbf{x}_s)]^{d_s-1}}, \quad (16.8)$$

where \prod_v and \prod_s run over the sets of clique nodes and separators, respectively, and d_s is the number of the cliques separator S is connected to.

Example 16.3. Let us consider the junction tree of Figure 16.8. Assume that $\psi_1(A, T)$, $\psi_2(T, L, E)$, $\psi_3(S, L, B)$, $\psi_4(B, L, E)$, $\psi_5(X, E)$, and $\psi_6(B, D, E)$ are known. For example,

$$\psi_1(A, T) = P(T|A)P(A),$$

and

$$\psi_3(S, L, B) = P(L|S)P(B|S)P(S).$$

The message-passing can start from the leaves, (A, T) and (X, E) , toward (S, L, B) ; once this message flow has been completed, message-passing takes place in the reverse direction. Some examples of message computations are given below.

The message received by node (T, L, E) is equal to

$$\mu_{S_1 \rightarrow V_2}(T) = \sum_A \psi_1(A, T).$$

Also,

$$\mu_{V_2 \rightarrow S_2}(L, E) = \sum_T \psi_2(T, L, E) \mu_{S_1 \rightarrow V_2}(T) = \mu_{S_2 \rightarrow V_4}(L, E),$$

and

$$\mu_{V_4 \rightarrow S_3}(L, B) = \sum_E \psi_4(B, L, E) \mu_{S_2 \rightarrow V_4}(L, E) \mu_{S_4 \rightarrow V_4}(B, E).$$

The rest of the messages are computed in a similar way.

For the marginal probability, $P(T, L, E)$, of the variables in clique node V_2 , we get

$$P(T, L, E) = \psi_{V_2}(T, L, E) \mu_{S_2 \rightarrow V_2}(L, E) \mu_{S_1 \rightarrow V_2}(T)$$

Observe that in this product, all other variables, besides T , L , and E , have been marginalized out. Also,

$$P(L, E) = \mu_{V_4 \rightarrow S_2}(L, E)\mu_{V_2 \rightarrow S_2}(L, E).$$

Remarks 16.1.

Note that a variable is part of more than one node in the tree. Hence, if one is interested in obtaining the marginal probability of an individual variable, this can be obtained by marginalizing over different variables in different nodes. The properties of the junction tree guarantee that all of them give the same result ([Problem 16.5](#)).

We have already commented that there is not a unique way to triangulate a graph. A natural question is now raised: Are all the triangulated versions equivalent from a computational point of view? Unfortunately, the answer is No. Let us consider the simple case where all the variables have the same number of possible states, k . Then the number of probability values for each clique node depends on the number of variables involved in it, and we know that this dependence is of an exponential form. Thus, our goal while triangulating a graph should be to implement it in such a way that the resulting cliques are as small as possible with respect to the number of nodes-variables involved. Let us define the size of a clique, V_i , as $s_i = k^{n_i}$, where n_i denotes the number of nodes comprising the clique. Ideally, we should aim at obtaining a triangulated version (or equivalently an elimination sequence) so that the total size of the triangulated graph, $\sum_i s_i$, where i runs over all cliques, to be minimum. Unfortunately, this is an NP-hard task [1]. One of the earliest algorithms proposed to obtain low-size triangulated graphs is given in [71]. A survey of related algorithms is provided in [39].

16.3 APPROXIMATE INFERENCE METHODS

So far, our focus has been to present efficient algorithms for exact inference in graphical models. Although such schemes form the basis of inference and have been applied in a number of applications, often one encounters tasks where exact inference is not practically possible. At the end of the previous section, we discussed the importance of small-sized cliques. However, in a number of cases, the graphical model may be so densely connected that it renders the task of obtaining cliques of a small size impossible. We will soon consider some examples.

In such cases, resorting to methods for tractable approximate inference is the only viable alternative. Obviously, there are various paths to approach this problem and a number of techniques have been proposed. Our goal in this section is to discuss the main directions that are currently popular. Our approach will be more on the descriptive side than that of rigorous mathematical proofs and theorems. The reader who is interested in delving deeper into this topic can refer to more specialized references, which are given in the text below.

16.3.1 VARIATIONAL METHODS: LOCAL APPROXIMATION

The current and the next subsections draw a lot of their theoretical basis on the variational approximation methods, which were introduced in [Chapter 13](#) and in particular [Sections 13.2](#) and [13.8](#).

The main goal in variational approximation methods is to replace probability distributions with computationally attractive bounds. The effect of such *deterministic* approximation methods is that it simplifies the computations; as we will soon see, this is equivalent to simplifying the graphical

structure. Yet, these simplifications are carried out in the context of an associated optimization process. The functional form of these bounds is very much problem-dependent, so we will demonstrate the methodology via some selected examples.

Two main directions are followed: the *sequential* one and the *block* one [34]. The former will be treated in this subsection and the latter in the next one.

In the sequential methods, the approximation is imposed on individual nodes in order to modify the functional form of the local probability distribution functions. This is the reason we called them *local methods*. One can impose the approximation on some of the nodes or to all of them. Usually, some of the nodes are selected, whose number is sufficient so that exact inference can take place with the remaining ones, within practically acceptable computational time and memory size. An alternative viewpoint is to look at the method as a sparsification procedure that removes nodes so as to transform the original graph to a “computationally” manageable one. There are different scenarios on how to select nodes. One way is to introduce approximation to one node at a time until a sufficiently simplified structure occurs. The other way is to introduce approximation to all the nodes and then reinstate the exact distributions one node at a time. The latter of the two has the advantage that the network is computationally tractable all the way; see, for example, [30]. Local approximations are inspired by the method of bounding convex/concave functions in terms of their conjugate ones, as discussed in [Section 13.8](#). Let us now unveil the secrets behind the method.

Multiple-cause networks and the noisy-OR model

In the beginning of [Chapter 15](#) ([Section 15.2](#)) we presented a simplified case from the medical diagnosis field, concerning a set of diseases and findings. Adopting the so called noisy-OR model, we arrived at Eqs. (15.4) and (15.5), which are repeated here for convenience.

$$P(f_i = 0 | \mathbf{d}) = \exp \left(- \sum_{j \in \text{Pa}_i} \theta_{ij} d_j \right), \quad (16.9)$$

$$P(f_i = 1 | \mathbf{d}) = 1 - \exp \left(- \sum_{j \in \text{Pa}_i} \theta_{ij} d_j \right), \quad (16.10)$$

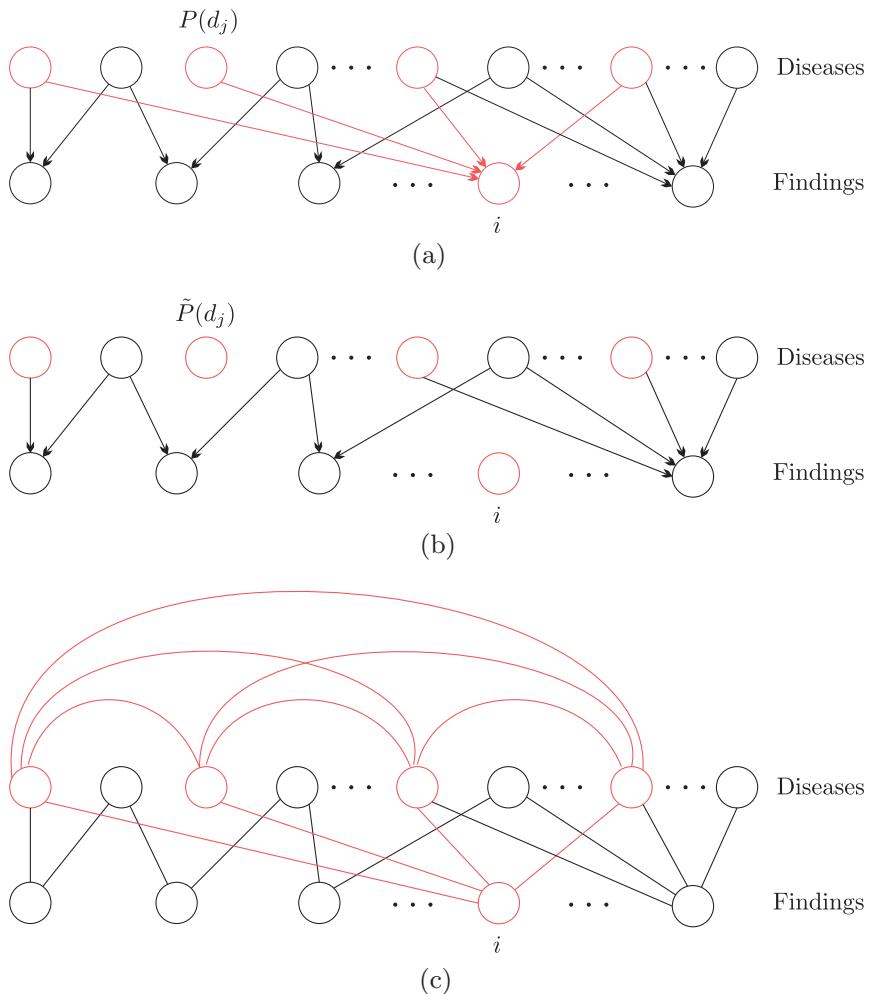
where we have exploited the experience we have gained so far and we have introduced in the notation the set of the parents Pa_i of the i th finding. The respective graphical model belongs to the family of multiple-cause networks ([Section 15.3.6](#)) and it is shown in [Figure 16.10a](#). We will now pick up a specific node, say the i th node, assume that it corresponds to a positive finding ($f_i = 1$), and demonstrate how the variational approximation method can offer a way out from the “curse” of the exponential dependence of the joint probability on the number of the involved terms; recall that this is caused by the form of Eq. (16.10).

Derivation of the Variational Bound: The function $1 - \exp(-x)$ belongs to the so-called *log-concave* family of functions, meaning that

$$f(x) = \ln(1 - \exp(-x)), \quad x > 0,$$

is concave ([Problem 16.9](#)). Being a concave function, we know from [Section 13.8](#) that it is upper bounded by

$$f(x) \leq \xi x - f^*(\xi),$$

**FIGURE 16.10**

(a) A Bayesian network for a set of findings and diseases. The node associated with the i th finding, together with its parents and respective edges, are shown in red; it is the node on which variational approximation is introduced. (b) After the variational approximation is performed for node i , the edges joining it with its parents are removed. At the same time, the prior probabilities of the respective parent nodes change values. (c) The graph that would have resulted after the moralization step, focusing on node, i .

where $f^*(\xi)$ is its conjugate function. Tailoring it to the needs of Eq. (16.10) and using ξ_i in place of ξ , to explicitly indicate the dependence on the node i , we obtain

$$P(f_i = 1 | \mathbf{d}) \leq \exp \left(\xi_i \left(\sum_{j \in \text{Pa}_i} \theta_{ij} d_j \right) - f^*(\xi_i) \right), \quad (16.11)$$

or

$$P(f_i = 1 | \mathbf{d}) \leq \exp \left(-f^*(\xi_i) \right) \prod_{j \in \text{Pa}_i} \left(\exp(\xi_i \theta_{ij}) \right)^{d_j}, \quad (16.12)$$

where (Problem 16.10)

$$f^*(\xi_i) = -\xi_i \ln(\xi_i) + (\xi_i + 1) \ln(\xi_i + 1), \quad \xi_i > 0.$$

Note that usually, a constant θ_{i0} is also present in the linear terms ($\sum_{j \in \text{Pa}_i} \theta_{ij} d_j + \theta_{i0}$) and in this case the first exponent in the upper bound becomes $\exp(-f^*(\xi_i) + \xi_i \theta_{i0})$.

Let us now observe Eq. (16.12). The first factor on the right-hand side is a constant, once ξ_i is determined. Moreover, each one of the factors, $\exp(\xi_i \theta_{ij})$, is also a constant raised in d_j . Thus, substituting Eq. (16.12) in the products in Eq. (15.1), in order to compute, for example, Eq. (15.3), each one of these constants can be absorbed by the respective $P(d_j)$, that is,

$$\tilde{P}(d_j) \propto P(d_j) \exp(\xi_i \theta_{ij} d_j), \quad j \in \text{Pa}_i.$$

Basically, from a graphical point of view, we can equivalently consider that the i th node is delinked and its influence on any subsequent processing is via the modified factors associated with its parent nodes, see Figure 16.10b. In other words, the variational approximation *decouples* the parent nodes. In contrast, for exact inference, during the moralization stage, all parents of node i are connected. This is the source of computational explosion, see Figure 16.10c. The idea is to remove a sufficient number of nodes, so that the remaining network can be handled using exact inference methods.

There is still a main point to be addressed: how the various ξ_i s are obtained. These are computed to make the bound as tight as possible, and any standard optimization technique can be used. Note that this minimization corresponds to a convex cost function (Problem 16.11). Besides the upper bound, a lower bound can also be derived [27]. Experiments performed in [30] verify that reasonably good accuracies can be obtained in affordable computational times. The method was first proposed in [27].

The Boltzmann machine

The Boltzmann machine, which was introduced in Subsection 15.4.2, is another example where any attempt for exact inference is confronted with cliques of sizes that make the task computationally intractable [28].

We will demonstrate the use of the variational approximation in the context of the computation of the normalizing constant Z . Recall from Eq. (15.23) that

$$\begin{aligned} Z &= \sum_{\mathbf{x}} \exp \left(- \sum_i \left(\sum_{j>i} \theta_{ij} x_i x_j + \theta_{i0} x_i \right) \right) \\ &= \sum_{\mathbf{x} \setminus x_k} \sum_{x_k=0}^1 \exp \left(- \sum_i \left(\sum_{j>i} \theta_{ij} x_i x_j + \theta_{i0} x_i \right) \right), \end{aligned} \quad (16.13)$$

where we chose node x_k to impose variational approximation. We split the summation into two, one with regard to x_k and one with regard to the rest of the variables; $\mathbf{x} \setminus x_k$ denotes summation over all variables excluding x_k . Performing the inner sum in Eq. (16.13) (terms different to x_k and $x_k = 0, x_k = 1$), we get

$$Z = \sum_{\mathbf{x} \setminus x_k} \exp \left(- \sum_{i \neq k} \left(\sum_{i < j \neq k} \theta_{ij} x_i x_j + \theta_{i0} x_i \right) \right) \left(1 + \exp \left(- \sum_{i \neq k} \theta_{ki} x_i - \theta_{k0} \right) \right)$$

where $i < j \neq k$ indicates that both i and j are different to k .

Derivation of the Variational Bound: The function $1 + \exp(-x)$, $x \in \mathbb{R}$ is *log-convex* (Problem 16.12); thus, following similar arguments to those adopted for the derivation of the bound in Eq. (16.12), but for convex instead of concave functions, we obtain

$$\begin{aligned} Z &\geq \sum_{\mathbf{x} \setminus x_k} \exp \left(- \sum_{i \neq k} \left(\sum_{i < j \neq k} \theta_{ij} x_i x_j + \theta_{i0} x_i \right) \right) \\ &\quad \times \exp \left(\xi_k \left(\sum_{i \neq k} \theta_{ki} x_i + \theta_{k0} \right) - f^*(\xi_k) \right), \end{aligned} \quad (16.14)$$

where $f^*(\xi)$ is the respective conjugate function (Problem 16.12). Note that the second exponential in the bound can be combined with the first one and we can write

$$Z \geq \exp(-f^*(\xi_k) + \xi_k \theta_{k0}) \sum_{\mathbf{x} \setminus x_k} \exp \left(- \sum_{i \neq k} \left(\sum_{i < j \neq k} \tilde{\theta}_{ij} x_i x_j + \tilde{\theta}_{i0} x_i \right) \right),$$

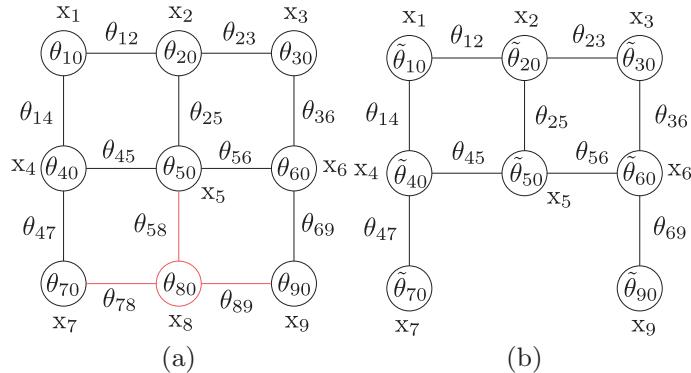
where

$$\tilde{\theta}_{ij} = \theta_{ij}, \quad i, j \neq k,$$

and

$$\tilde{\theta}_{i0} = \theta_{i0} - \xi_k \theta_{ki}, \quad i \neq k.$$

In other words, if from now on we replace Z with the bound, it is as if node x_k has been removed and the remaining network is a Boltzmann machine with one node less and the respective parameters modified, compared to the original ones. The value of ξ_k can be obtained via optimization so as to make the bound as tight as possible.

**FIGURE 16.11**

(a) An MRF corresponding to a Boltzmann machine. (b) The resulting MRF after removing x_8 via variational approximation. Note that if x_5 is removed next, then the remaining graphical structure becomes a chain.

Figure 16.11 illustrates the effect of applying variational approximation to the node x_8 . For the case of this figure, note that if x_5 is removed next (after x_8) the remaining graphical structure becomes a chain and exact inference can be carried out.

Following exactly similar arguments and employing the conjugate of the sigmoid function (Problem 13.18), one can apply the technique to the sigmoidal Bayesian networks, discussed in Section 15.3.4; see also [27].

16.3.2 BLOCK METHODS FOR VARIATIONAL APPROXIMATION

In contrast to the previous approach, where the approximation is introduced on selected nodes individually, here the approximation is imposed on a set of nodes. Once more, a derived bound of the involved probability distribution is optimized. In principle, the method is equivalent to imposing a specific graphical structure on the nodes, which can then be addressed by tractable exact inference techniques. In the sequel, the family of distributions, which can be factorized over this simplified substructure, is optimized with respect to a set of variational parameters. The method builds upon the same arguments as those used in Section 13.2. We will retain the same notation and we will provide the related formulas for the discrete variable case, to be used later on in our selected examples.

Let \mathcal{X} be the set of observed and \mathcal{X}^l the set of latent random variables associated with the nodes of a graphical structure; in the graphical model terminology we can refer to them as evidence and hidden nodes, respectively. Define

$$\mathcal{F}(Q) = \sum_{x \in \mathcal{X}^l} Q(\mathcal{X}^l) \ln \frac{P(\mathcal{X}, \mathcal{X}^l)}{Q(\mathcal{X}^l)}, \quad (16.15)$$

where Q is any probability function. Then, from Eq. (16.15), we readily obtain that

$$\mathcal{F}(Q) = \ln P(\mathcal{X}) + \sum_{x \in \mathcal{X}^l} Q(\mathcal{X}^l) \ln \frac{P(\mathcal{X}^l | \mathcal{X})}{Q(\mathcal{X}^l)},$$

or

$$\ln P(\mathcal{X}) = \mathcal{F}(Q) + \sum_{x \in \mathcal{X}^l} Q(x^l) \ln \frac{Q(x^l)}{P(x^l | \mathcal{X})}. \quad (16.16)$$

Note that the second term in Eq. (16.16) is the Kullback-Leibler divergence between $P(\mathcal{X}^l | \mathcal{X})$ and $Q(\mathcal{X}^l)$. Because KL divergence is always nonnegative (Problem 12.12), we can write

$$\ln P(\mathcal{X}) \geq \mathcal{F}(Q),$$

and the lower bound is maximized if we minimize the KL divergence.

Let us now return to our goal; that is, given the evidence, to perform inference on the graph associated with $P(\mathcal{X}^l | \mathcal{X})$. If this cannot be performed in a tractable way, the method adopts an approximation, $Q(\mathcal{X}^l)$, of $P(\mathcal{X}^l | \mathcal{X})$ and at the same time imposes a specific factorization on $Q(\mathcal{X}^l)$ (which equivalently induces a specific graphical structure) so that exact inference techniques can be employed. From the adopted family of distributions, we choose the one that minimizes the KL divergence between $P(\mathcal{X}^l | \mathcal{X})$ and $Q(\mathcal{X}^l)$. Such a choice guarantees the maximum lower bound for the log-evidence function. Among the different ways of factorization, the so-called *mean field* factorization is the simplest and, possibly, the most popular. The imposed structure on the graph has no edges, which leads to a complete factorization of $Q(\mathcal{X}^l)$, that is,

$$Q(\mathcal{X}^l) = \prod_{i: x_i \in \mathcal{X}^l} Q_i(x_i) : \text{ Mean Field Factorization.}$$

(16.17)

The mean field approximation and the Boltzmann machine

As we already know, the joint probability for the Boltzmann machine is given by

$$P(\mathcal{X}, \mathcal{X}^l) = \frac{1}{Z} \exp \left(- \sum_i \left(\sum_{j > i} \theta_{ij} x_i x_j + \theta_{i0} x_i \right) \right),$$

where some of x_i (x_j) belong to \mathcal{X} and some to \mathcal{X}^l . Our first goal is to compute $P(\mathcal{X}^l | \mathcal{X})$ so as to use it in the KL divergence. Note that if both $x_i, x_j \in \mathcal{X}$ their contribution results to a constant, which is finally absorbed by the normalizing factor Z . If one is observed and the other one is latent, then the product contribution becomes linear with regard to the hidden variable and it is absorbed by the respective linear term. Then we can write that

$$P(\mathcal{X}^l | \mathcal{X}) = \frac{1}{\tilde{Z}} \exp \left(- \sum_{i: x_i \in \mathcal{X}^l} \left(\sum_{x_j \in \mathcal{X}^l: j > i} \theta_{ij} x_i x_j + \tilde{\theta}_{i0} x_i \right) \right). \quad (16.18)$$

We now turn our attention into the form of $Q(\cdot)$. Due to the (assumed) binary nature of the variables, a sensible completely factorized form of Q is ([34])

$$Q(\mathcal{X}^l; \boldsymbol{\mu}) = \prod_{i: x_i \in \mathcal{X}^l} \mu_i^{x_i} (1 - \mu_i)^{1-x_i}, \quad (16.19)$$

where the dependence on the variational parameters, $\boldsymbol{\mu}$, is explicitly shown. Also, due to the adopted Bernoulli distribution for each variable, $\mathbb{E}[x_i] = \mu_i$ (Chapter 2). The goal now is to optimize the KL divergence with respect to the variational parameters. Plugging Eqs. (16.18) and (16.19) into

$$KL(Q||P) = \sum_{x_i \in \mathcal{X}^l} Q(\mathcal{X}^l; \boldsymbol{\mu}) \ln \frac{Q(\mathcal{X}^l; \boldsymbol{\mu})}{P(\mathcal{X}^l | \mathcal{X})}, \quad (16.20)$$

we obtain (Problem 16.13, [34]),

$$KL(Q||P) = \sum_i \left(\mu_i \ln \mu_i + (1 - \mu_i) \ln (1 - \mu_i) + \sum_{j>i} \theta_{ij} \mu_i \mu_j + \tilde{\theta}_{i0} \mu_i \right) + \ln \tilde{Z},$$

whose minimization with regard to μ_i finally results in (Problem 16.13)

$$\mu_i = \sigma \left(- \left(\sum_{j \neq i} \theta_{ij} \mu_j + \tilde{\theta}_{i0} \right) \right) : \text{ Mean Field Equations,}$$

(16.21)

where $\sigma(\cdot)$ is the sigmoid link function; recall from the definition of the Ising model that $\theta_{ij} = \theta_{ji} \neq 0$, if x_i and x_j are connected and it is zero otherwise. Plugging the values μ_i into Eq. (16.19), an approximation of $P(\mathcal{X}^l | \mathcal{X})$ in terms of $Q(\mathcal{X}^l; \boldsymbol{\mu})$ has been obtained.

Equation (16.21) is equivalent to a set of coupled equations known as *mean field equations* and they are used in a recursive manner to compute a solution fixed point set, assuming that one exists. Eq. (16.21) is quite interesting. Although we assumed independence among hidden nodes, imposing minimization of the KL divergence, information related to the (true) mutually dependent nature of the variables (as this is conveyed by $P(\mathcal{X}^l | \mathcal{X})$) is “embedded” into the mean values with respect to $Q(\mathcal{X}^l | \boldsymbol{\mu})$; the mean values of the respective variables are *interrelated*. Eq. (16.21) can also be viewed as a message-passing algorithm, see Figure 16.12. Figure 16.13 shows the graph associated with a Boltzmann machine prior to and after the application of the mean field approximation.

Note that what we have said before is nothing but an instance of the variational EM algorithm, presented in Section 13.2; as a matter of fact, Eq. (16.21) is the outcome of the E-step for each one of the factors of Q_i , assuming the rest are fixed.

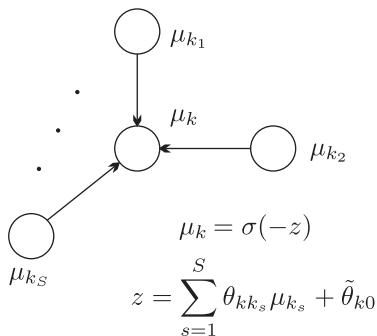
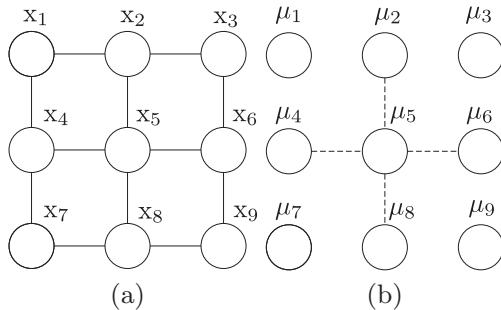


FIGURE 16.12

Node k is connected to S nodes and receives messages from its neighbors; then, passes messages to its neighbors.

**FIGURE 16.13**

(a) The nodes of the graph representing a Boltzmann machine. (b) The mean field approximation results in a graph without edges. The dotted lines indicate the deterministic relation that is imposed among nodes, which were linked prior to the approximation with node x_5 .

Thus far in the chapter, we have not mentioned the important task of how to obtain estimates of the parameters describing a graphical structure; in our current context, these are the parameters θ_{ij} and θ_{i0} , comprising the set $\boldsymbol{\theta}$. Although the parameter estimation task is discussed at the end of this chapter, there is no harm in saying a few words at this point. Let us give the dependence of $\boldsymbol{\theta}$ explicitly and denote the involved probabilities as $Q(\mathcal{X}^l; \boldsymbol{\mu}, \boldsymbol{\theta})$, $P(\mathcal{X}, \mathcal{X}^l; \boldsymbol{\theta})$, and $P(\mathcal{X}^l | \mathcal{X}; \boldsymbol{\theta})$. Treating $\boldsymbol{\theta}$ as an unknown parameter vector, we know from the variational EM, that this can be iteratively estimated by adding the M-step in the algorithm and optimizing the lower bound, $\mathcal{F}(Q)$, with regard to $\boldsymbol{\theta}$, fixing the rest of the involved parameters; see, for example, [29, 69].

Remarks 16.2.

- The mean field approximation method has also been applied in the case of sigmoidal neural networks, defined in [Section 15.3.4](#) (see, e.g., [69]).
- The mean field approximation involving the completely factorized form of Q is the simplest and crudest approximation. More sophisticated attempts have also been suggested, where Q is allowed to have a richer structure while retaining its computational tractability (see, e.g., [17, 31, 80]).
- In [82], the mean field approximation has been applied to a general Bayesian network, where, as we know, the joint probability distribution is given by the product of the conditionals across the nodes,

$$p(\mathbf{x}) = \prod_i p(x_i | \text{Pa}_i).$$

Unless the conditionals are given in a structurally simple form, exact message-passing can become computationally tough. For such cases, the mean field approximation can be introduced in the hidden variables.

$$Q(\mathcal{X}^l) = \prod_{i: x_i \in \mathcal{X}^l} Q_i(x_i),$$

which are then estimated so as to maximize the lower bound $\mathcal{F}(Q)$ in [\(16.15\)](#). Following the arguments that were introduced in [Section 13.2](#), this is achieved iteratively starting from some

initial estimates and at each iteration step optimization takes place with respect to a single factor, holding the rest fixed. At the $(j + 1)$ step, the m th factor is obtained as (Eq. (13.14))

$$\begin{aligned}\ln Q_m^{(j+1)}(x_m^l) &= \mathbb{E} \left[\ln \prod_i p(x_i | \text{Pa}_i) \right] + \text{constant} \\ &= \mathbb{E} \left[\sum_i \ln p(x_i | \text{Pa}_i) \right] + \text{constant},\end{aligned}$$

where the expectation is with respect to the currently available estimates of the factors, *excluding* Q_m , and x_m^l is the respective hidden variable. When restricting the conditionals within the conjugate-exponential family, the computations of expectations of the logarithms become tractable. The resulting scheme is equivalent to a message-passing algorithm, known as *variational message-passing*, and it comprises passing moments and parameters associated with the exponential distributions. An implementation example of the variational message-passing scheme in the context of MIMO-OFDM communication systems is given in [6, 23, 38].

16.3.3 LOOPY BELIEF PROPAGATION

The message-passing algorithms, which were considered previously for exact inference in graphs with a tree structure, can also be used for approximate inference in general graphs with cycles (loops). Such schemes are known as *loopy belief propagation* algorithms.

The idea of using the message-passing (sum-product) algorithm with graphs with cycles goes back to Pearl [57]. Note that algorithmically, there is nothing to prevent us from applying the algorithm in such general structures. On the other hand, if we do it, there is no guarantee that the algorithm will converge in two passes and, more important, that it will recover the true values for the marginals. As a matter of fact, there is no guarantee that such a message propagation will ever converge. Thus, without any clear theoretical understanding, the idea of using the algorithm in general graphs was rather forgotten. Interestingly enough, the spark for its comeback was ignited by a breakthrough in coding theory, under the name *turbo codes* [5]. It was empirically verified that the scheme can achieve performance very close to the theoretical Shannon limit.

Although, in the beginning, such coding schemes seemed to be unrelated to belief propagation, it was subsequently shown, [49], that the turbo decoding is just an instance of the sum-product algorithm, when applied to a graphical structure that represents the turbo code. As an example of the use of the loopy belief propagation for decoding, consider the case of Figure 15.19. This is a graph with cycles. Applying belief propagation on this graph, we can obtain after convergence the conditional probabilities $P(x_i | y_i)$ and, hence, decide on the received sequence of bits. This finding revived interest in loopy belief propagation; after all, it “may be useful” in practice. Moreover, it initiated activity on theoretical research in order to understand its performance as well as its more general convergence properties.

In [83], it is shown, on the basis of pairwise connected MRF’s (undirected graphical models with potential functions involving at most pairs of variables, e.g., trees), that whenever the sum-product algorithm converges on loopy graphs, the fixed points of the message-passing algorithm are actually stationary points of the so-called *Bethe free energy* cost. This is directly related to the KL divergence, between the true and an approximating distribution (Section 12.5.2). Recall from Eq. (16.20) that one of

the terms in KL divergence is the negative entropy associated with Q . In the mean field approximation, this entropy term can be easily computed. However, this is not the case for more general structures with cycles and one has to be content to settle for an approximation. The so-called Bethe entropy approximation is employed, which in turn gives rise to the Bethe free energy cost function. To obtain the Bethe entropy approximation, one “embeds” into the approximating distribution, Q , a structure that is in line with (16.8), which holds true for trees. Indeed, it can be checked out (try it) that for (singly connected) trees, the product in the numerator runs over all pairs of connected nodes in the tree, and let us denote it as $\prod_{(i,j)} P_{ij}(x_i, x_j)$. Also, d_s is equal to the number of nodes that node s is connected with. Thus, we can write the joint probability as

$$P(\mathbf{x}) = \frac{\prod_{(i,j)} P_{ij}(x_i, x_j)}{\prod_s [P_s(x_s)]^{d_s-1}}.$$

Note that nodes that are connected to only one node have no contribution in the denominator. Then, the entropy of the tree, that is,

$$E = -\mathbb{E}[\ln P(\mathbf{x})],$$

can be written as

$$E = - \sum_{(i,j)} \sum_{x_i} \sum_{x_j} P_{ij}(x_i, x_j) \ln P_{ij}(x_i, x_j) + \sum_s (d_s - 1) \sum_{x_s} P_s(x_s) \ln P_s(x_s). \quad (16.22)$$

Thus, this expression for the entropy is exact for trees. However, for more general graphs with cycles, this can only hold approximately true, and it is known as the *Bethe approximation of the entropy*. The closer to a tree a graph is, the better the approximation becomes; see [83] for a concise, related introduction.

It turns out that in the case of trees, the sum-product algorithm leads to the true marginal values because no approximation is involved and minimizing the free energy is equivalent to minimizing the KL divergence. Thus, from this perspective, the sum-product algorithm gets an optimization flavor. In a number of practical cases, the Bethe approximation is accurate enough, which justifies the good performance that is often achieved in practice by the loopy belief algorithm (see, e.g., [52]). The loopy belief propagation algorithm is not guaranteed to converge in graphs with cycles, so one may choose to minimize the Bethe energy cost directly; although such schemes are slower compared to message-passing, they are guaranteed to converge (e.g., [85]).

An alternative interpretation of the sum-product algorithm as an optimization algorithm of an appropriately selected cost function is given in [75, 77]. A unifying framework for exact, as well as approximate, inference is provided in the context of the exponential family of distributions. Both the mean field approximation as well as the loopy belief propagation algorithm are considered and viewed as different ways to approximate a convex set of realizable mean parameters, which are associated with the corresponding distribution. Although we will not proceed in a detailed presentation, we will provide a few “brush strokes,” which are indicative of the main points around which this theory develops. At the same time, this is a good excuse for us to be exposed to an interesting interplay among the notions of convex duality, entropy, cumulant generating function, and mean parameters, in the context of the exponential family.

The general form of a probability distribution in the exponential family is given by (Section 12.4.1),

$$\begin{aligned} p(\mathbf{x}; \boldsymbol{\theta}) &= C \exp \left(\sum_{i \in I} \theta_i u_i(\mathbf{x}) \right) \\ &= \exp \left(\boldsymbol{\theta}^T \mathbf{u}(\mathbf{x}) - A(\boldsymbol{\theta}) \right), \end{aligned}$$

with

$$A(\boldsymbol{\theta}) = -\ln C = \ln \int \exp \left(\boldsymbol{\theta}^T \mathbf{u}(\mathbf{x}) \right) d\mathbf{x},$$

where the integral becomes summation for discrete variables. $A(\boldsymbol{\theta})$ is a convex function and it is known as the *log-partition* or *cumulant generating function* (Problem 16.14, [75, 77]). It turns out that the conjugate function of $A(\boldsymbol{\theta})$, denoted as $A^*(\boldsymbol{\mu})$, is the *negative entropy* function of $p(\mathbf{x}; \boldsymbol{\theta}(\boldsymbol{\mu}))$, where $\boldsymbol{\theta}(\boldsymbol{\mu})$ is the value of $\boldsymbol{\theta}$ where the maximum (in the definition of the conjugate function) occurs given the value of $\boldsymbol{\mu}$; we say that $\boldsymbol{\theta}(\boldsymbol{\mu})$ and $\boldsymbol{\mu}$ are dually coupled (Problem 16.15). Moreover,

$$\mathbb{E}[\mathbf{u}(\mathbf{x})] = \boldsymbol{\mu},$$

where the expectation is with respect to $p(\mathbf{x}; \boldsymbol{\theta}(\boldsymbol{\mu}))$. This is an interesting interpretation of $\boldsymbol{\mu}$ as a mean parameter vector; recall from Section 12.4.1 that these mean parameters define the respective exponential distribution. Then

$$A(\boldsymbol{\theta}) = \max_{\boldsymbol{\mu} \in \mathcal{M}} \left(\boldsymbol{\theta}^T \boldsymbol{\mu} - A^*(\boldsymbol{\mu}) \right). \quad (16.23)$$

\mathcal{M} is the set that guarantees that $A^*(\boldsymbol{\mu})$ is finite, according to the definition of the conjugate function in (13.77). It turns out that in graphs of a tree structure, the sum-product algorithm is an iterative scheme of solving a Lagrangian dual formulation of (16.23), [75, 77]. Moreover, in this case, the set \mathcal{M} , which can be shown to be a convex one, is possible to be characterized explicitly in a straightforward way and the negative entropy $A^*(\boldsymbol{\mu})$ has an explicit form. These properties are no more valid in graphs with cycles. The mean field approximation involves an inner approximation of the set \mathcal{M} ; hence, it restricts optimization to a limited class of distributions, for which the entropy can be recovered exactly. On the other hand, the loopy belief algorithm provides an outer approximation and, hence, enlarges the class of distributions; entropy can only approximately be recovered, which for the case of pairwise MRFs can take the form of the Bethe approximation.

The previously summarized theoretical findings have been generalized to the case of junction trees, where the potential functions involve more than two variables. Such methods involve the so-called *Kikuchi* energy, which is a generalization of the Bethe approximation [77, 84]. Such arguments have their origins in statistical physics [37].

Remarks 16.3.

- Following the success of loopy belief propagation in turbo decoding, further research verified its performance potential in a number of tasks such as low-density parity check codes [15, 47], network diagnostics [48], sensor network applications [24], and multiuser communications [70]. Furthermore, a number of modified versions of the basic scheme have been proposed. In [74], the so-called tree reweighted belief propagation is proposed. In [26], arguments from information geometry are employed and in [78], projection arguments in the context of information geometry are used. More recently, the belief propagation algorithm and the mean field approximation are

proposed to be optimally combined to exploit their respective advantages [65]. A related review can be found in [76]. In a nutshell, this old scheme is still alive and kicking!

- In Section 13.11, the expectation propagation algorithm was discussed in the context of parameter inference. The scheme can also be adopted in the more general framework of graphical models, if the place of parameters is taken by the hidden variables. Graphical models are particularly tailored for this approach because the joint pdf is factorized. It turns out that if the approximate pdf is completely factorized, corresponding to a partially disconnected network, the expectation propagation algorithm turns out to be the loopy belief propagation algorithm [50]. In [51], it is shown that a new family of message-passing algorithms can be obtained by utilizing a generalization of the KL divergence as the optimizing cost. This family encompasses a number of previously developed schemes.
- Besides the approximation techniques that were previously presented, another popular pool of methods is the Markov chain Monte Carlo (MCMC) framework. Such techniques were discussed in Chapter 14; see, for example, [25] and the references therein.

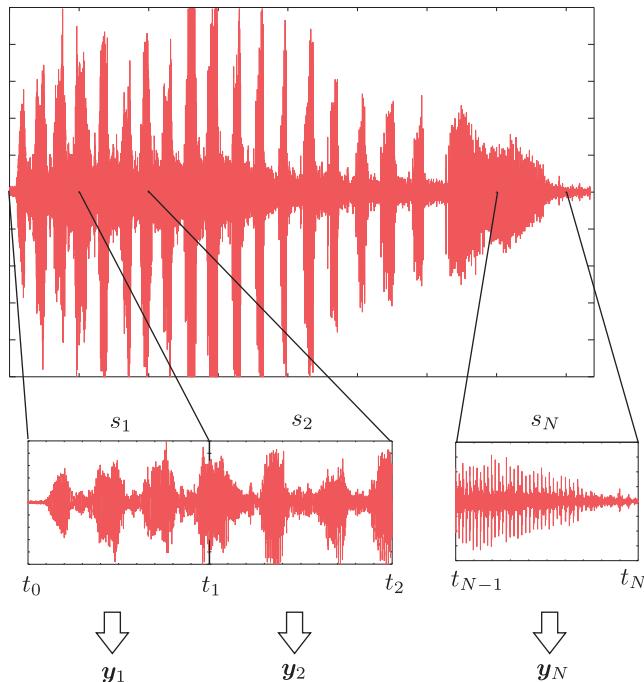
16.4 DYNAMIC GRAPHICAL MODELS

All the graphical models that have been discussed so far were developed to serve the needs of random variables whose statistical properties remained fixed over time. However, this is not always the case. As a matter of fact, the terms *time adaptivity* and *time variation* are central for most parts of this book. Our focus in this section is to deal with random variables whose statistical properties are not fixed but are allowed to undergo changes. A number of time series as well as sequentially obtained data fall under this setting with applications ranging from signal processing and robotics to finance and bioinformatics.

A key difference here, compared to what we have discussed in the previous sections of this chapter, is that now observations are sensed sequentially and the *specific sequence* in which they occur carries important information, which has to be respected and exploited in any subsequent inference task. For example, in speech recognition, the sequence in which the feature vectors result is very important. In a typical speech recognition task, the raw speech data are *sequentially* segmented in short (usually overlapping) time windows and from each window a feature vector is obtained (e.g., DFT of the samples in the respective time slot). This is illustrated in Figure 16.14. These feature vectors constitute the observation sequence. Besides the information that resides in the specific values of these observation vectors, the sequence in which the observations appear discloses important information about the word that is spoken; our language and spoken words are highly structured human activities. Similar arguments hold true for applications such as learning and reasoning concerning biological molecules, for example, DNA and proteins.

Although any type of graphical model has its dynamic counterpart, we will focus on the family of *dynamic Bayesian networks* and, in particular, a specific type known as *hidden Markov models*.

A very popular and effective framework to model sequential data is via the so-called *state-observation* or *state-space* models. Each set of random variables, $\mathbf{y}_n \in \mathbb{R}^l$, which are observed at time n , is associated with a corresponding hidden/latent random vector \mathbf{x}_n (not necessarily of the same dimensionality as that of the observations). The system dynamics are modeled via the latent variables and observations are considered to be the output of a measuring *noisy* sensing device. The so-called *latent Markov models* are built around the following two independence assumptions:

**FIGURE 16.14**

A speech segment and N time windows, each one of length equal to 500 ms. They correspond to time intervals $[0, 500]$, $[500, 1000]$, and $[3500, 4000]$, respectively. From each one of them, a feature vector, \mathbf{y} , is generated. In practice, an overlap between successive windows is allowed.

$$(1) \quad \mathbf{x}_{n+1} \perp (\mathbf{x}_1, \dots, \mathbf{x}_{n-1}) \mid \mathbf{x}_n \quad (16.24)$$

$$(2) \quad \mathbf{y}_n \perp (\mathbf{x}_1, \dots, \mathbf{x}_{n-1}, \mathbf{x}_{n+1}, \dots, \mathbf{x}_N) \mid \mathbf{x}_n, \quad (16.25)$$

where N is the total number of observations. The first condition defines the system dynamics via the transition model

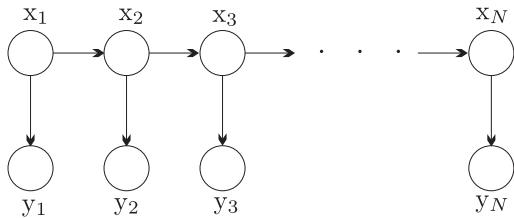
$$p(\mathbf{x}_{n+1} \mid \mathbf{x}_1, \dots, \mathbf{x}_n) = p(\mathbf{x}_{n+1} \mid \mathbf{x}_n), \quad (16.26)$$

and the second one the observation model

$$p(\mathbf{y}_n \mid \mathbf{x}_1, \dots, \mathbf{x}_N) = p(\mathbf{y}_n \mid \mathbf{x}_n). \quad (16.27)$$

In words, *the future is independent of the past given the present, and the observations are independent of the future and past given the present*.

The previously stated independencies are graphically represented via the graph of Figure 16.15. If the hidden variables are of a discrete nature, the resulting model is known as a hidden Markov model. If on the other hand, both hidden as well as observation variables are of a continuous nature, the resulting model gets rather involved to deal with. However, analytically tractable tools can be and have been

**FIGURE 16.15**

The Bayesian network corresponding to a latent Markov model. If latent variables are of a discrete nature, this corresponds to an HMM. If both observed as well as latent variables are continuous and follow a Gaussian distribution, this corresponds to a linear dynamic system (LDS). Note that the observed variables comprise the leaves of the graph.

developed for some special cases. In the so-called *linear dynamic systems* (LDS), the system dynamics and the generation of the observations are modeled as

$$\mathbf{x}_n = F_n \mathbf{x}_{n-1} + \boldsymbol{\eta}_n, \quad (16.28)$$

$$\mathbf{y}_n = H_n \mathbf{x}_n + \boldsymbol{v}_n, \quad (16.29)$$

where $\boldsymbol{\eta}_n$ and \boldsymbol{v}_n are zero-mean, mutually independent noise disturbances modeled by Gaussian distributions. This is the celebrated Kalman filter, which we have already discussed in [Chapter 4](#) and it will also be considered, from a probabilistic perspective, in [Chapter 17](#). The probabilistic counterparts of (16.28)-(16.29) are

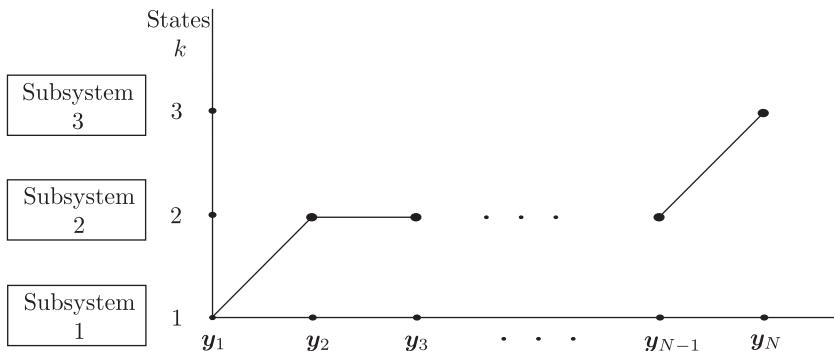
$$p(\mathbf{x}_n | \mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{x}_n | F_n \mathbf{x}_{n-1}, Q_n), \quad (16.30)$$

$$p(\mathbf{y}_n | \mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n | H_n \mathbf{x}_n, R_n), \quad (16.31)$$

where Q_n and R_n are the covariance matrices of $\boldsymbol{\eta}_n$ and \boldsymbol{v}_n , respectively.

16.5 HIDDEN MARKOV MODELS

Hidden Markov models are represented by the graphical model in [Figures 16.15](#) and Eqs. (16.26), (16.27). The latent variables are discrete; hence, we write the *transition* probability as $P(\mathbf{x}_n | \mathbf{x}_{n-1})$ and this corresponds to a table of probabilities. Observation variables can either be discrete or continuous. Basically, an HMM is used to model a *quasi-stationary* process that undergoes *sudden* changes among a number of, say K , subprocesses. Each one of these subprocesses is described by different statistical properties. One could alternatively view it as a combined system comprising a number of subsystems; each one of these subsystems generates data/observations according to a different statistical model; for example, one may follow a Gaussian and the other one a student's t-distribution. Observations are emitted by these subsystems; however, once an observation is received, we do not know which subsystem this was emitted from. This reminds us of the mixture modeling task of a pdf; however, in mixture modeling, we did not care about the sequence in which observations occur.

**FIGURE 16.16**

The unfolding in time of a trajectory that associates observations with states.

For modeling purposes, we associate with each observation, y_n , a hidden variable, $k_n = 1, 2, \dots, K$, which is the (random) index indicating the subsystem/subprocess that generated the respective observation vector. We will call it the *state*. Each k_n corresponds to \mathbf{x}_n of the general model. The sequence of the complete observation set, (y_n, k_n) , $n = 1, 2, \dots, N$, forms a trajectory in a two-dimensional grid, having the states on one axis and the observations on the other. This is shown in Figure 16.16 for $K = 3$. Such a path reveals the origin of each observation; y_1 was emitted from state $k_1 = 1$, y_2 from $k_2 = 2$, y_3 from $k_3 = 2$, and y_N from $k_N = 3$. Note that each trajectory is associated with a probability distribution; that is, the joint distribution of the complete set. Indeed, the probability that the trajectory of Figure 16.16 will occur depends on the value of $P((y_1, k_1 = 1), (y_2, k_2 = 2), (y_3, k_3 = 2), \dots, (y_N, k_N = 3))$. We will soon see that some of the possible trajectories that can be drawn in the grid are not allowed in practice; this may be due to physical constraints concerning the data generation mechanism that underlies the corresponding system/process.

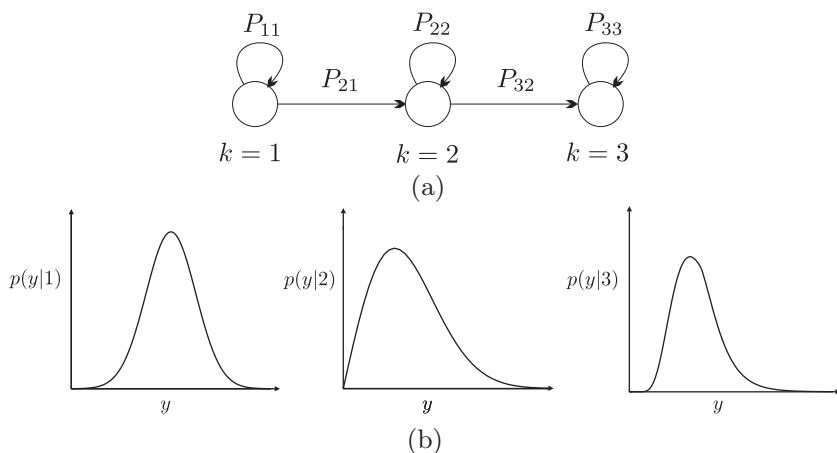
Transition Probabilities. As already said, the dynamics of a latent Markov model are described in terms of the distribution $p(\mathbf{x}_n | \mathbf{x}_{n-1})$, which for an HMM becomes the set of probabilities

$$P(k_n | k_{n-1}), \quad k_n, k_{n-1} = 1, 2, \dots, K,$$

indicating the probability of the system to “jump” at time n to state k_n from state k_{n-1} , where it was at time $n - 1$. In general, this table of probabilities may be time-varying. In the standard form of HMM, this is considered to be independent of time and we say that our model is *homogeneous*. Thus, we can write

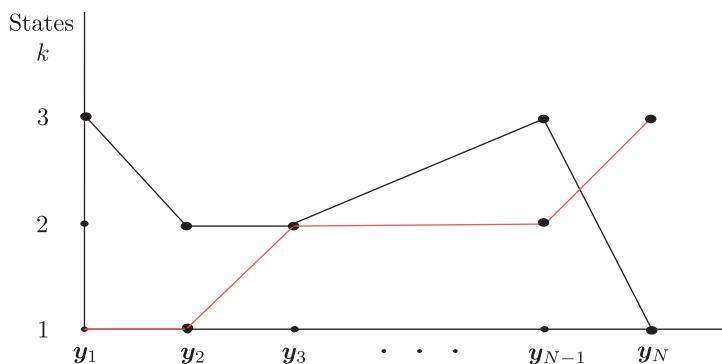
$$P(k_n | k_{n-1}) = P(i|j) := P_{ij}, \quad i, j = 1, 2, \dots, K.$$

Note that some of these transition probabilities can be zero depending on the modeling assumptions. Figure 16.17a shows an example of a three-state system. The model is of the so called *left-to-right* type, where two types of transitions are allowed: (a) self transitions and (b) transitions from a state of a lower index to a state of a higher index. The system, once it jumps into a state k , emits data according to a probability distribution $p(y|k)$, as illustrated in Figure 16.17b. Besides the left-to-right models, other alternatives have also been proposed [8, 63]. The states correspond to certain physical characteristics of the corresponding system. For example in speech recognition, the number of states, that are chosen

**FIGURE 16.17**

(a) A three-state left-to-right HMM model. (b) Each state is characterized by different statistical properties.

to model a spoken word, depends on the expected number of sound phenomena (phonemes) within the word. Typically, three to four states are used per phoneme. Another modeling path uses the average number of observations resulting from various versions of a spoken word as an indication of the number of states. Seen from the transition probabilities perspective, an HMM is basically a stochastic finite state automaton that generates an observation string. Note that the semantics of Figure 16.17 is different and must not be confused with the graphical structure given in Figure 16.15. Figure 16.17 is a graphical interpretation of the transition probabilities among the states; it says nothing about independencies among the involved random variables. Once a state transition model has been adopted, some trajectories in the trellis diagram of Figure 16.16 will not be allowed. In Figure 16.18, the red trajectory is not in line with the model of Figure 16.17.

**FIGURE 16.18**

The black trajectory is not allowed to occur, under the HMM model of Figure 16.17. Transitions from state $k = 3$ to state $k = 2$ and from $k = 3$ to $k = 1$ are not permitted. In contrast, the state unfolding in the red curve is a valid one.

16.5.1 INFERENCE

As in any graphical modeling task, the ultimate goal is inference. Two types of inference are of particular interest in the context of classification/recognition. Let us discuss it in the framework of speech recognition; similar arguments hold true for other applications. We are given a set of (output variables) observations, y_1, \dots, y_N , and we have to decide to which spoken word these correspond. In the database, each spoken word is represented by an HMM model, which is the result of extensive training. An HMM model is fully described by the following set of parameters:

HMM Model Parameters

1. Number of states K .
2. The probabilities for the initial state at $n = 1$ to be at state k , that is, P_k , $k = 1, 2, \dots, K$.
3. The set of transition probabilities P_{ij} , $i, j = 1, 2, \dots, K$.
4. The state emission distributions $p(y|k)$, $k = 1, 2, \dots, K$, which can either be discrete or continuous. Often, these probability distributions may be parameterized, $p(y|k; \theta_k)$, $k = 1, 2, \dots, K$.

Prior to inference, all the involved parameters are assumed to be known. Learning of the HMM parameters takes place in the training phase; we will come to it shortly.

For the recognition, a number of scores can be used. Here we will discuss two alternatives that come as a direct consequence of our graphical modeling approach. For a more detailed discussion, see, for example, [72].

In the first one, the joint distribution for the observed sequence is computed, after marginalizing out all hidden variables; this is done for each one of the models/words. Then the word that scores the larger value is selected. This method corresponds to the sum-product rule. The other path is to compute, for each model/word, the optimal trajectory in the trellis diagram; that is, the trajectory that scores the highest joint probability. In the sequel, we decide in favor of the model/word that corresponds to the largest optimal value. This method is an implementation of the max-sum rule.

The Sum-Product Algorithm: The HMM Case.

The first step is to transform the directed graph of Figure 16.15 to an undirected one; a factor graph or a junction tree graph. Note that this is trivial for this case, as the graph is already a tree. Let us work with the junction tree formulation. Also, in order to use the message-passing formulas of (16.4) and (16.5) as well as (16.7) and (16.6) for computing the distribution values, we will first adopt a more compact way of representing the conditional probabilities. We will employ the technique that was used in Section 13.4 for the mixture modeling case. Let us denote each latent variable as a K -dimensional vector, $x_n \in \mathbb{R}^K$, $n = 1, 2, \dots, N$, whose elements are all zero except at the k th location, where k is the index of the (unknown) state from which y_n has been emitted, that is,

$$\mathbf{x}_n^T = [x_{n,1}, x_{n,2}, \dots, x_{n,K}] : \begin{cases} x_{n,i} = 0, & i \neq k \\ x_{n,k} = 1. \end{cases}$$

Then, we can compactly write

$$P(x_1) = \prod_{k=1}^K P_k^{x_{1,k}}, \quad (16.32)$$

and

$$P(\mathbf{x}_n | \mathbf{x}_{n-1}) = \prod_{i=1}^K \prod_{j=1}^K P_{ij}^{x_{n-1,j} x_{n,i}}. \quad (16.33)$$

Indeed, if the jump is from a specific state j at time $n - 1$, to a specific state i at time n , then the only term that survives in the previous product is the corresponding factor, P_{ij} . The joint probability distribution of the complete set, as a direct consequence of the Bayesian network model of Figure 16.15, is written as

$$p(Y, X) = P(\mathbf{x}_1)p(\mathbf{y}_1 | \mathbf{x}_1) \prod_{n=2}^N P(\mathbf{x}_n | \mathbf{x}_{n-1})p(\mathbf{y}_n | \mathbf{x}_n), \quad (16.34)$$

where

$$p(\mathbf{y}_n | \mathbf{x}_n) = \prod_{k=1}^K (p(\mathbf{y}_n | k; \theta_k))^{x_{n,k}}. \quad (16.35)$$

The corresponding junction tree is trivially obtained from the graph in 16.15. Replacing directed links with undirected ones and considering cliques of size two, the graph in Figure 16.19a results. However, as all the \mathbf{y}_n variables are observed (*instantiated*) and no marginalization is required, their multiplicative contribution can be absorbed by the respective conditional probabilities, which leads to the graph of 16.19b. Alternatively, this junction tree can be obtained if one considers the nodes $(\mathbf{x}_1, \mathbf{y}_1)$ and $(\mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{y}_n)$, $n = 2, 3, \dots, N$, to form cliques associated with the potential functions

$$\psi_1(\mathbf{x}_1, \mathbf{y}_1) = P(\mathbf{x}_1)p(\mathbf{y}_1 | \mathbf{x}_1), \quad (16.36)$$

and

$$\psi_n(\mathbf{x}_{n-1}, \mathbf{y}_n, \mathbf{x}_n) = P(\mathbf{x}_n | \mathbf{x}_{n-1})p(\mathbf{y}_n | \mathbf{x}_n), \quad n = 2, \dots, N. \quad (16.37)$$

The junction tree of Figure 16.19b results by eliminating nodes from the cliques starting from \mathbf{x}_1 . Note that the normalizing constant is equal to one, $Z = 1$.

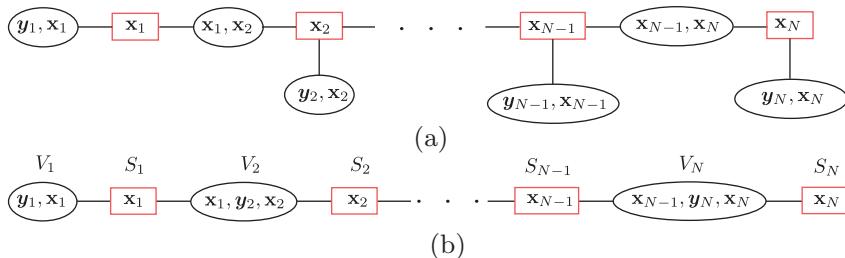


FIGURE 16.19

(a) The junction tree that results from the graph of Figure 16.15. (b) Because \mathbf{y}_n are observed, their effect is only of a multiplicative nature (no marginalization is involved) and its contribution can be trivially absorbed by the potential functions (distributions) associated with the latent variables.

To apply the sum-product rule for junction trees, Eq. (16.5) now becomes

$$\begin{aligned}\mu_{V_n \rightarrow S_n}(\mathbf{x}_n) &= \sum_{\mathbf{x}_{n-1}} \psi_n(\mathbf{x}_{n-1}, \mathbf{y}_n, \mathbf{x}_n) \mu_{S_{n-1} \rightarrow V_n}(\mathbf{x}_{n-1}) \\ &= \sum_{\mathbf{x}_{n-1}} \mu_{S_{n-1} \rightarrow V_n}(\mathbf{x}_{n-1}) P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{y}_n | \mathbf{x}_n).\end{aligned}$$

Also,

$$\mu_{S_{n-1} \rightarrow V_n}(\mathbf{x}_{n-1}) = \mu_{V_{n-1} \rightarrow S_{n-1}}(\mathbf{x}_{n-1}). \quad (16.38)$$

Thus,

$$\mu_{V_n \rightarrow S_n}(\mathbf{x}_n) = \sum_{\mathbf{x}_{n-1}} \mu_{V_{n-1} \rightarrow S_{n-1}}(\mathbf{x}_{n-1}) P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{y}_n | \mathbf{x}_n), \quad (16.39)$$

with

$$\mu_{V_1 \rightarrow S_1}(\mathbf{x}_1) = P(\mathbf{x}_1) p(\mathbf{y}_1 | \mathbf{x}_1). \quad (16.40)$$

In the HMM literature, it is common to use the “alpha” symbol for the exchanged messages, that is,

$$\alpha(\mathbf{x}_n) := \mu_{V_n \rightarrow S_n}(\mathbf{x}_n). \quad (16.41)$$

If one considers that the message-passing terminates at a node V_n , then based on (16.7), and taking into account that the variables $\mathbf{y}_1, \dots, \mathbf{y}_n$, are clumped to the observed values (recall related comment following Eq. (15.44)), it is readily seen that

$$\alpha(\mathbf{x}_n) = p(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n, \mathbf{x}_n), \quad (16.42)$$

which can also be deduced by the respective definitions in (16.39, 16.40); all hidden variables, except \mathbf{x}_n , have been marginalized out. This is a set of K probability values (one for each value of \mathbf{x}_n). For example, for $\mathbf{x}_n : \mathbf{x}_{n,k} = 1$, $\alpha(\mathbf{x}_n)$ is the probability of the trajectory to be at time n at state k , and having obtained the specific observations up to and including time n . From (16.42), one can readily obtain the joint probability distribution (evidence) over the observation sequence, comprising N time instants, that is,

$$p(Y) = \sum_{\mathbf{x}_N} p(\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N, \mathbf{x}_N) = \sum_{\mathbf{x}_N} \alpha(\mathbf{x}_N) : \text{Evidence of Observations,}$$

which, as said in the beginning of the section, is a quantity used for classification/recognition.

In the signal processing “jargon,” the computation of $\alpha(\mathbf{x}_n)$ is referred to as the *filtering* recursion. By the definition of $\alpha(\mathbf{x}_n)$, we have that [2]

$$\alpha(\mathbf{x}_n) = \underbrace{p(\mathbf{y}_n | \mathbf{x}_n)}_{\text{corrector}} \cdot \underbrace{\sum_{\mathbf{x}_{n-1}} \alpha(\mathbf{x}_{n-1}) P(\mathbf{x}_n | \mathbf{x}_{n-1})}_{\text{predictor}} : \text{Filtering Recursion.} \quad (16.43)$$

As it is the case with the Kalman filter to be treated in Chapter 17, the only difference there will be that the summation is replaced by integration. Having adopted Gaussian distributions, these integrations translate into updates of the respective mean values and covariance matrices. The physical meaning of (16.43) is that the predictor provides a prediction on the state using all the past information prior to n .

Then this information is corrected based on the observation \mathbf{y}_n , which is received at time n . Thus, the updated information, based on the entire observation sequence up to and including the current time n , is readily available by

$$P(\mathbf{x}_n | Y_{[1:n]}) = \frac{\alpha(\mathbf{x}_n)}{p(Y_{[1:n]})},$$

where the denominator is given by $\sum_{\mathbf{x}_n} \alpha(\mathbf{x}_n)$, and $Y_{[1:n]} := (\mathbf{y}_1, \dots, \mathbf{y}_n)$.

Let us now carry on with the second message-passing phase, in the opposite direction than before, in order to obtain

$$\mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n) = \sum_{\mathbf{x}_{n+1}} \mu_{S_{n+1} \rightarrow V_{n+1}}(\mathbf{x}_{n+1}) P(\mathbf{x}_{n+1} | \mathbf{x}_n) p(\mathbf{y}_{n+1} | \mathbf{x}_{n+1}),$$

with

$$\mu_{S_{n+1} \rightarrow V_{n+1}}(\mathbf{x}_{n+1}) = \mu_{V_{n+2} \rightarrow S_{n+1}}(\mathbf{x}_{n+1}).$$

Hence,

$$\mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n) = \sum_{\mathbf{x}_{n+1}} \mu_{V_{n+2} \rightarrow S_{n+1}}(\mathbf{x}_{n+1}) P(\mathbf{x}_{n+1} | \mathbf{x}_n) p(\mathbf{y}_{n+1} | \mathbf{x}_{n+1}), \quad (16.44)$$

with

$$\mu_{V_{N+1} \rightarrow S_N}(\mathbf{x}_N) = 1. \quad (16.45)$$

Note that $\mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n)$ involves K values and for the computation of each one of them K summations are performed. So, the complexity scales as $\mathcal{O}(K^2)$ per time instant. In HMM literature, the symbol “beta” is used,

$$\beta(\mathbf{x}_n) = \mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n). \quad (16.46)$$

From the recursive definition in (16.44, 16.45), where $\mathbf{x}_{n+1}, \mathbf{x}_{n+2}, \dots, \mathbf{x}_N$ have been marginalized out, we can equivalently write

$$\beta(\mathbf{x}_n) = p(\mathbf{y}_{n+1}, \mathbf{y}_{n+2}, \dots, \mathbf{y}_N | \mathbf{x}_n). \quad (16.47)$$

That is, conditioned on the values of \mathbf{x}_n , for example, $\mathbf{x}_n : x_{nk} = 1$, $\beta(\mathbf{x}_n)$ is the value of the joint distribution for the observed values, $\mathbf{y}_{n+1}, \dots, \mathbf{y}_N$, to be emitted when the system is at state k at time n .

We have now all the “ingredients” in order to compute marginals. From (16.7), we obtain (justify it based on the independence properties that underlie an HMM)

$$\begin{aligned} p(\mathbf{x}_n, \mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N) &= \mu_{V_{n-1} \rightarrow S_n}(\mathbf{x}_n) \mu_{V_{n+1} \rightarrow S_n}(\mathbf{x}_n) \\ &= \alpha(\mathbf{x}_n) \beta(\mathbf{x}_n), \end{aligned} \quad (16.48)$$

which in turns leads to

$$\gamma(\mathbf{x}_n) := P(\mathbf{x}_n | Y) = \frac{\alpha(\mathbf{x}_n) \beta(\mathbf{x}_n)}{p(Y)} : \text{ Smoothing Recursion.}$$

(16.49)

This part of the recursion is known as the *smoothing* recursion. Note that in this computation, both past (via $\alpha(\mathbf{x}_n)$) and future (via $\beta(\mathbf{x}_n)$) data are involved.

An alternative way to obtain $\gamma(\mathbf{x}_n)$ is via its own recursion together with $\alpha(\mathbf{x}_n)$, by avoiding $\beta(\mathbf{x}_n)$ ([Problem 16.16](#)). In such a scenario, both passing messages are related to densities with regard to \mathbf{x}_n , which has certain advantages for the case of linear dynamic systems.

Finally, from [\(16.6\)](#) and recalling [\(16.38\)](#), [\(16.41\)](#), and [\(16.46\)](#), we obtain

$$\begin{aligned} p(\mathbf{x}_{n-1}, \mathbf{x}_n, Y) &= P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{y}_n | \mathbf{x}_n) \mu_{S_n \rightarrow V_n}(\mathbf{x}_n) \mu_{S_{n-1} \rightarrow V_{n-1}}(\mathbf{x}_{n-1}) \\ &= \alpha(\mathbf{x}_{n-1}) P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{y}_n | \mathbf{x}_n) \beta(\mathbf{x}_n), \end{aligned} \quad (16.50)$$

or

$$\begin{aligned} p(\mathbf{x}_{n-1}, \mathbf{x}_n | Y) &= \frac{\alpha(\mathbf{x}_{n-1}) P(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{y}_n | \mathbf{x}_n) \beta(\mathbf{x}_n)}{p(Y)} \\ &:= \xi(\mathbf{x}_{n-1}, \mathbf{x}_n). \end{aligned} \quad (16.51)$$

Thus, $\xi(\cdot, \cdot)$ is a table of K^2 probability values. Let $\xi(x_{n-1,j}, x_{n,i})$ correspond to $x_{n-1,j} = x_{n,i} = 1$. Then $\xi(x_{n-1,j}, x_{n,i})$ is the probability of the system being at states j and i at times $n - 1$ and n , respectively, conditioned on the transmitted sequence of observations.

In [Section 15.7.4](#) a message-passing scheme was proposed for the efficient computation of the maximum of the joint distribution. This can also be applied in the junction tree associated with an HMM. The resulting algorithm is known as the *Viterbi* algorithm. The Viterbi algorithm results in a straightforward way from the general max-sum algorithm. The algorithm is similar to the one derived before; all one has to do is to replace summations with the maximum operations. As we have already commented, while discussing the max-product rule, computing the sequence of the complete set $(\mathbf{y}_n, \mathbf{x}_n)$, $n = 1, 2, \dots, N$, that maximizes the joint probability, using back-tracking, equivalently defines the optimal trajectory in the two-dimensional grid.

Another inference task that is of interest in practice, besides recognition, is prediction. That is, given an HMM and the observation sequence \mathbf{y}_n , $n = 1, 2, \dots, N$, to optimally predict the value \mathbf{y}_{n+1} . This can also be performed efficiently by appropriate marginalization ([Problem 16.17](#)).

16.5.2 LEARNING THE PARAMETERS IN AN HMM

This is the second time we refer to the learning of graphical models. The first time was at the end of [Section 16.3.2](#). The most natural way to obtain the unknown parameters is to maximize the likelihood/evidence of the joint probability distribution. Because our task involves both observed as well as latent variables, the EM algorithm is the first one that comes to mind. However, the underlying independencies in an HMM will be employed in order to come up with an efficient learning scheme. The set of the unknown parameters, Θ , involves (a) the initial state probabilities, P_k , $k = 1, \dots, K$, (b) the transition probabilities, P_{ij} , $i, j = 1, 2, \dots, K$, and (c) the parameters in the probability distributions associated with the observations, θ_k , $k = 1, 2, \dots, K$.

Expectation Step: From the general scheme presented in [Section 12.5.1](#) (with Y in place of \mathcal{X} , X in place of \mathcal{X}^l , and Θ in place of ξ) at the $(t + 1)$ th iteration, we have to compute

$$\mathcal{Q}(\Theta, \Theta^{(t)}) = \mathbb{E} [\ln p(Y, X; \Theta)]$$

where $\mathbb{E}[\cdot]$ is the expectation with respect to $P(X|Y; \Theta^{(t)})$. From (16.32), (16.33), (16.34), and (16.35) we obtain

$$\begin{aligned}\ln p(Y, X; \Theta) &= \sum_{k=1}^K (x_{1,k} \ln P_k + \ln p(y_1|k; \theta_k)) \\ &\quad + \sum_{n=2}^N \sum_{i=1}^K \sum_{j=1}^K (x_{n-1,j} x_{n,i}) \ln P_{ij} \\ &\quad + \sum_{n=2}^N \sum_{k=1}^K x_{n,k} \ln p(y_n|k; \theta_k),\end{aligned}$$

thus,

$$\begin{aligned}\mathcal{Q}(\Theta, \Theta^{(t)}) &= \sum_{k=1}^K \mathbb{E}[x_{1,k}] \ln P_k + \sum_{n=2}^N \sum_{i=1}^K \sum_{j=1}^K \mathbb{E}[x_{n-1,j} x_{n,i}] \ln P_{ij} \\ &\quad + \sum_{n=1}^N \sum_{k=1}^K \mathbb{E}[x_{n,k}] \ln p(y_n|k; \theta_k).\end{aligned}\tag{16.52}$$

Let us now recall (16.49) to obtain

$$\mathbb{E}[x_{n,k}] = \sum_{\mathbf{x}_n} P(\mathbf{x}_n|Y; \Theta^{(t)}) x_{n,k} = \sum_{\mathbf{x}_n} \gamma(\mathbf{x}_n; \Theta^{(t)}) x_{n,k}.$$

Note that $x_{n,k}$ can either be zero or one; hence, its mean value will be equal to the probability that \mathbf{x}_n has the k th element $x_{n,k} = 1$ and we denote it as

$$\mathbb{E}[x_{n,k}] = \gamma(x_{n,k} = 1; \Theta^{(t)}),\tag{16.53}$$

Recall that given $\Theta^{(t)}$, $\gamma(\cdot; \Theta^{(t)})$ can be efficiently computed via the sum-product algorithm described before. In a similar spirit and mobilizing the definition in (16.51), we can write

$$\begin{aligned}\mathbb{E}[x_{n-1,j} x_{n,i}] &= \sum_{\mathbf{x}_n} \sum_{\mathbf{x}_{n-1}} P(\mathbf{x}_n, \mathbf{x}_{n-1}|Y; \Theta^{(t)}) x_{n-1,j} x_{n,i} \\ &= \sum_{\mathbf{x}_n} \sum_{\mathbf{x}_{n-1}} \xi(\mathbf{x}_n, \mathbf{x}_{n-1}; \Theta^{(t)}) x_{n-1,j} x_{n,i} \\ &= \xi(x_{n-1,j} = 1, x_{n,i} = 1; \Theta^{(t)}).\end{aligned}\tag{16.54}$$

Note that $\xi(\cdot, \cdot; \Theta^{(t)})$ can also be efficiently computed as a by-product of the sum-product algorithm, given $\Theta^{(t)}$. Thus, we can summarize the E-step as

$$\begin{aligned}\mathcal{Q}(\Theta, \Theta^{(t)}) &= \sum_{k=1}^K \gamma(x_{1,k} = 1; \Theta^{(t)}) \ln P_k \\ &\quad + \sum_{n=2}^N \sum_{i=1}^K \sum_{j=1}^K \xi(x_{n-1,j} = 1, x_{n,i} = 1; \Theta^{(t)}) \ln P_{ij} \\ &\quad + \sum_{n=1}^N \sum_{k=1}^K \gamma(x_{n,k} = 1; \Theta^{(t)}) \ln p(y_n|k; \theta_k).\end{aligned}\tag{16.55}$$

Maximization Step: In this step, it suffices to obtain the derivatives/gradients with regard to P_k , P_{ij} , and θ_k and equate them to zero in order to obtain the new estimates, which will comprise $\Theta^{(t+1)}$. Note that P_k and P_{ij} are probabilities; hence, their maximization should be constrained so that

$$\sum_{k=1}^K P_k = 1 \quad \text{and} \quad \sum_{i=1}^K P_{ij} = 1, \quad j = 1, 2, \dots, K.$$

The resulting reestimation formulas are (Problem 16.18)

$$P_k^{(t+1)} = \frac{\gamma(x_{1,k} = 1; \Theta^{(t)})}{\sum_{i=1}^K \gamma(x_{1,i} = 1; \Theta^{(t)})}, \quad (16.56)$$

$$P_{ij}^{(t+1)} = \frac{\sum_{n=2}^N \xi(x_{n-1,j} = 1, x_{n,i} = 1; \Theta^{(t)})}{\sum_{n=2}^N \sum_{k=1}^K \xi(x_{n-1,j} = 1, x_{n,k} = 1; \Theta^{(t)})}. \quad (16.57)$$

The reestimation of θ_k depends on the form of the corresponding distribution $p(y_n | k; \theta_k)$. For example, in the Gaussian scenario, the parameters are the mean values and the elements of the covariance matrix. In this case, we obtain exactly the same iterations as those resulting for the problem of Gaussian mixtures (see Eqs. (12.87) and (12.88)), if in place of the posterior we use γ .

In summary, training an HMM comprises the following steps:

1. Initialize the parameters in Θ .
2. Run the sum-product algorithm to obtain $\gamma(\cdot)$ and $\xi(\cdot, \cdot)$, using the current set of parameter estimates.
3. Update the parameters as in (16.56) and (16.57).

Iterations in steps 2 and 3 continue until a convergence criterion is met, such as in EM. This iterative scheme is also known as the *Baum-Welch* or *forward-backward* algorithm. Besides the forward-backward algorithm for training HMMs, the literature is rich in a number of alternatives with the goal of either simplifying computations or improving performance. For example, a simpler training algorithm can be derived tailored to the Viterbi scheme for computing the optimum path (e.g., [63, 72]). Also, to further simplify the training algorithm, we can assume that our state observation variables, y_n , are discretized (quantized) and can take values from a finite set of L possible ones, $\{1, 2, \dots, L\}$. This is often the case in practice. Furthermore, assume that the first state is also known. This is, for example, the case for left-to-right models like the one shown in Figure 16.17. In such a case, we need not compute estimates of the initial probabilities. Thus, the unknown parameters to be estimated are the transition probabilities and the probabilities $P_y(r|i)$, $r = 1, 2, \dots, L$, $i = 1, 2, \dots, K$; that is, the probability of emitting symbol r from state i .

Viterbi reestimation: In the speech literature, the algorithm is also known as the *segmental k-means training* algorithm [63]. It evolves around the concept of the best path.

Definitions:

- $n_{ilj} :=$ number of transitions from state j to state i .
- $n_{.lj} :=$ number of transitions originated from state j .
- $n_{il.} :=$ number of transitions terminated at state i .
- $n(r|i) :=$ number of times observation $r \in \{1, 2, \dots, L\}$ occurs jointly with state i .

Iterations:

- Initial conditions: Assume the initial estimates of the unknown parameters. Obtain the best path and compute the associated cost, say D , along the path.
- Step 1: From the available best path, reestimate the new model parameters as

$$P^{(\text{new})}(i|j) = \frac{n_{ij}}{n_{\cdot j}}$$

$$P_x^{(\text{new})}(r|i) = \frac{n(r|i)}{n_{ri}}$$

- Step 2: For the new model parameters, obtain the best path and compute the corresponding overall cost $D^{(\text{new})}$. Compare it with the cost D of the previous iteration. If $D^{(\text{new})} - D > \epsilon$, set $D = D^{(\text{new})}$ and go to step 1. Otherwise stop.

The Viterbi reestimation algorithm can be shown to converge to a proper characterization of the underlying observations [14].

Remarks 16.4.

- Scaling:* The probabilities, α and β , being less than one, as iterations progress can take very small values. In practice, the dynamic range of their computed values may exceed that of the computer. This phenomenon can be efficiently dealt within an appropriate scaling. If this is done properly on both α and β , then the effect of scaling cancels out [63].
- Insufficient Training Data Set:* Generally, a large amount of training data is necessary to learn the HMM parameters. The observation sequence must be sufficiently long with respect to the number of states of the HMM model. This will guarantee that all state transitions will appear a sufficient number of times, so that the reestimation algorithm learns their respective parameters. If this is not the case, a number of techniques have been devised to cope with the issue. For a more detailed treatment, the reader may consult [8, 63] and the references therein.

16.5.3 DISCRIMINATIVE LEARNING

Discriminative learning is another path that has attracted a lot of attention. Note that the EM algorithm optimizes the likelihood with respect to the unknown parameters of a *single* HMM in “isolation”; that is, without considering the rest of the HMMs, which model the other words (in case of speech recognition) or other templates/prototypes that are stored in the database. Such an approach is in line with what we defined as generative learning in Chapter 3. In contrast, the essence of discriminative learning is to optimize the set of parameters so that the models become optimally discriminated over the training sets (e.g., in terms of the error probability criterion). In other words, the parameters describing the different statistical models (HMMs) are optimized in a *combined* way, not individually. The goal is to make the different HMM models as distinct as possible, according to a criterion. This has been an intense line of research and a number of techniques have been developed around criteria that lead to either convex or nonconvex optimization methods; see, for example, [33] and the references therein.

Remarks 16.5.

- Besides the basic HMM scheme, which was described in this section, a number of variants have been proposed in order to overcome some of its shortcomings. For example, alternative modeling paths concern the first-order Markov property and propose models to extend correlations to longer times.

In the *autoregressive HMM* [11], links are added among the observation nodes of the basic HMM scheme in [Figure 16.15](#); for example, y_n is not only linked to x_n but it shares direct links with, for example, y_{n-2} , y_{n-1} , y_{n+1} , and y_{n+2} , if the model extends correlations up to two time instants away.

A different concept has been introduced in [56] in the context of *segment modeling*. According to this model, each state is allowed to emit, say, d , successive observations, that comprise a segment. The length of the segment, d , is itself a random variable and it is associated with a probability $P(d|k)$, $k = 1, 2, \dots, K$. In this way, correlation is introduced via the joint distribution of the samples comprising the segment.

- *Variable Duration HMM*: A serious shortcoming of the HMMS, which is often observed in practice, is associated with the self transition probabilities, $P(k|k)$, which are among the model parameters associated with an HMM. Note that the probability of the model being at state k for d successive instants (initial transition to the state and $d - 1$ self transitions) is given by

$$P_k(d) = (P(k|k))^{d-1} (1 - P(k|k)),$$

where $1 - P(k|k)$ is the probability of leaving the state. For many cases, this exponential state duration dependence is not realistic. In variable duration HMMS, $P_k(d)$ is explicitly modeled. Different models for $P_k(d)$ can be employed (see, e.g., [46, 68, 72]).

- Hidden Markov modeling is among the most powerful tools in machine learning and has been widely used in a large number of applications besides speech recognition. Some sampled references are [9] in bioinformatics, [16, 36] in communications, [4, 73] in optical character recognition (OCR), and [40, 61, 62] in music analysis/recognition, to name but a few.

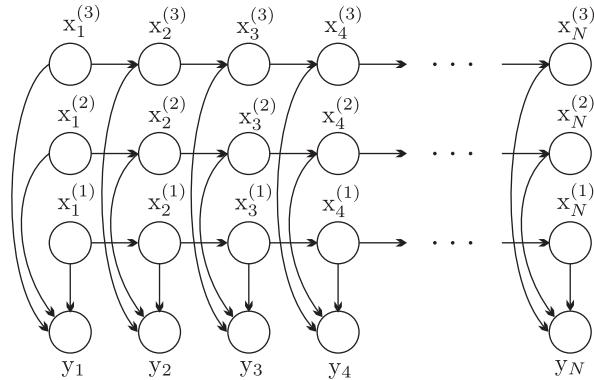
For a further discussion on HMMS, see, for example, [8, 64, 72].

16.6 BEYOND HMMS: A DISCUSSION

In this section, some notable extensions of the hidden Markov models, which were previously discussed, are considered in order to meet requirements of applications where either the number of states is large or the homogeneity assumption is no more justified.

16.6.1 FACTORIAL HIDDEN MARKOV MODELS

In the HMMS considered before, the system dynamics is described via the hidden variables, whose graphical representation is a chain. However, such a model may turn out to be too simple for certain applications. A variant of the HMM involves M chains, instead of one chain, where each chain of hidden variables unfolds in time independently of the others. Thus at time n , M hidden variables are involved, denoted as $x_n^{(m)}$, $m = 1, 2, \dots, M$, [17, 34, 81]. The observations occur as a combined emission where all hidden variables are involved. The respective graphical structure is shown in [Figure 16.20](#) for $M = 3$. Each one of the chains develops on its own, as the graphical model suggests. Such models are known as *factorial HMM* (FHMM). One obvious question is why not use a single chain of hidden variables by increasing the number of possible states? It turns out that such a naive approach would blow up complexity. Take as an example the case of $M = 3$, where for each one of the hidden variables, the number of states is equal to 10. The table of transition probabilities for

**FIGURE 16.20**

A factorial HMM with three chains of hidden variables.

each chain requires 10^2 entries which amounts to a total number of 300; i.e., $P_{ij}^{(m)}$, $i,j = 1, 2, \dots, 10$, $m = 1, 2, 3$. Moreover, the total number of state combinations, which can be realized is $10^3 = 1000$. To implement the same number of states via a single chain one would need a table of transition probabilities equal to $(10^3)^2 = 10^6$!

Let \mathbb{X}_n be the M-tuple $(x_n^{(1)}, \dots, x_n^{(M)})$, where each $x_n^{(m)}$ has only one of its elements equal to 1 (indicating a state) and the rest are zero. Then,

$$P(\mathbb{X}_n | \mathbb{X}_{n-1}) = \prod_{m=1}^M P^{(m)}(x_n^{(m)} | x_{n-1}^{(m)}).$$

In [17], the Gaussian distribution was employed for the observations, that is,

$$p(y_n | \mathbb{X}_n) = \mathcal{N}\left(y_n \middle| \sum_{m=1}^M \mathcal{M}^{(m)} x_n^{(m)}, \Sigma\right), \quad (16.58)$$

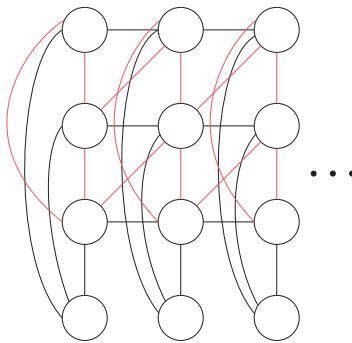
where

$$\mathcal{M}^{(m)} = [\mu_1^{(m)}, \dots, \mu_K^{(m)}], \quad m = 1, 2, \dots, M \quad (16.59)$$

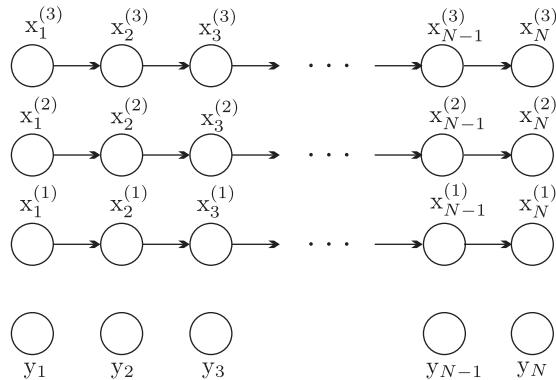
are the matrices comprising the mean vectors associated with each state and the covariance matrix is assumed to be known and the same for all. The joint probability distribution is given by

$$p(\mathbb{X}_1 \dots \mathbb{X}_N, Y) = \prod_{m=1}^M \left(P^{(m)}(x_1^{(m)}) \prod_{n=2}^N P^{(m)}(x_n^{(m)} | x_{n-1}^{(m)}) \right) \times \prod_{n=1}^N p(y_n | \mathbb{X}_n) \quad (16.60)$$

The challenging task in factorial HMMs is complexity. This is illustrated in Figure 16.21, where the explosion in the size of cliques after performing the moralization and triangulation steps is readily deduced.

**FIGURE 16.21**

The graph resulting from a factorial HMM with three chains of hidden variables, after the moralization (it links variables in the same time instant) and triangulation (it links variables between neighboring time instants) steps.

**FIGURE 16.22**

The simplified graphical structure of a FHMM comprising three chains used in the framework of variational approximation. The nodes associated with the observed variables are delinked.

In [17], the variational approximation method is adopted to simplify the structure. However, in contrast to the complete factorization scheme, which was adopted for the approximating distribution, Q , in Eq. (16.17) for the Boltzmann machine (corresponding to the removal of all edges in the graph), here the approximating graph will have a more complex structure. Only the edges connected to the output nodes are removed; this results in the graphical structure of Figure 16.22, for $M = 3$. Because this structure is tractable, there is no need for further simplifications. The approximate conditional distribution, Q , of the simplified structure is parameterized in terms of a set of variational parameters, $\lambda_n^{(m)}$ (one for each delinked node), and it is written as

$$Q(\mathbb{X}_1 \dots \mathbb{X}_N | Y; \lambda) = \prod_{m=1}^M \left(\tilde{P}^{(m)}(x_1^{(m)}) \prod_{n=2}^N \tilde{P}^{(m)}(x_n^{(m)} | x_{n-1}^{(m)}) \right), \quad (16.61)$$

where,

$$\tilde{P}^{(m)} \left(\mathbf{x}_n^{(m)} | \mathbf{x}_{n-1}^{(m)} \right) = P^{(m)} \left(\mathbf{x}_n^{(m)} | (\mathbf{x}_{n-1}^{(m)}) \lambda_n^{(m)} \right), \quad m = 2, \dots, M, \quad n = 1, 2, \dots, N,$$

and

$$\tilde{P}^{(1)} \left(\mathbf{x}_1 \right) = P^{(1)} \left(\mathbf{x}_1 \right) \lambda_1^{(m)}.$$

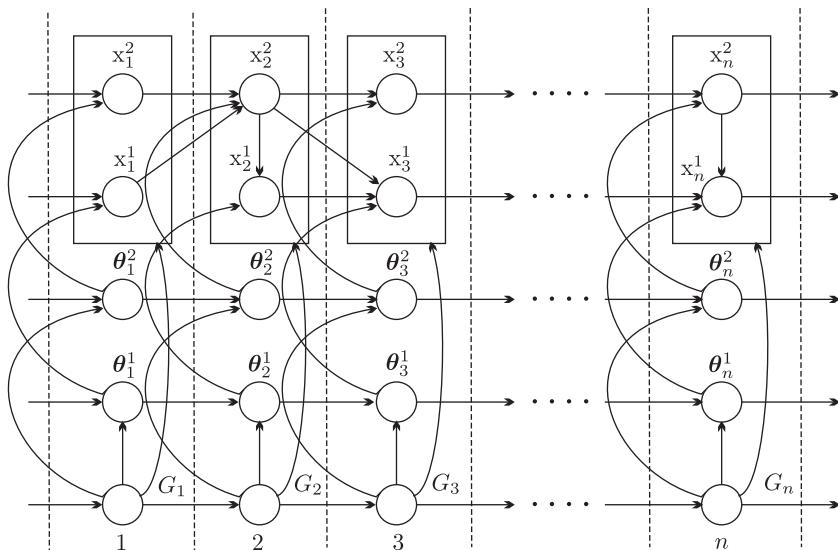
The variational parameters are estimated by minimizing the Kullback-Leibler distance between Q and the conditional distribution associated with (16.60). This compensates for some of the information loss caused by the removal of the observation nodes. The optimization process renders the variational parameters interdependent; this (deterministic) interdependence can be viewed as an approximation to the probabilistic dependence imposed by the exact structure prior to the approximation.

16.6.2 TIME-VARYING DYNAMIC BAYESIAN NETWORKS

Hidden Markov as well as factorial hidden Markov models are homogeneous; hence, both the structure and the parameters are fixed throughout time. However, such an assumption is not satisfying for a number of applications where the underlying relationships as well as the structural pattern of a system undergoes changes as time evolves. For example, the gene interactions do not remain the same throughout life; the appearance of an object across multiple cameras is continuously changing. For systems that are described by parameters whose values are varying slowly in an interval, we have already discussed a number of alternatives in previous chapters. The theory of graphical models provides the tools to study systems with a mixed set of parameters (discrete and continuous); also, graphical models lend themselves to modeling of nonstationary environments, where step changes are also involved.

One path toward time-varying modeling is to consider graphical models of fixed structure but with time varying parameters, known as *switching linear dynamic systems* (SLDS). Such models serve the needs of systems in which a linear dynamic model jumps from one parameter setting to another; hence, the latent variables are both of discrete as well as of continuous nature. At time instant n , a switch discrete variable, $s_n \in \{1, 2, \dots, M\}$, selects a single LDS from an available set of M (sub)systems. The dynamics of s_n is also modeled to comply with the Markovian philosophy, and transitions from one LDS to another are governed by $P(s_n|s_{n-1})$. This problem has a long history and its origins can be traced back to the time just after the publication of the seminal paper by Kalman [35]; see, for example, [18] and [2, 3] for a more recent review of related techniques concerning the approximate inference task in such networks.

Another path is to consider that both the structure as well as the parameters change over time. One route is to adopt a quasi-stationary rationale, and assume that the data sequence is piece-wise stationary in time, for example, [12, 54, 66]. Nonstationarity is conceived as a cascade of stationary models, which have previously been learned by presegmented subintervals. The other route assumes that the structure and parameters are continuously changing, for example, [42, 79]. An example for the latter case is a Bayesian network where the parents of each node and the parameters, which define the conditional distributions, are time varying. A separate variable is employed that defines the structure at each time

**FIGURE 16.23**

The figure corresponds to a time varying dynamic Bayesian network with two variables x_n^1 and x_n^2 , $n = 1, 2, \dots$. The parameters controlling the conditional distributions are considered as separate nodes, θ_n^1 and θ_n^2 , respectively, for the two variables. The structure variable, G_n , controls the values of the parameters as well as the structure of the network, which is continuously changing.

instant; that is, the set of linking directed edges. The concept is illustrated in Figure 16.23. The method has been applied to the task of active camera tracking [79].

16.7 LEARNING GRAPHICAL MODELS

Learning a graphical model consists of two parts. Given a number of observations, one has to specify both the graphical structure as well as the associated parameters.

16.7.1 PARAMETER ESTIMATION

Once a graphical model has been adopted, one has to estimate the unknown parameters. For example, in a Bayesian network involving discrete variables one has to estimate the values of the conditional probabilities. In Section 16.5, the case of learning the unknown parameters in the context of an HMM was presented. The key point was to maximize the joint pdf over the observed output variables. This is among the most popular criteria used for parameter estimation in different graphical structures. In the HMM case, some of the variables were latent, hence the EM algorithm was mobilized. If all the

variables of the graph can be observed, then the task of parameter learning becomes a typical maximum likelihood one. More specifically, let a network with l nodes representing the variables, x_1, \dots, x_l , which are compactly written as a random vector \mathbf{x} . Let also, x_1, x_2, \dots, x_N , be a set of observations; then

$$\hat{\theta} = \arg \max_{\theta} p(x_1, x_2, \dots, x_N; \theta),$$

where θ comprises all the parameters in the graph. If latent variables are involved, then one has to marginalize them out. Any of the parameter estimation techniques that were discussed in [Chapters 12](#) and [13](#) can be used. Moreover, one can take advantage of the special structure of the graph (i.e., the underlying independencies) to simplify computations. In the HMM case, its Bayesian network structure was exploited by bringing the sum-product algorithm into the game. Besides maximum likelihood, one can adopt any other method related to parameter estimation/inference. For example, one can impose a prior $p(\theta)$ on the unknown parameters and resort to a MAP estimation. Moreover, the full Bayesian scenario can also be employed and assume the parameters to be random variables. Such a line presupposes that the unknown parameters have been included as extra nodes to the network, linked appropriately to those of the variables that they affect. As a matter of fact, this is what we did in [Figure 13.2](#), although there, we had not talked about graphical models yet (see also [Figure 16.24](#)). Note that in this case, in order to perform any inference on the variables of the network one should marginalize out the parameters. For example, assume that our l variables correspond to the nodes of a Bayesian network, where the local conditional distributions,

$$p(x_i | \text{Pa}_i; \theta_i), \quad i = 1, 2, \dots, l,$$

depend on the parameters θ_i . Also, assume that the (random) parameters $\theta_i, i = 1, 2, \dots, l$, are mutually independent. Then, the joint distribution over the variables is given by

$$p(x_1, x_2, \dots, x_l) = \prod_{i=1}^l \int_{\theta_i} p(x_i | \text{Pa}_i; \theta_i) p(\theta_i) d\theta_i.$$

Using convenient priors, that is, conjugate priors, computations can be significantly facilitated; we have demonstrated such examples in [Chapters 12](#) and [13](#).

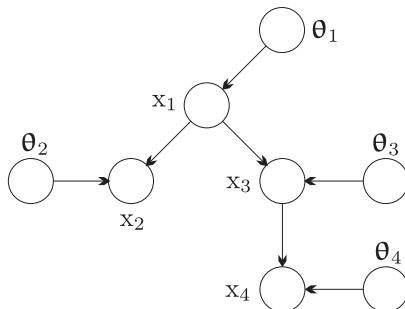
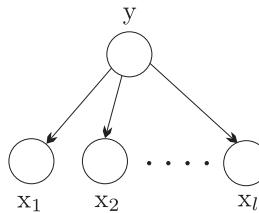


FIGURE 16.24

An example of a Bayesian network, where new nodes associated with the parameters have been included in order to treat parameters as random variables, as it is required by the Bayesian parameter learning approach.

**FIGURE 16.25**

The Bayesian network associated with the naive Bayes classifier. The joint pdf factorizes as $p(y, x_1, \dots, x_l) = p(y) \prod_{i=1}^l p(x_i|y)$.

Besides the previous techniques, which are off-springs of the generative modeling of the underlying processes, discriminative techniques have also been developed.

In a general setting, let us consider a pattern recognition task where the (output) label variable y , and the (input) feature variables, x_1, \dots, x_l , are jointly distributed according to a distribution that can be factorized over a graph, which is parameterized in terms of a vector parameter θ ; that is, $p(y, x_1, x_2, \dots, x_l; \theta)$ [13]. A typical example of such modeling is the naive Bayes classifier, which was discussed in [Chapter 7](#), whose graphical representation is given in [Figure 16.25](#). For a given set of training data, (y_n, x_n) , $n = 1, 2, \dots, N$, the log-likelihood function becomes

$$L(Y, X; \theta) = \sum_{n=1}^N \ln p(y_n, x_n; \theta). \quad (16.62)$$

Estimating θ by maximizing $L(\cdot, \cdot; \theta)$, one would obtain an estimate that guarantees the best (according to the ML criterion) fit of the corresponding distribution to the available training set. However, our ultimate goal is not to model the generation “mechanism” of the data. Our ultimate goal is to classify them correctly. Let us rewrite (16.62) as

$$L(Y, X; \theta) = \sum_{n=1}^N \ln P(y_n|x_n; \theta) + \sum_{n=1}^N \ln p(x_n; \theta).$$

Getting biased toward the classification task, it is more sensible to obtain θ by maximizing the first of the two terms only, that is,

$$\begin{aligned} L_c(Y, X; \theta) &= \sum_{n=1}^N \ln P(y_n|x_n; \theta) \\ &= \sum_{n=1}^N \left(\ln p(y_n, x_n; \theta) - \ln \sum_{y_n} p(y_n, x_n; \theta) \right), \end{aligned} \quad (16.63)$$

where the summation over y_n is over all possible values of y_n (classes). This is known as the *conditional log-likelihood*, see, for example [19, 20, 67]. The resulting estimate, $\hat{\theta}$, guarantees that overall the posterior class probabilities, given the feature values, are maximized over the training data set; after all, Bayesian classification is based on selecting the class of x according to the maximum of the posterior probability. However, one has to be careful. The price one pays for such approaches is that the

conditional log-likelihood is not decomposable, and more sophisticated optimization schemes have to be mobilized. Maximizing the conditional log-likelihood does not guarantee that the error probability is also minimized. This can only be guaranteed if one estimates θ so as to minimize the empirical error probability. However, such a criterion is hard to deal with, as it is not differentiable; attempts to deal with it by using approximate smoothing functions or hill-climbing greedy techniques have been proposed, for example, [58] and the references therein. Note that the rationale behind the conditional log-likelihood is closely related to that behind conditional random fields, discussed in Section 15.4.3.

Another route in discriminative learning is to obtain the estimate of θ by *maximizing the margin*. The probabilistic class margin, for example, [21, 59], is defined as

$$\begin{aligned} d_n &= \min_{y \neq y_n} \frac{P(y_n|x_n; \theta)}{P(y|x_n; \theta)} = \frac{P(y_n|x_n; \theta)}{\max_{y \neq y_n} P(y|x_n; \theta)} \\ &= \frac{p(y_n, x_n; \theta)}{\max_{y \neq y_n} p(y, x_n; \theta)}. \end{aligned}$$

The idea is to estimate θ so as to maximize the minimum margin over all training data, that is,

$$\hat{\theta} = \arg \max_{\theta} \min(d_1, d_2, \dots, d_N).$$

The interested reader may also consult [10, 55, 60] for related reviews and methodologies.

Example 16.4. The goal of this example is to obtain the values in the conditional probability table in a general Bayesian network, which consists of l discrete random nodes/variables, x_1, x_2, \dots, x_l . We assume that all the involved variables can be observed and we have a training set of N observations. The maximum likelihood method will be employed. Let $x_i(n)$, $n = 1, 2, \dots, N$, denote the n th observation of the i th variable.

The joint pdf under the Bayesian network assumption is given by

$$P(x_1, \dots, x_l) = \prod_{i=1}^l P(x_i|\text{Pa}_i; \theta_i),$$

and the respective log-likelihood is

$$L(X; \theta) = \sum_{n=1}^N \sum_{i=1}^l \ln P(x_i(n)|\text{Pa}_i(n); \theta_i).$$

Assuming θ_i to be disjoint with θ_j , $i \neq j$, then optimization over each θ_i , $i = 1, 2, \dots, l$, can take place separately. This property is referred to as the *global decomposition* of the likelihood function. Thus, it suffices to perform the optimization locally on each node, that is,

$$l(\theta_i) = \sum_{n=1}^N \ln P(x_i(n)|\text{Pa}_i(n); \theta_i), \quad i = 1, 2, \dots, l. \quad (16.64)$$

Let us now focus on the case where all the involved variables are discrete, and the unknown quantities at any node, i , are the values of the conditional probabilities in the respective conditional probability table. For notational convenience, denote as \mathbf{h}_i the vector comprising the state indices of the parent variables of x_i . Then, the respective (unknown) probabilities are denoted as $P_{x_i|\mathbf{h}_i}(x_i, \mathbf{h}_i)$, for all possible combinations of values of x_i and \mathbf{h}_i . For example, if all the involved variables are binary and x_i has two

parent nodes, then $P_{x_i|\mathbf{h}_i}(x_i, \mathbf{h}_i)$ can take a total of eight values that have to be estimated. Equation (16.64) can now be rewritten as

$$l(\boldsymbol{\theta}_i) = \sum_{\mathbf{h}_i} \sum_{x_i} s(x_i, \mathbf{h}_i) \ln P_{x_i|\mathbf{h}_i}(x_i, \mathbf{h}_i), \quad (16.65)$$

where $s(x_i, \mathbf{h}_i)$ is the number of times the specific combination of (x_i, \mathbf{h}_i) appeared in the N samples of the training set. We assume that N is large enough so that all possible combinations occurred at least once, that is, $s(x_i, \mathbf{h}_i) \neq 0, \forall (x_i, \mathbf{h}_i)$. All one has to do now is to maximize (16.65) with respect to $P_{x_i|\mathbf{h}_i}(\cdot, \cdot)$, taking into account that

$$\sum_{x_i} P_{x_i|\mathbf{h}_i}(x_i, \mathbf{h}_i) = 1.$$

Note that $P_{x_i|\mathbf{h}_i}$ are independent for different values of \mathbf{h}_i . Thus, maximization of (16.65) can take place separately for each \mathbf{h}_i , and it is straightforward to see that

$$\hat{P}_{x_i|\mathbf{h}_i} = \frac{s(x_i, \mathbf{h}_i)}{\sum_{x_i} s(x_i, \mathbf{h}_i)}. \quad (16.66)$$

In words, the maximum likelihood estimate of the unknown conditional probabilities complies with our common sense; given a specific combination of the parent values, \mathbf{h}_i , $P_{x_i|\mathbf{h}_i}$ is approximated by the fraction of times the specific combination (x_i, \mathbf{h}_i) appeared in the data set, over the total number of times \mathbf{h}_i occurred (relate (16.66) to the Viterbi algorithm in [Section 16.5.2](#)). One can now see that in order to obtain good estimates, the number of training points, N , should be large enough so that each combination occurs a sufficiently large number of times. If the average number of parent nodes is large and/or the number of states is large, this poses heavy demands on the size of the training set. This is where parametrization of the conditional probabilities can prove to be very helpful.

16.7.2 LEARNING THE STRUCTURE

In the previous subsection, we considered the structure of the graph to be known and our task was to estimate the unknown parameters. We now turn our attention to learning the structure. In general, this is a much harder task. We only intend to provide a sketch of some general directions.

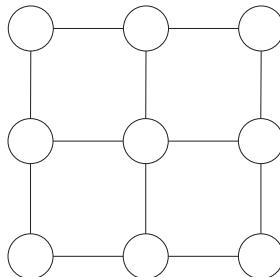
One path, known as *constrained-based*, is to try to build up a network that satisfies the data independencies, which are “measured” using different statistical tests on the training data set. The method relies a lot on intuition and such methods are not particularly popular in practice.

The other path comes under the name of *s-based* methods. This path treats the task as a typical model selection problem. The score that is chosen to be maximized provides a tradeoff between model complexity and accuracy of the fit to the data. Classical model fitting criteria such as Bayesian information criterion (BIC) and minimum description length (MDL) have been used among others. The main difficulty with all these criteria is that their optimization is an NP-hard task and the issue is to find appropriate approximate optimization schemes.

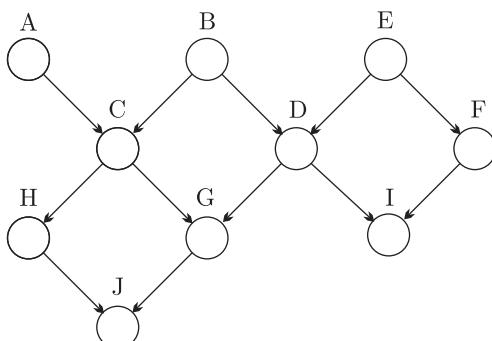
The third main path draws its existence from the Bayesian philosophy. Instead of a single structure, an ensemble of structures is employed by embedding appropriate priors into the problem. The readers who are interested in a further and deeper study are referred to more specialized books and papers, for example, [22, 41, 53].

PROBLEMS

- 16.1** Prove that an undirected graph is triangulated if and only if its cliques can be organized into a join tree.
- 16.2** For the graph of 16.3a give all possible perfect elimination sequences and draw the resulting sequence of graphs.
- 16.3** Derive the formulas for the marginal probabilities of the variables in (a) a clique node and (b) in a separator node in a junction tree.
- 16.4** Prove that in a junction tree the joint pdf of the variables is given by Eq. (16.8).
- 16.5** Show that obtaining the marginal over a single variable is independent of which one from the clique/separator nodes, which contain the variable, the marginalization is performed.
Hint. Prove it for the case of two neighboring clique nodes in the junction tree.
- 16.6** Consider the graph in Figure 16.26. Obtain a triangulated version of it.
- 16.7** Consider the Bayesian network structure given in Figure 16.27. Obtain an equivalent join tree.

**FIGURE 16.26**

The graph for the Problem 16.6.

**FIGURE 16.27**

The Bayesian network structure for Problem 16.7.

- 16.8** Consider the random variables A, B, C, D, E, F, G, H, I, J and assume that the joint distribution is given by the product of the following potential functions

$$p = \frac{1}{Z} \psi_1(A, B, C, D) \psi_2(B, E, D) \psi_3(E, D, F, I) \psi_4(C, D, G) \psi_5(C, H, G, I).$$

Construct an undirected graphical model on which the previous joint probability factorizes and in the sequence derive an equivalent junction tree.

- 16.9** Prove that the function

$$g(x) = 1 - \exp(-x), \quad x > 0,$$

is log-concave.

- 16.10** Derive the conjugate function of

$$f(x) = \ln(1 - \exp(-x)).$$

- 16.11** Show that minimizing the bound in (16.12) is a convex optimization task.

- 16.12** Show that the function $1 + \exp(-x)$, $x \in \mathbb{R}$ is log-convex and derive the respective conjugate one.

- 16.13** Derive the KL divergence between $P(\mathcal{X}^l | \mathcal{X})$ and $Q(\mathcal{X}^l)$ for the mean field Boltzmann machine and obtain the respective l variational parameters.

- 16.14** Given a distribution in the exponential family,

$$p(x) = \exp\left(\theta^T u(x) - A(\theta)\right),$$

show that $A(\theta)$ generates the respective mean parameters that define the exponential family

$$\frac{\partial A(\theta)}{\partial \theta_i} = \mathbb{E}[u_i(x)] = \mu_i.$$

Also, show that $A(\theta)$ is a convex function.

- 16.15** Show that the conjugate function of $A(\theta)$, associated with an exponential distribution such as that in Problem 16.14, is the corresponding negative entropy function. Moreover, if μ and $\theta(\mu)$ are doubly coupled, then

$$\mu = \mathbb{E}[u(x)],$$

where $\mathbb{E}[\cdot]$ is with respect to $p(x; \theta(\mu))$.

- 16.16** Derive a recursion for updating $\gamma(x_n)$ in HMMs independent on $\beta(x_n)$.

- 16.17** Derive an efficient scheme for prediction in HMM models; that is, to obtain $p(y_{N+1} | Y)$, where $Y = \{y_1, y_2, \dots, y_N\}$.

- 16.18** Prove the estimation formulas for the probabilities P_k , $k = 1, 2, \dots, K$, and P_{ij} , $i, j = 1, 2, \dots, K$, in the context of the forward-backward algorithm for training HMM.

- 16.19** Consider the Gaussian Bayesian network of Section 15.3.5 defined by the local conditional pdfs

$$p(x_i | \text{Pa}_i) = \mathcal{N}\left(x_i \mid \sum_{k: x_k \in \text{Pa}_i} \theta_{ik} x_k + \theta_{i0}, \sigma^2\right), \quad i = 1, 2, \dots, l.$$

Assume a set of N observations, $x_i(n)$, $n = 1, 2, \dots, N$, $i = 1, 2, \dots, l$, and derive a maximum likelihood estimate of the parameters θ ; assume the common variance σ^2 to be known.

REFERENCES

- [1] S. Arnborg, D. Cornell, A. Proskurowski, Complexity of finding embeddings in a k-tree, *SIAM J. Algebr. Discrete Meth* 8 (2) (1987) 277-284.
- [2] D. Barber, A.T. Cemgil, Graphical models for time series, *IEEE Signal Process. Mag.* 27 (2010) 18-28.
- [3] D. Barber, *Bayesian Reasoning and Machine Learning*, Cambridge University Press, Cambridge, 2013.
- [4] R. Bartolami, H. Bunke, Hidden Markov model-based ensemble methods for off-line handwritten text line recognition, *Pattern Recognit.* 41 (11) (2008) 3452-3460.
- [5] C.A. Berrou, A. Glavieux, P. Thitimajshima, Near Shannon limit error-correcting coding and decoding: turbo-codes, in: *Proceedings IEEE International Conference on Communications*, Geneva, Switzerland, 1993.
- [6] L. Christensen, J. Zarsen, On data and parameter estimation using the variational Bayesian EM algorithm for block-fading frequency-selective MIMO channels, in: *International Conference on Acoustics Speech and Signal Processing (ICASSP)*, vol. 4, 2006, pp. 465-468.
- [7] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, D.J. Spiegelhalter, *Probabilistic Networks and Expert Systems*, Springer-Verlag, New York, 1999.
- [8] J. Deller, J. Proakis, J.H.L. Hansen, *Discrete-Time Processing of Speech Signals*, Macmillan, New York, 1993.
- [9] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nuclear Acids*, Cambridge University Press, Cambridge, 1998.
- [10] J. Domke, Learning graphical parameters with approximate marginal inference, 2013, arXiv:1301.3193v1 [cs, LG] 15 January 2013.
- [11] Y. Ephraim, D. Malah, B.H. Juang, On the application of hidden Markov models for enhancing noisy speech, *IEEE Trans. Acoust. Speech Signal Process.* 37 (12) (1989) 1846-1856.
- [12] P. Fearhead, Exact and efficient Bayesian inference for multiple problems, *Stat. Comput.* 16 (2) (2006) 203-213.
- [13] N. Friedman, D. Geiger, M. Goldszmidt, Bayesian network classifiers, *Mach. Learn.* 29 (1997) 131-163.
- [14] K.S. Fu, *Syntactic Pattern Recognition and Applications*, Prentice Hall, Upper Saddle River, NJ, 1982.
- [15] R.G. Gallager, Low density parity-check codes, *IEEE Trans. Inform. Theory* 2 (1968) 21-28.
- [16] C. Georgoulakis, S. Theodoridis, Blind and Semi-blind equalization using hidden Markov models and clustering techniques, *Signal Process.* 80 (9) (2000) 1795-1805.
- [17] Z. Ghahramani, M.I. Jordan, Factorial hidden Markov models, *Mach. Learn.* 29 (1997) 245-273.
- [18] Z. Ghahramani, G.E. Hinton, Variational Learning for switching state space models, *Neural Comput* 12 (4) (1998) 963-996.
- [19] R. Greiner, W. Zhou, Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers, in: *Proceedings 18th International Conference on Artificial Intelligence*, 2002, pp. 167-173.
- [20] D. Grossman, P. Domingos, Learning Bayesian network classifiers by maximizing conditional likelihood, in: *Proceedings 21st International Conference on Machine Learning*, Bauff, Canada, 2004.
- [21] Y. Guo, D. Wilkinson, D. Schuurmans, Maximum margin Bayesian networks, in: *Proceedings, International Conference on Uncertainty in Artificial Intelligence*, 2005.
- [22] D. Heckerman, D. Geiger, M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, *Mach. Learn.* 20 (1995) 197-243.
- [23] B. Hu, I. Land, L. Rasmussen, R. Piton, B. Fleury, A divergence minimization approach to joint multiuser decoding for coded CDMA, *IEEE J. Select. Areas Commun.* 26 (3) (2008) 432-445.

- [24] A. Ihler, J.W. Fisher, P.L. Moses, A.S. Willsky, Nonparametric belief propagation for self-localization of sensor networks, *J. Select. Areas Commun.* 23 (4) (2005) 809-819.
- [25] A. Ihler, D. McAllester, Particle belief propagation, in: International Conference on Artificial Intelligence and Statistics, 2009, pp. 256-263.
- [26] S. Ikeda, T. Tanaka, S.I. Amari, Information geometry of turbo and low-density parity-check codes, *IEEE Trans. Inform. Theory*, 50 (6) (2004) 1097-1114.
- [27] T.S. Jaakkola, Variational Methods for Inference and Estimation in Graphical Models, PhD Thesis, Department of Brain and Cognitive Sciences, M.I.T., 1997.
- [28] T.S. Jaakkola, M.I. Jordan, Recursive algorithms for approximating probabilities in graphical models, in: M.C. Mozer, M.I. Jordan, T. Petsche (Eds.), *Proceedings in Advances in Neural Information Processing Systems*, NIPS, MIT Press, Cambridge, MA, 1997.
- [29] T.S. Jaakkola, M.I. Jordan, Improving the mean field approximation via the use of mixture distributions, in: M.I. Jordan (Ed.), *Learning in Graphical Models*, MIT Press, Cambridge, MA, 1999.
- [30] T.S. Jaakkola, M.I. Jordan, Variational methods and the QMR-DT database, *J. Artif. Intell. Res.* 10 (1999) 291-322.
- [31] T.S. Jaakkola, Tutorial on variational approximation methods, in: M. Opper, D. Saad (Eds.), *Advanced Mean Field Methods: Theory and Practice*, MIT Press, Cambridge, MA, 2001, pp. 129-160.
- [32] F.V. Jensen, *Bayesian Networks and Decision Graphs*, Springer, New York, 2001.
- [33] H. Jiang, X. Li, Parameter estimation of statistical models using convex optimization, *IEEE Signal Process. Mag.* 27 (3) (2010) 115-127.
- [34] M.I. Jordan, Z. Ghahramani, T.S. Jaakkola, L.K. Saul, An introduction to variational methods for graphical models, *Mach. Learn.* 37 (1999) 183-233.
- [35] R.E. Kalman, A new approach to linear filtering and prediction problems, *Trans. ASME J. Basic Eng.* 82 (1960) 34-45.
- [36] G.K. Kaleh, R. Vallet, Joint parameter estimation and symbol detection for linear and nonlinear channels, *IEEE Trans. Commun.* 42 (7) (1994) 2406-2414.
- [37] R. Kikuchi, The theory of cooperative phenomena, *Phys. Rev.* 81 (1951) 988-1003.
- [38] G.E. Kirkelund, C.N. Manchon, L.P.B. Christensen, E. Riegler, Variational message-passing for joint channel estimation and decoding in MIMO-OFDM, in: Proceedings, IEEE Globecom, 2010.
- [39] U. Kjærulff, Triangulation of graphs: Algorithms giving small total state space, Technical Report, R90-09, Aalborg University, Denmark, 1990.
- [40] A.P. Klapuri, A.J. Eronen, J.T. Astola, Analysis of the meter of acoustic musical signals, *IEEE Trans. Audio Speech Lang. Process.* 14 (1) (2006) 342-355.
- [41] D. Koller, N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press, Cambridge, MA, 2009.
- [42] M. Kolar, L. Song, A. Ahmed, E.P. Xing, Estimating time-varying networks, *Ann. Appl. Stat.* 4 (2010) 94-123.
- [43] J.B. Kruskal, On the shortest spanning subtree and the travelling salesman problem, *Proc. Am. Math. Soc.* 7 (1956) 48-50.
- [44] S.L. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems, *J. R. Stat. Soc. B* 50 (1988) 157-224.
- [45] S.L. Lauritzen, *Graphical Models*, Oxford University Press, Oxford, 1996.
- [46] S.E. Levinson, Continuously variable duration HMMs for automatic speech recognition, *Comput. Speech Lang.* 1 (1986) 29-45.
- [47] D.J.C. MacKay, Good error-correcting codes based on very sparse matrices, *IEEE Trans. Inform. Theory* 45 (2) (1999) 399-431.
- [48] Y. Mao, F.R. Kschischang, B. Li, S. Pasupathy, A factor graph approach to link loss monitoring in wireless sensor networks, *J. Select. Areas Commun.* 23 (4) (2005) 820-829.

- [49] R.J. McEliece, D.J.C. MacKay, J.F. Cheng, Turbo decoding as an instance of Pearl's belief propagation algorithm, *IEEE J. Select. Areas Commun.* 16 (2) (1998) 140-152.
- [50] T.P. Minka, Expectation propagation for approximate inference, in: *Proceedings 17th Conference on Uncertainty in Artificial Intelligence*, Morgan-Kaufmann, San Mateo, 2001, pp. 362-369.
- [51] T.P. Minka, Divergence measures and message passing, Technical Report MSR-TR-2005-173, Microsoft Research Cambridge, 2005.
- [52] K.P. Murphy, T. Weiss, M.J. Jordan, Loopy belief propagation for approximate inference: an empirical study, in: *Proceedings 15th Conference on Uncertainties on Artificial Intelligence*, 1999.
- [53] K.P. Murphy, *Machine Learning: A Probabilistic Perspective*, MIT Press, Cambridge, MA, 2012.
- [54] S.H. Nielsen, T.D. Nielsen, Adapting Bayesian network structures to non-stationary domains, *Int. J. Approx. Reason.* 49 (2) (2008) 379-397.
- [55] S. Nowozin, C.H. Lampert, Structured learning and prediction in computer vision, *Found. Trends Comput. Graph. Vis.* 6 (2011) 185-365.
- [56] M. Ostendorf, V. Digalakis, O. Kimball, From HMM's to segment models: a unified view of stochastic modeling for speech, *IEEE Trans. Audio Speech Process.* 4 (5) (1996) 360-378.
- [57] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan-Kaufmann, San Mateo, 1988.
- [58] F. Pernkopf, J. Bilmes, Efficient heuristics for discriminative structure learning of Bayesian network classifiers, *J. Mach. Learn. Res.* 11 (2010) 2323-2360.
- [59] F. Pernkopf, M. Wohlmayr, S. Tschiatschek, Maximum margin Bayesian network classifiers, *IEEE Trans. Pattern Anal. Mach. Intell.* 34 (3) (2012) 521-532.
- [60] F. Pernkopf, R. Peharz, S. Tschiatschek, Introduction to probabilistic graphical models, in: R. Chellappa, S. Theodoridis (Eds.), *E-Reference in Signal Processing*, vol. 1, 2013.
- [61] A. Pikrakis, S. Theodoridis, D. Kamarotos, Recognition of musical patterns using hidden Markov models, *IEEE Trans. Audio Speech Lang. Process.* 14 (5) (2006) 1795-1807.
- [62] Y. Qi, J.W. Paisley, L. Carin, Music analysis using hidden Markov mixture models, *IEEE Trans. Signal Process.* 55 (11) (2007) 5209-5224.
- [63] L. Rabiner, A tutorial on hidden Markov models and selected applications in speech processing, *Proc. IEEE* 77 (1989) 257-286.
- [64] L. Rabiner, B.H. Juang, *Fundamentals of Speech Recognition*, Prentice Hall, Upper Saddle River, NJ, 1993.
- [65] E. Riegler, G.E. Kirkeland, C.N. Manchon, M.A. Bodin, B.H. Fleury, Merging belief propagation and the mean field approximation: a free energy approach, 2012, arXiv:1112.0467v2 [cs.IT].
- [66] J.W. Robinson, A.J. Hartemink, Learning nonstationary dynamic Bayesian networks, *J. Mach. Learn. Res.* 11 (2010) 3647-3680.
- [67] T. Roos, H. Wertig, P. Grunvald, P. Myllmaki, H. Tirvi, On discriminative Bayesian network classifiers and logistic regression, *Mach. Learn.* 59 (2005) 267-296.
- [68] M.J. Russell, R.K. Moore, Explicit modeling of state occupancy in HMMs for automatic speech recognition, in: *Proceedings of the Intranational Conference on Acoustics, Speech and Signal processing, ICASSP*, vol. 1, 1985, pp. 5-8.
- [69] L.K. Saul, M.I. Jordan, A mean field learning algorithm for unsupervised neural networks, in: M.I. Jordan (Ed.), *Learning in Graphical Models*, MIT Press, Cambridge, MA, 1999.
- [70] Z. Shi, C. Schlegel, Iterative multiuser detection and error control coding in random CDMA, *IEEE Trans. Inform. Theory* 54 (5) (2006) 1886-1895.
- [71] R. Tarjan, M. Yanakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, *SIAM J. Comput.* 13 (3) (1984) 566-579.
- [72] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, fourth ed., Academic Press, Boston, 2009.
- [73] J.A. Vlontzos, S.Y. Kung, Hidden Markov models for character recognition, *IEEE Trans. Image Process.* 14 (4) (1992) 539-543.

- [74] M.J. Wainwright, T.S. Jaakkola, A.S. Willsky, A new class of upper bounds on the log partition function, *IEEE Trans. Inform. Theory* 51 (7) (2005) 2313-2335.
- [75] M.J. Wainwright, M.I. Jordan, A variational principle for graphical models, in: S. Haykin, J. Principe, T. Sejnowski, J. McWhirter (Eds.), *New Directions in Statistical Signal Processing*, MIT Press, Cambridge, MA, 2005.
- [76] M.J. Wainwright, Sparse graph codes for side information and binning, *IEEE Signal Process. Mag.* 24 (5) (2007) 47-57.
- [77] M.J. Wainwright, M.I. Jordan, Graphical models, exponential families, and variational inference, *Found. Trends Mach. Learn.* 1 (1-2) (2008) 1-305.
- [78] J.M. Walsh, P.A. Regalia, Belief propagation, Dykstra's algorithm, and iterated information projections, *IEEE Trans. Inform. Theory* 56 (8) (2010) 4114-4128.
- [79] Z. Wang, E.E. Kuruoglu, X. Yang, T. Xu, T.S. Huang, Time varying dynamic Bayesian network for nonstationary events modeling and online inference, *IEEE Trans. Signal Process.* 59 (2011) 1553-1568.
- [80] W. Wiegerinck, Variational approximations between mean field theory and the junction tree algorithm, in: *Proceedings 16th Conference on Uncertainty in Artificial Intelligence*, 2000.
- [81] C.K.I. Williams, G.E. Hinton, Mean field networks that learn to discriminate temporally distorted strings, in: D.S. Touretzky, J.L. Elman, T.J. Sejnowski, G.E. Hinton (Eds.), *Proceedings of 1990 Connectionist Models Summer School*, Morgan-Kauffman, San Mateo, CA, 1991.
- [82] J. Win, C.M. Bishop, Variational message passing, *J. Mach. Learn. Res.* 6 (2005) 661-644.
- [83] J. Yedidia, W.T. Freeman, T. Weiss, Generalized belief propagation, in: *Advances on Neural Information Processing System, NIPS*, MIT Press, Cambridge, MA, 2001, pp. 689-695.
- [84] J. Yedidia, W.T. Freeman, T. Weiss, Understanding belief propagation and its generalization, Technical Report TR-2001-22, Mitsubishi Electric Research Laboratories, 2001.
- [85] A.L. Yuille, CCCP algorithms to minimize the Bethe and Kikuchi free energies: convergent alternatives to belief propagation, *Neural Comput.* 14 (7) (2002) 1691-1722.

This page intentionally left blank

PARTICLE FILTERING

17

CHAPTER OUTLINE

17.1	Introduction	855
17.2	Sequential Importance Sampling	855
17.2.1	Importance Sampling Revisited	856
17.2.2	Resampling	857
17.2.3	Sequential Sampling	859
17.3	Kalman and Particle Filtering	861
17.3.1	Kalman Filtering: A Bayesian Point of View	862
17.4	Particle Filtering	864
17.4.1	Degeneracy	868
17.4.2	Generic Particle Filtering	870
17.4.3	Auxiliary Particle Filtering	872
Problems.....		878
<i>MATLAB Exercises</i>		881
References.....		882

17.1 INTRODUCTION

This chapter is a follow-up to [Chapter 14](#), whose focus was on Monte Carlo methods. Our interest now turns to a special type of sampling techniques known as sequential-sampling methods. In contrast to the Monte Carlo methods, considered in [Chapter 14](#), here we will assume that distributions from which we want to sample are time varying, and that sampling will take place in a sequential fashion. The main emphasis of this chapter is on particle filtering techniques for inference in state-space dynamic models. In contrast to the classical form of Kalman filtering, here the model is allowed to be nonlinear and/or the distributions associated with the involved variables non-Gaussians.

17.2 SEQUENTIAL IMPORTANCE SAMPLING

Our interest in this section shifts toward tasks where data are sequentially arriving, and our goal becomes that of sampling from their joint distribution. In other words, we are receiving observations, $\mathbf{x}_n \in \mathbb{R}^l$, of random vectors \mathbf{x}_n . At some time n , let $\mathbf{x}_{1:n} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ denote the set of the available samples

and let the respective joint distribution be denoted as $p_n(\mathbf{x}_{1:n})$. No doubt, this new task is to be treated with special care. Not only is the dimensionality of the task (number of random variables, i.e., $\mathbf{x}_{1:n}$) now time varying, but also, after some time has elapsed, the dimensionality will be very large and, in general, we expect the corresponding distribution, $p_n(\mathbf{x}_{1:n})$, to be of a rather complex form. Moreover, at time instant n , even if we knew how to sample from $p_n(\mathbf{x}_{1:n})$, the required time for sampling would be at least of the order of n . Hence, as n increases, even such a case could not be computationally feasible for large values of n . Sequential sampling is of particular interest in dynamic systems in the context of particle filtering, and we will come to deal with such systems very soon. Our discussion will develop around the importance sampling method, which was introduced in [Section 14.5](#).

17.2.1 IMPORTANCE SAMPLING REVISITED

Recall from Eq. (14.28) that given (a) a function $f(\mathbf{x})$, (b) a desired distribution $p(\mathbf{x}) = \frac{1}{Z}\phi(\mathbf{x})$, and (c) a proposal distribution $q(\mathbf{x})$, then

$$\mathbb{E}[f(\mathbf{x})] := \mu_f \simeq \sum_{i=1}^N W(\mathbf{x}_i)f(\mathbf{x}_i) := \hat{\mu}, \quad (17.1)$$

where \mathbf{x}_i are samples drawn from $q(\mathbf{x})$. Recall, also, that the estimate

$$\hat{Z} = \frac{1}{N} \sum_{i=1}^N w(\mathbf{x}_i) \quad (17.2)$$

defines an unbiased estimator of the true normalizing constant Z , where $w(\mathbf{x}_i)$ are the nonnormalized weights, $w(\mathbf{x}_i) = \frac{\phi(\mathbf{x}_i)}{q(\mathbf{x}_i)}$. Note that the approximation in Eq. (17.1) equivalently implies the following approximation of the desired distribution:

$$p(\mathbf{x}) \simeq \sum_{i=1}^N W(\mathbf{x}_i)\delta(\mathbf{x} - \mathbf{x}_i) : \quad \text{Discrete Random Measure Approximation.} \quad (17.3)$$

In other words, even a continuous pdf is approximated by a set of discrete points and weights assigned to them. We say that the distribution is approximated by a *discrete random measure* defined by the particles \mathbf{x}_i , $i = 1, 2, \dots, N$, with respective normalized weights $W(\mathbf{x}_i) := W^{(i)}$. The approximating random measure is denoted as $\chi = \{\mathbf{x}_i, W^{(i)}\}_{i=1}^N$.

Also, we have already commented in [Section 14.5](#) that a major drawback of importance sampling is the large variance of the weights, which becomes more severe in high-dimensional spaces, where our interest will be from now on. Let us elaborate on this variance problem a bit more and seek ways to bypass/reduce this undesired behavior.

It can be shown (e.g., [33] and [Problem 17.1](#)) that the variance of the corresponding estimator, $\hat{\mu}$, in Eq. (17.1) is given by

$$\text{var}[\hat{\mu}] = \frac{1}{N} \left(\int \frac{f^2(\mathbf{x})p^2(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} - \mu_f^2 \right). \quad (17.4)$$

Observe that if the numerator $f^2(\mathbf{x})p^2(\mathbf{x})$ tends to zero slower than $q(\mathbf{x})$ does, then for fixed N , the variance $\text{var}[\hat{\mu}] \rightarrow \infty$. This demonstrates the significance of selecting q very carefully. It is not difficult

to see, by minimizing Eq. (17.4), that the optimal choice for $q(\mathbf{x})$, leading to the minimum (zero) variance, is proportional to the product $f(\mathbf{x})p(\mathbf{x})$. We will make use of this result later on. Note, of course, that the proportionality constant is $1/\mu_f$, which is not known. Thus, this result can only be considered as a benchmark.

Concerning the variance issue, let us turn our attention to the unbiased estimator \hat{Z} of Z in Eq. (17.2). It can be shown (Problem 17.2) that

$$\text{var}[\hat{Z}] = \frac{Z^2}{N} \left(\int \frac{p^2(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} - 1 \right). \quad (17.5)$$

By its definition, the variance of \hat{Z} is directly related to the variance of the weights. It turns out that, in practice, the variance in Eq. (17.5) exhibits an exponential dependence on the dimensionality (e.g., [11, 15] and Problem 17.5). In such cases, the number of samples, N , has to be excessively large in order to keep the variance relatively small. One way to *partially* cope with the variance-related problem is the *resampling* technique.

17.2.2 RESAMPLING

Resampling is a very intuitive approach where one attempts a *randomized* pruning of the available samples (particles), drawn from q , by (most likely) discarding those associated with low weights and replacing them with samples whose weights have larger values. This is achieved by drawing samples from the approximation of $p(\mathbf{x})$, denoted as $\hat{p}(\mathbf{x})$, which is based on the discrete random measure $\{\mathbf{x}_i, W^{(i)}\}_{i=1}^N$, in Eq. (17.3). In importance sampling, the involved particles are drawn from $q(\mathbf{x})$ and the weights are appropriately computed in order to “match” the desired distribution. Adding the extra step of resampling, a *new* set of *unweighted* samples is drawn from the discrete approximation \hat{p} of p . Using the resampling step, we still obtain samples that are approximately distributed as p ; moreover, particles of low weights have been removed with high probability and thereby, for the next time instant, the probability of exploring regions with larger probability masses is increased. There are different ways of sampling from a discrete distribution.

- *Multinomial resampling.* This method is equivalent to the one presented in Example 14.2. Each particle, \mathbf{x}_i , is associated with a probability $P_i = W^{(i)}$. Redrawing N (new) particles will generate from each particle, \mathbf{x}_i , $N^{(i)}$ “offsprings” ($\sum_{i=1}^N N^{(i)} = N$), depending on their respective probability P_i . Hence, $N^{(1)}, \dots, N^{(N)}$, will follow a multinomial distribution (Section 2.3), that is,

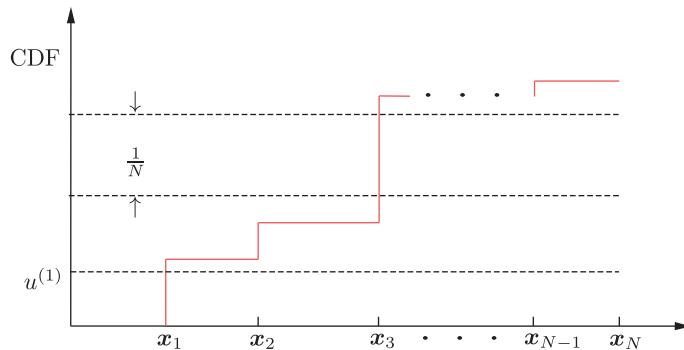
$$P(N^{(1)}, \dots, N^{(N)}) = \binom{N}{N^{(1)} \dots N^{(N)}} \prod_{i=1}^N P_i^{N^{(i)}}.$$

In this way, the higher the probability (weight $W^{(i)}$) of an originally drawn particle, the higher the number of times, $N^{(i)}$, this particle will be redrawn.

The new discrete estimate of the desired distribution will now be given by

$$\bar{p}(\mathbf{x}) = \sum_{i=1}^N \frac{N^{(i)}}{N} \delta(\mathbf{x} - \mathbf{x}_i).$$

From the properties of the multinomial distribution, we have that $\mathbb{E}[N^{(i)}] = NP_i = NW^{(i)}$ and hence $\bar{p}(\mathbf{x})$ is an unbiased approximation of $\hat{p}(\mathbf{x})$.

**FIGURE 17.1**

The sample $u^{(1)}$ drawn from $\mathcal{U}\left(0, \frac{1}{N}\right)$ determines the first point that defines the set of N equidistant lines, which are drawn and cut across the cumulative distribution function (CDF). The respective intersections determine the number of times, $N^{(i)}$, the corresponding particle, x_i , will be represented in the set. For the case of the figure, x_1 will appear once, x_2 is missed, x_3 two times.

- *Systematic resampling.* Systematic resampling is a variant of the multinomial approach. Recall from [Example 14.2](#) that every time a particle is to be (re)drawn, a new sample is generated from the uniform distribution $\mathcal{U}(0, 1)$. In contrast, in systematic resampling, the process is not entirely random. To generate N particles, we only select randomly one sample, $u^{(1)} \sim \mathcal{U}(0, \frac{1}{N})$. Then, define

$$u^{(j)} = u^{(1)} + \frac{j-1}{N}, \quad j = 2, 3, \dots, N,$$

and set

$$N^{(i)} = \text{card} \left\{ \text{All } j : \sum_{k=1}^{i-1} W^{(k)} \leq u^{(j)} < \sum_{k=1}^i W^{(k)} \right\},$$

where $\text{card}\{\cdot\}$ denotes the cardinality of the respective set. [Figure 17.1](#) illustrates the method. The resampling algorithm is summarized next.

Algorithm 17.1 (Resampling).

- Initialization
 - Input the samples x_i and respective weights $W^{(i)}$, $i = 1, 2, \dots, N$.
 - $c_0 = 0$, $N^{(i)} = 0$, $i = 1, 2, \dots, N$.
- **For** $i = 1, 2, \dots, N$, **Do**
 - $c_i = c_{i-1} + W^{(i)}$; Construct CDF.
- **End For**
- Draw $u^{(1)} \sim \mathcal{U}(0, \frac{1}{N})$
- $i = 1$
- **For** $j = 1, 2, \dots, N$, **Do**
 - $u^{(j)} = u^{(1)} + \frac{j-1}{N}$

- **While** $u^{(j)} > c_i$
 - $i = i + 1$
- **End While**
- $\bar{x}_j = \mathbf{x}_i$; Assign sample.
- $N^{(i)} = N^{(i)} + 1$
- **End For**

The output comprises the new samples $\bar{x}_j, j = 1, 2, \dots, N$, and all the weights are set equal to $\frac{1}{N}$. The sample \mathbf{x}_i will now appear, after resampling, $N^{(i)}$ times.

The previously stated two resampling methods are not the only possibilities (see, e.g., [12]). However, the systematic resampling is the one that is usually adopted due mainly to its easy implementation. Systematic resampling was introduced in [25].

Resampling schema result in estimates that converge to their true values, as long as the number of particles tends to infinity ([Problem 17.3](#)).

17.2.3 SEQUENTIAL SAMPLING

Let us now apply the experience we have gained in importance sampling to the case of sequentially arriving particles. The first examples of such techniques date back to the fifties (e.g., [19, 36]). At time n , our interest is to draw samples from the joint distribution

$$p_n(\mathbf{x}_{1:n}) = \frac{\phi_n(\mathbf{x}_{1:n})}{Z_n}, \quad (17.6)$$

based on a proposal distribution $q_n(\mathbf{x}_{1:n})$, where Z_n is the normalizing constant at time n . However, we are going to set the same goal that we have adopted for any time-recursive setting throughout this book, that is, to keep computational complexity *fixed, independent* of the time instant, n . Such a rationale dictates a *time-recursive computation* of the involved quantities. To this end, we select a proposal distribution of the form

$$q_n(\mathbf{x}_{1:n}) = q_{n-1}(\mathbf{x}_{1:n-1})q_n(\mathbf{x}_n|\mathbf{x}_{1:n-1}). \quad (17.7)$$

From Eq. (17.7), it is readily seen that

$$q_n(\mathbf{x}_{1:n}) = q_1(\mathbf{x}_1) \prod_{k=2}^n q_k(\mathbf{x}_k|\mathbf{x}_{1:k-1}). \quad (17.8)$$

This means that one has only to choose $q_k(\mathbf{x}_k|\mathbf{x}_{1:k-1}), k = 2, 3, \dots, n$, together with the initial (prior) $q_1(\mathbf{x}_1)$. Note that the dimensionality of the involved random vector in $q_k(\cdot|\cdot)$, given the past, remains *fixed* for all time instants. Equation (17.8), viewed from another angle, reveals that in order to draw a single (multivariate) sample that spans the time interval up to time n , that is, $\mathbf{x}_{1:n}^{(i)} = \{\mathbf{x}_1^{(i)}, \mathbf{x}_2^{(i)}, \dots, \mathbf{x}_n^{(i)}\}$, we build it up *recursively*; we first draw $\mathbf{x}_1^{(i)} \sim q_1(\mathbf{x})$ and then draw $\mathbf{x}_k^{(i)} \sim q_k(\mathbf{x}|\mathbf{x}_{1:k-1}), k = 2, 3, \dots, n$. The corresponding nonnormalized weights are also computed recursively [15]. Indeed,

$$\begin{aligned} w_n(\mathbf{x}_{1:n}) &:= \frac{\phi_n(\mathbf{x}_{1:n})}{q_n(\mathbf{x}_{1:n})} = \frac{\phi_{n-1}(\mathbf{x}_{1:n-1})}{q_{n-1}(\mathbf{x}_{1:n-1})} \frac{\phi_n(\mathbf{x}_{1:n})}{\phi_{n-1}(\mathbf{x}_{1:n-1})} \\ &= \frac{\phi_{n-1}(\mathbf{x}_{1:n-1})}{q_{n-1}(\mathbf{x}_{1:n-1})} \frac{\phi_n(\mathbf{x}_{1:n})}{\phi_{n-1}(\mathbf{x}_{1:n-1})q_n(\mathbf{x}_n|\mathbf{x}_{1:n-1})} \end{aligned}$$

$$\begin{aligned}
&= w_{n-1}(\mathbf{x}_{1:n-1}) a_n(\mathbf{x}_{1:n}) \\
&= w_1(\mathbf{x}_1) \prod_{k=2}^n a_k(\mathbf{x}_{1:k}),
\end{aligned} \tag{17.9}$$

where

$$a_k(\mathbf{x}_{1:k}) := \frac{\phi_k(\mathbf{x}_{1:k})}{\phi_{k-1}(\mathbf{x}_{1:k-1}) q_k(\mathbf{x}_k | \mathbf{x}_{1:k-1})}, \quad k = 2, 3, \dots, n. \tag{17.10}$$

The question that is now raised is how to choose $q_n(\mathbf{x}_n | \mathbf{x}_{1:n-1})$, $n = 2, 3, \dots$. A sensible strategy is to select it in order to minimize the variance of the weight $w_n(\mathbf{x}_{1:n})$, given the samples $\mathbf{x}_{1:n-1}$. It turns out that the optimal value, which actually makes the variance zero ([Problem 17.4](#)), is given by

$$q_n^{opt}(\mathbf{x}_n | \mathbf{x}_{1:n-1}) = p_n(\mathbf{x}_n | \mathbf{x}_{1:n-1}) : \text{Optimal Proposal Distribution.} \tag{17.11}$$

However, most often in practice, $p_n(\mathbf{x}_n | \mathbf{x}_{1:n-1})$ is not easy to sample and one has to be content with adopting some approximation of it. We are now ready to state our first algorithm for sequential sampling.

Algorithm 17.2 (Sequential importance sampling (SIS)).

- Select $q_1(\cdot)$, $q_n(\cdot | \cdot)$, $n = 2, 3, \dots$
- Select number of particles, N .
- **For** $i = 1, 2, \dots, N$, **Do**; Initialize N different realizationsstreams.
 - Draw $\mathbf{x}_1^{(i)} \sim q_1(\mathbf{x})$
 - Compute the weights $w_1(\mathbf{x}_1^{(i)}) = \frac{\phi_1(\mathbf{x}_1^{(i)})}{q_1(\mathbf{x}_1^{(i)})}$
- **End For**
- **For** $i = 1, 2, \dots, N$, **Do**
 - Compute the normalized weights $W_1^{(i)}$.
- **End For**
- **For** $n = 2, 3, \dots$, **Do**
 - **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x}_n^{(i)} \sim q_n(\mathbf{x} | \mathbf{x}_{1:n-1}^{(i)})$
 - Compute the weights

$$w_n(\mathbf{x}_{1:n}^{(i)}) = w_{n-1}(\mathbf{x}_{1:n-1}^{(i)}) a_n(\mathbf{x}_{1:n}^{(i)}); \text{ from Eq. (17.10).}$$

- **End For**
- **For** $i = 1, 2, \dots, N$, **Do**
 - $W_n^{(i)} \propto w_n(\mathbf{x}_{1:n}^{(i)})$
- **End For**
- **End For**

Once the algorithm has been completed, we can write

$$\hat{p}_n(\mathbf{x}_{1:n}) = \sum_{i=1}^N W_n^{(i)} \delta(\mathbf{x}_{1:n} - \mathbf{x}_{1:n}^{(i)}).$$

However, as we have already said, the variance of the weights has the tendency to increase with n ([Problem 17.5](#)). Thus, the resampling version of the sequential importance sampling is usually employed.

Algorithm 17.3 (SIS with resampling).

- Select $q_1(\cdot)$, $q_n(\cdot|\cdot)$, $n = 1, 2, \dots$
- Select number of particles N .
- **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x}_1^{(i)} \sim q_1(\mathbf{x})$
 - Compute the weights $w_1(\mathbf{x}_1^{(i)}) = \frac{\phi_1(\mathbf{x}_1^{(i)})}{q_1(\mathbf{x}_1^{(i)})}$.
- **End For**
- **For** $i = 1, \dots, N$, **Do**
 - Compute the normalized weights $W_1^{(i)}$
- **End For**
- Resample $\{\mathbf{x}_1^{(i)}, W_1^{(i)}\}_{i=1}^N$ to obtain $\{\bar{\mathbf{x}}_1^{(i)}, \frac{1}{N}\}_{i=1}^N$, using [Algorithm 17.1](#)
- **For** $n = 2, 3, \dots$, **Do**
 - **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x}_n^{(i)} \sim q_n(\mathbf{x}|\bar{\mathbf{x}}_{1:n-1}^{(i)})$
 - Set $\mathbf{x}_{1:n}^{(i)} = \{\mathbf{x}_n^{(i)}, \bar{\mathbf{x}}_{1:n-1}^{(i)}\}$
 - Compute $w_n(\mathbf{x}_{1:n}^{(i)}) = \frac{1}{N} a_n(\mathbf{x}_{1:n}^{(i)})$; (Eq. (17.10)).
 - **End For**
 - **For** $i = 1, 2, \dots, N$, **Do**
 - Compute $W_n^{(i)}$
 - **End Do**
 - Resample $\{\mathbf{x}_{1:n}^{(i)}, W_n^{(i)}\}_{i=1}^N$ to obtain $\{\bar{\mathbf{x}}_{1:n}^{(i)}, \frac{1}{N}\}_{i=1}^N$
- **End For**

Remarks 17.1.

- Convergence results concerning sequential importance sampling can be found in, for example, [5–7]. It turns out that, in practice, the use of resampling leads to substantially smaller variances.
- From a practical point of view, sequential importance methods with resampling are expected to work reasonably well, if the desired successive distributions at different time instants do not differ much and the choice of $q_n(\mathbf{x}_n|\mathbf{x}_{1:n-1})$ is *close to the optimal* one (see, e.g., [15]).

17.3 KALMAN AND PARTICLE FILTERING

Particle filtering is an instance of the sequential Monte Carlo methods. Particle filtering is a technique born in the 1990s and it was first introduced in [18] as an attempt to solve estimation tasks in the context of state-space modeling for the more general nonlinear and non-Gaussian scenarios. The term “particle filtering” was coined in [3], although the term “particle” had been used in [25].

Hidden Markov models, which are treated in [Section 16.4](#), and Kalman filters, treated in [Chapter 4](#), are special types of the state-space (state-observation) modeling. The former address the case of discrete state (latent) variables and the latter the continuous case, albeit at the very special case of linear and Gaussian scenario. In particle filtering, the interest shifts to models of the following form:

$$\mathbf{x}_n = \mathbf{f}_n(\mathbf{x}_{n-1}, \boldsymbol{\eta}_n) : \text{ State Equation} \quad (17.12)$$

$$\mathbf{y}_n = \mathbf{h}_n(\mathbf{x}_n, \mathbf{v}_n) : \text{ Observations Equation,} \quad (17.13)$$

where $\mathbf{f}_n(\cdot, \cdot)$ and $\mathbf{h}_n(\cdot, \cdot)$ are nonlinear, in general, (vector) functions; $\boldsymbol{\eta}_n$ and \mathbf{v}_n are noise sequences; and the dimensions of \mathbf{x}_n and \mathbf{y}_n can be different. The random vector \mathbf{x}_n is the (latent) state vector and \mathbf{y}_n corresponds to the observations. There are two inference tasks that are of interest in practice.

Filtering: Given the set of measurements, $\mathbf{y}_{1:n}$, in the time interval $[1, n]$, compute

$$p(\mathbf{x}_n | \mathbf{y}_{1:n}).$$

Smoothing: Given the set of measurements $\mathbf{y}_{1:N}$ in a time interval $[1, N]$, compute

$$p(\mathbf{x}_n | \mathbf{y}_{1:N}), \quad 1 \leq n \leq N.$$

Before we proceed to our main goal, let us review the simpler case, that of Kalman filters, this time from a Bayesian viewpoint.

17.3.1 KALMAN FILTERING: A BAYESIAN POINT OF VIEW

Kalman filtering was first discussed in [Section 4.10](#) in the context of linear estimation methods and the mean-square error criterion. In the current section, the Kalman filtering algorithm will be rederived following concepts from the theory of graphical models and Bayesian networks, which are treated in [Chapters 15](#) and [16](#). This probabilistic view will then be used for the subsequent nonlinear generalizations in the framework of particle filtering. For the linear case model, Eqs. [\(17.12\)](#) and [\(17.13\)](#) become

$$\mathbf{x}_n = F_n \mathbf{x}_{n-1} + \boldsymbol{\eta}_n, \quad (17.14)$$

$$\mathbf{y}_n = H_n \mathbf{x}_n + \mathbf{v}_n, \quad (17.15)$$

where F_n and H_n are matrices of appropriate dimensions. We further assume that the two noise sequences are statistically independent and of a Gaussian nature, that is,

$$p(\boldsymbol{\eta}_n) = \mathcal{N}(\boldsymbol{\eta}_n | \mathbf{0}, Q_n), \quad (17.16)$$

$$p(\mathbf{v}_n) = \mathcal{N}(\mathbf{v}_n | \mathbf{0}, R_n). \quad (17.17)$$

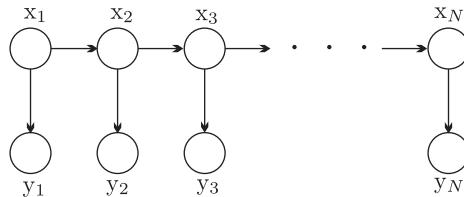
The kick-off point for deriving the associated recursions is the Bayes rule,

$$\begin{aligned} p(\mathbf{x}_n | \mathbf{y}_{1:n}) &= \frac{p(\mathbf{y}_n | \mathbf{x}_n, \mathbf{y}_{1:n-1}) p(\mathbf{x}_n | \mathbf{y}_{1:n-1})}{Z_n} \\ &= \frac{p(\mathbf{y}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{y}_{1:n-1})}{Z_n}, \end{aligned} \quad (17.18)$$

where

$$\begin{aligned} Z_n &= \int p(\mathbf{y}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{y}_{1:n-1}) d\mathbf{x}_n \\ &= p(\mathbf{y}_n | \mathbf{y}_{1:n-1}), \end{aligned} \quad (17.19)$$

and we have used the fact that $p(\mathbf{y}_n | \mathbf{x}_n, \mathbf{y}_{1:n-1}) = p(\mathbf{y}_n | \mathbf{x}_n)$, which is a consequence of [\(17.15\)](#). For those who have already read [Chapter 15](#), recall that Kalman filtering is a special case of a Bayesian network

**FIGURE 17.2**

Graphical model corresponding to the state-space modeling for Kalman and particle filters.

and corresponds to the graphical model given in Figure 17.2. Hence, due to the Markov property, \mathbf{y}_n is independent of the past given the values in \mathbf{x}_n . Moreover, note that

$$\begin{aligned} p(\mathbf{x}_n | \mathbf{y}_{1:n-1}) &= \int p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{y}_{1:n-1}) p(\mathbf{x}_{n-1} | \mathbf{y}_{1:n-1}) d\mathbf{x}_{n-1} \\ &= \int p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{n-1} | \mathbf{y}_{1:n-1}) d\mathbf{x}_{n-1}, \end{aligned} \quad (17.20)$$

where, once more, the Markov property (i.e., Eq. (17.14)) has been used.

Equations (17.18)–(17.20) comprise the set of recursions, which lead to the update

$$p(\mathbf{x}_{n-1} | \mathbf{y}_{1:n-1}) \longrightarrow p(\mathbf{x}_n | \mathbf{y}_{1:n}),$$

starting from the initial (prior) $p(\mathbf{x}_0 | \mathbf{y}_0) := p(\mathbf{x}_0)$. If $p(\mathbf{x}_0)$ is chosen to be Gaussian, then all the involved pdfs turn out to be Gaussian due to Eqs. (17.16) and (17.17), and the linearity of Eqs. (17.14) and (17.15); this makes the computational of the integrals a trivial task following the recipe rules in the Appendix in Section 12.9.

Before we proceed further, note that the recursions in Eqs. (17.18) and (17.20) are an instance of the sum-product algorithm for graphical models. Indeed, to put our current discussion in this context, let us compactly write the previous recursions as

$$p(\mathbf{x}_n | \mathbf{y}_{1:n}) = \underbrace{\frac{p(\mathbf{y}_n | \mathbf{x}_n)}{Z_n}}_{\text{corrector}} \underbrace{\int p(\mathbf{x}_{n-1} | \mathbf{y}_{1:n-1}) p(\mathbf{x}_n | \mathbf{x}_{n-1}) d\mathbf{x}_{n-1}}_{\text{predictor}} : \text{Filtering.} \quad (17.21)$$

Note that this is of exactly the same form, within the normalizing factor, as Eq. (16.43) of Chapter 16; just replace summation with integration. One can rederive Eq. (17.21) using the sum-product rule, following similar steps as for Eq. (16.43). The only difference is that the normalizing constant has to be involved in all respective definitions and we replace summations with integrations. Because all the involved pdfs are Gaussians, the computation of the involved normalizing constants is trivially done; moreover, it suffices to derive recursions only for the respective mean values and covariances.

In Eq. (17.20), we have that

$$p(\mathbf{x}_n | \mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{x}_n | F_n \mathbf{x}_{n-1}, Q_n).$$

Let, also, $p(\mathbf{x}_{n-1} | \mathbf{y}_{1:n-1})$ be Gaussian with mean and covariance matrix,

$$\boldsymbol{\mu}_{n-1|n-1}, \quad P_{n-1|n-1},$$

respectively, where the notation is chosen so that in order for the derived recursions to comply with the resulting algorithm in [Section 4.10](#). Then, according to the Appendix in [Section 12.9](#), $p(\mathbf{x}_n|\mathbf{y}_{1:n-1})$ is a Gaussian marginal pdf with mean and covariance given by (see Eqs. (12.147) and (12.148))

$$\boldsymbol{\mu}_{n|n-1} = F_n \boldsymbol{\mu}_{n-1|n-1}, \quad (17.22)$$

$$P_{n|n-1} = Q_n + F_n P_{n-1|n-1} F_n^T. \quad (17.23)$$

Also, in Eq. (17.18) we have that

$$p(\mathbf{y}_n|\mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n|H_n \mathbf{x}_n, R_n).$$

From [Section 12.9](#), and taking into account (Eqs. (17.22) and (17.23)), we get that $p(\mathbf{x}_n|\mathbf{y}_{1:n})$ is the posterior (Gaussian) with mean and covariance given by (see Eqs. (12.145) and (12.146))

$$\boldsymbol{\mu}_{n|n} = \boldsymbol{\mu}_{n|n-1} + K_n (\mathbf{y}_n - H_n \boldsymbol{\mu}_{n|n-1}), \quad (17.24)$$

$$P_{n|n} = P_{n|n-1} - K_n H_n P_{n|n-1}, \quad (17.25)$$

where

$$K_n = P_{n|n-1} H_n^T S_n^{-1}, \quad (17.26)$$

and

$$S_n = R_n + H_n P_{n|n-1} H_n^T. \quad (17.27)$$

Note that these are exactly the same recursions that were derived in [Section 4.10](#) for the state estimation; recall that under the Gaussian assumption, the posterior mean coincides with the least-squares estimate.

Here we have assumed that matrices F_n , H_n as well as the covariance matrices are known. This is most often the case. If not, these can be learned using similar arguments as those used in learning the hidden Markov model (HMM) parameters, which are discussed in [Section 16.5.2](#) (see, e.g., [2]).

17.4 PARTICLE FILTERING

In [Section 4.10](#), extended Kalman filtering (EKF) was discussed as one possibility to generalize Kalman filtering to nonlinear models. Particle filtering, to be discussed next, is a powerful alternative technique to EKF. The involved pdfs are approximated by *discrete random measures*. The underlying theory is that of sequential importance sampling (SIS); as a matter of fact, particle filtering is an instance of SIS.

Let us now consider the state-space model of the general form in Eqs. (17.12) and (17.13). From the specific form of these equations (and by the Bayesian network nature of such models, for the more familiar reader) we can write

$$p(\mathbf{x}_n|\mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}) = p(\mathbf{x}_n|\mathbf{x}_{n-1}) \quad (17.28)$$

and

$$p(\mathbf{y}_n|\mathbf{x}_{1:n}, \mathbf{y}_{1:n-1}) = p(\mathbf{y}_n|\mathbf{x}_n). \quad (17.29)$$

Our starting point is the *sequential* estimation of $p(\mathbf{x}_{1:n}|\mathbf{y}_{1:n})$; the estimation of $p(\mathbf{x}_n|\mathbf{y}_{1:n})$, which comprises our main goal, will be obtained as a by-product. Note that [15]

$$\begin{aligned} p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) &= p(\mathbf{x}_n, \mathbf{x}_{1:n-1}, \mathbf{y}_n, \mathbf{y}_{1:n-1}) \\ &= p(\mathbf{x}_n, \mathbf{y}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}) p(\mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}) \\ &= p(\mathbf{y}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) p(\mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}), \end{aligned} \quad (17.30)$$

where Eqs. (17.28) and (17.29) have been employed.

Our goal is to obtain an approximation, via the generation of particles, of the conditional pdf,

$$p(\mathbf{x}_{1:n}|\mathbf{y}_{1:n}) = \frac{p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n})}{\int p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) d\mathbf{x}_{1:n}} = \frac{p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n})}{Z_n}, \quad (17.31)$$

where

$$Z_n := \int p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) d\mathbf{x}_{1:n}.$$

To put the current discussion in the general framework of SIS, compare Eq. (17.31) with Eq. (17.6), which leads to the definition

$$\phi_n(\mathbf{x}_{1:n}) := p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n}). \quad (17.32)$$

Then, Eq. (17.9) becomes

$$w_n(\mathbf{x}_{1:n}) = w_{n-1}(\mathbf{x}_{1:n-1}) \alpha_n(\mathbf{x}_{1:n}), \quad (17.33)$$

where now

$$\alpha_n(\mathbf{x}_{1:n}) = \frac{p(\mathbf{x}_{1:n}, \mathbf{y}_{1:n})}{p(\mathbf{x}_{1:n-1}, \mathbf{y}_{1:n-1}) q_n(\mathbf{x}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n})},$$

which from Eq. (17.30) becomes

$$\alpha_n(\mathbf{x}_{1:n}) = \frac{p(\mathbf{y}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1})}{q_n(\mathbf{x}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n})}. \quad (17.34)$$

The final step is to select the proposal distribution. From Section 17.2, recall that the optimal proposal distribution is given from Eq. (17.11), which for our case takes the form

$$\begin{aligned} q_n^{opt}(\mathbf{x}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n}) &= p(\mathbf{x}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n}) \\ &= p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{x}_{1:n-2}, \mathbf{y}_n, \mathbf{y}_{1:n-1}), \end{aligned}$$

and exploiting the underlying independencies, as they are imposed by the Bayesian network structure of the state-space model, we finally get

$$q_n^{opt}(\mathbf{x}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n}) = p(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{y}_n) : \text{Optimal Proposal Distribution.} \quad (17.35)$$

The use of the optimal proposal distribution leads to the following weight update recursion (Problem 17.6):

$$\boxed{w_n(\mathbf{x}_{1:n}) = w_{n-1}(\mathbf{x}_{1:n-1}) p(\mathbf{y}_n | \mathbf{x}_{n-1}) : \text{Optimal Weights.}} \quad (17.36)$$

However, as is most often the case in practice, optimality is not always easy to obtain. Note that Eq. (17.36) requires the following integration,

$$p(\mathbf{y}_n | \mathbf{x}_{n-1}) = \int p(\mathbf{y}_n | \mathbf{x}_n) p(\mathbf{x}_n | \mathbf{x}_{n-1}) d\mathbf{x}_n,$$

which may not be tractable. Moreover, even if the integral can be computed, sampling from $p(\mathbf{y}_n | \mathbf{x}_{n-1})$ directly may not be feasible. In any case, even if the optimal proposal distribution cannot be used, we can still select the proposal distribution to be of the form

$$q_n(\mathbf{x}_n | \mathbf{x}_{1:n-1}, \mathbf{y}_{1:n}) = q(\mathbf{x}_n | \mathbf{x}_{n-1}, \mathbf{y}_n). \quad (17.37)$$

Note that such a choice is particularly convenient, because sampling at time, n , only depends on \mathbf{x}_{n-1} , and \mathbf{y}_n and *not* on the entire history. If, in addition, the goal is to obtain estimates of $p(\mathbf{x}_n | \mathbf{y}_{1:n})$, then one need not keep in memory all previously generated samples, but only the most recent one, \mathbf{x}_n .

We are now ready to write the first particle-filtering algorithm.

Algorithm 17.4 (SIS particle filtering).

- Select a prior distribution, p , to generate the initial state \mathbf{x}_0 .
- Select the number of particle streams, N .
- **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x}_0^{(i)} \sim p(\mathbf{x})$; Initialize the N streams of particles.
 - Set $w_0^{(i)} = \frac{1}{N}$; Set all initial weights equal.
- **End For**
- **For** $n = 1, 2, \dots$, **Do**
 - **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x}_n^{(i)} \sim q(\mathbf{x} | \mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n)$
 - $w_n^{(i)} = w_{n-1}^{(i)} \frac{p(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}) p(\mathbf{y}_n | \mathbf{x}_n^{(i)})}{q(\mathbf{x}_n^{(i)} | \mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n)}$; formulae (17.33), (17.34), and (17.37).
 - **End For**
 - **For** $i = 1, 2, \dots, N$, **Do**
 - Compute the normalized weights $W_n^{(i)}$
 - **End For**
- **End For**

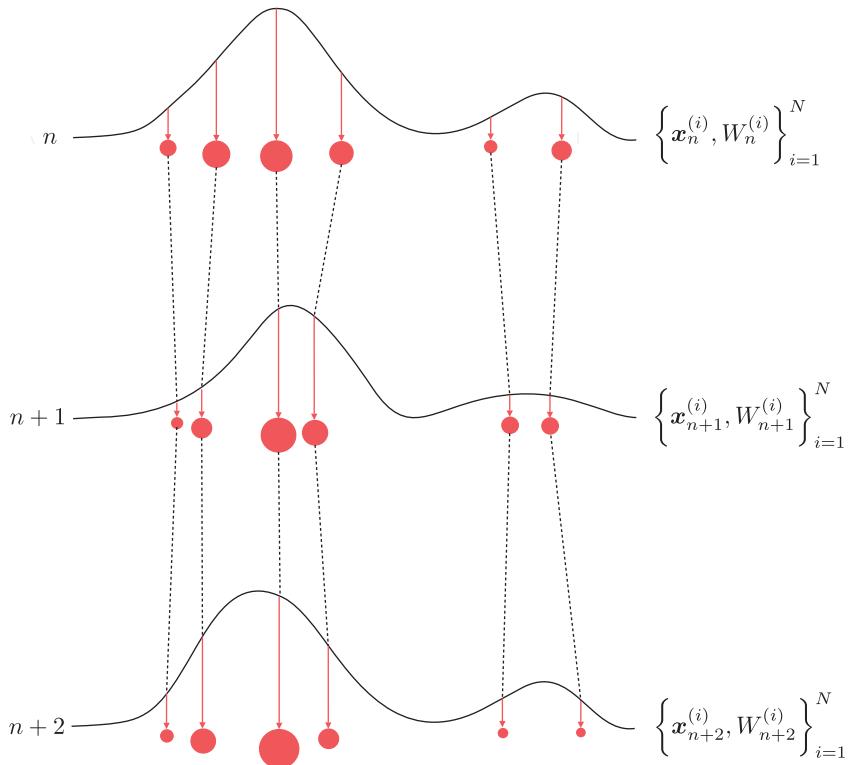
Note that the generation of the N streams of particles can take place *concurrently*, by exploiting parallel processing capabilities, if they are available in the processor.

The particles generated along the i th stream $\mathbf{x}_n^{(i)}$, $n = 1, 2, \dots$, represent a *path/trajectory* through the state-space. Once the particles have been drawn and the normalized weights computed, we obtain the estimate

$$\hat{p}(\mathbf{x}_{1:n} | \mathbf{y}_{1:n}) = \sum_{i=1}^N W_n^{(i)} \delta(\mathbf{x}_{1:n} - \mathbf{x}_{1:n}^{(i)}).$$

If, as commented earlier, our interest lies in keeping the terminal sample, $\mathbf{x}_n^{(i)}$, only, then discarding the path history, $\mathbf{x}_{1:n-1}^{(i)}$, we can write

$$\hat{p}(\mathbf{x}_n | \mathbf{y}_{1:n}) = \sum_{i=1}^N W_n^{(i)} \delta(\mathbf{x}_n - \mathbf{x}_n^{(i)}).$$

**FIGURE 17.3**

Three consecutive recursions, for the particle-filtering scheme given in [Algorithm 17.4](#), with $N = 7$ streams of particles. The area of the circles corresponds to the size of the normalized weights of the respective particles drawn from the proposal distribution.

Note that as the number of particles, N , tends to infinity, the previous approximations tend to the true posterior densities. [Figure 17.3](#) provides a graphical interpretation of the SIS algorithm [17.4](#).

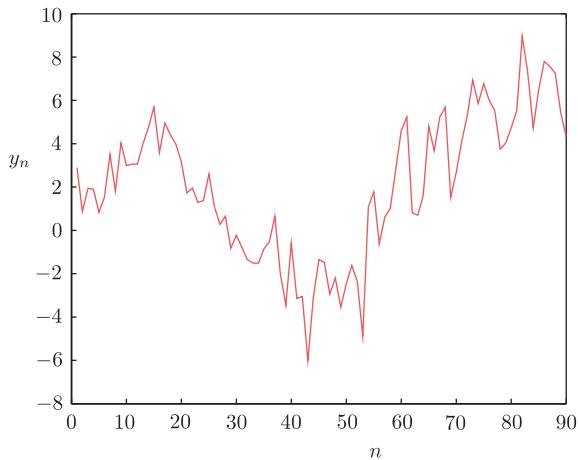
Example 17.1. Consider the one-dimensional random walk model written as

$$x_n = x_{n-1} + \eta_n, \quad (17.38)$$

$$y_n = x_n + v_n, \quad (17.39)$$

where $\eta_n \sim \mathcal{N}(\eta_n | 0, \sigma_\eta^2)$, $v_n \sim \mathcal{N}(v_n | 0, \sigma_u^2)$, with $\sigma_\eta^2 = 1$, $\sigma_u^2 = 1$. Although this is a typical task for (linear) Kalman filtering, we will attack it here via the particle-filtering rationale in order to demonstrate some of the previously reported performance-related issues. The proposal distribution is selected to be

$$q(x_n | x_{n-1}, y_n) = p(x_n | x_{n-1}) = \mathcal{N}(x_n | x_{n-1}, \sigma_\eta^2).$$

**FIGURE 17.4**

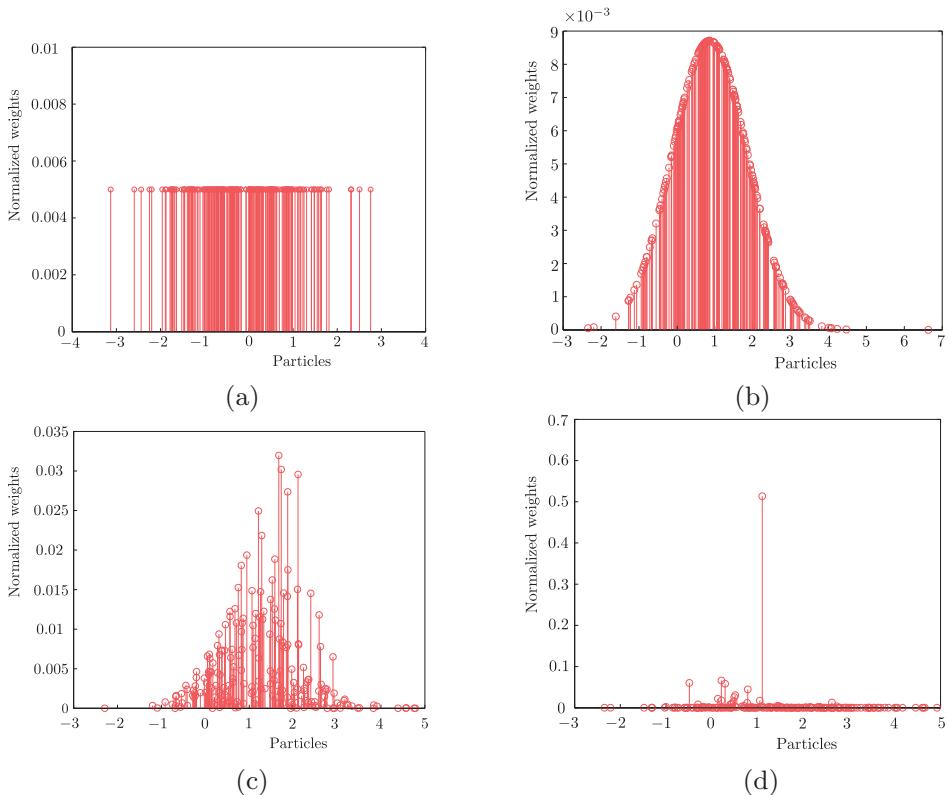
The observation sequence for Example 17.1.

1. Generate $T = 100$ observations, $y_n, n = 1, 2, \dots, T$, to be used by the [Algorithm 17.4](#). To this end, start with an arbitrary state value, for example, $x_0 = 0$ and generate a realization of the random walk, drawing samples from the Gaussians (we know how to generate Gaussian samples) $\mathcal{N}(\cdot | 0, \sigma_n^2)$ and $\mathcal{N}(\cdot | 0, \sigma_u^2)$, according to Eqs. (17.38) and (17.39). [Figure 17.4](#) shows a realization for the output variable. Our goal is to use the sequence of the resulting observations to generate particles and demonstrate the increase of the variance of the associated weights as time goes by.
2. Use $\mathcal{N}(\cdot | 0, 1)$ to initialize $N = 200$ particle streams, $x^{(i)}, i = 1, 2, \dots, N$, and initialize the normalized weights to equal values, $W_0^{(i)} = \frac{1}{N}, i = 1, 2, \dots, N$. [Figure 17.5](#) provides the corresponding plot.
3. Perform [Algorithm 17.4](#) and plot the resulting particles together with the respective weights at time instants, $n = 0, n = 1, n = 3$, and $n = 30$. Observe how the variance of the weights increases with time. At time $n = 30$ only a few particles have nonzero weights.
4. Repeat the experiment with $N = 1000$. [Figure 17.6](#) is the counterpart of [Figure 17.5](#) for the snapshots of $n = 3$ and $n = 30$. Observe that increasing the number of particles improves the performance with respect to the variance of weights. This is one path to obtain more particles with significant weight values. The other path is via resampling techniques.

17.4.1 DEGENERACY

Particle filtering is a special case of sequential importance sampling; hence, everything that has been said in [Section 17.2](#) concerning the respective performance is also applied here.

A major problem is the *degeneracy* phenomenon. The variance of the importance weights increases in time, and after a few iterations only very few (or even only one) of the particles are assigned

**FIGURE 17.5**

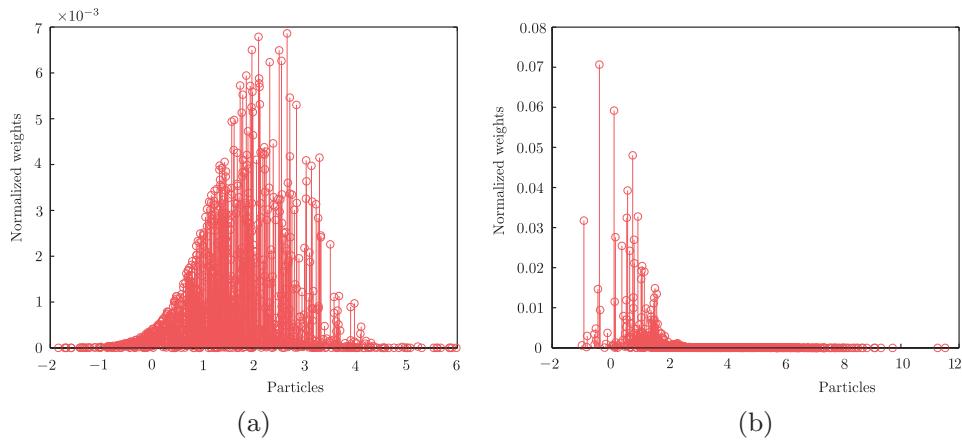
Plot of $N = 200$ generated particles with the corresponding (normalized) weights, for Example 17.1, at time instants (a) $n = 0$, (b) $n = 1$, (c) $n = 3$, and (d) $n = 30$. Observe that as time goes by, the variance of the weights increases. At time $n = 30$, only very few particles have a nonzero weight value.

nonnegligible weights, and the discrete random measure degenerates quickly. There are two methods for reducing degeneracy: one is selecting a good proposal distribution and the other is resampling.

We know the optimal choice for the proposal distribution is

$$q(\cdot | \mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n) = p(\cdot | \mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n).$$

There are cases where this is available in analytic form. For example, this happens if the noise sources are Gaussian and the observation equation is linear (e.g., [11]). If analytic forms are not available and direct sampling is not possible, approximations of $p(\cdot | \mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n)$ are mobilized. Our familiar (from Chapter 12) Gaussian approximation via local linearization of $\ln p(\cdot | \mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n)$ is a possibility [11]. The use of suboptimal filtering techniques such as the extended/unscented Kalman filter have also been advocated [37]. In general, it must be kept in mind that the choice of the proposal distribution plays a crucial role in the performance of particle filtering. Resampling is the other path that has been discussed

**FIGURE 17.6**

Plot of $N = 1000$ generated particles with the corresponding (normalized) weights, for Example 17.1, at time instants (a) $n = 3$, (b) $n = 30$. As expected, compared to Figure 17.5, more particles with significant weights survive.

in Section 17.2.2. The counterpart of Algorithm 17.1 can also be adopted for the case of particle filtering. However, we are going to give a slightly modified version of it.

17.4.2 GENERIC PARTICLE FILTERING

Resampling has a number of advantages. It discards, with high probability, particles of low weights; that is, only particles corresponding to regions of high-probability mass are propagated. Of course, resampling has its own limitations. For example, a particle of low weight at time, n , will not necessarily have a low weight at later time instants. In such a case, resampling is rather wasteful. Moreover, resampling limits the potential of parallelizing the computational process, because particles along the different streams have to be “combined” at each time instant. However, some efforts for enhancing parallelism have been reported (see, e.g., [21]). Also, particles corresponding to high values of weights are drawn many times and lead to a set of samples of low diversity; this phenomenon is also known as *sample impoverishment*. The effects of this phenomenon become more severe in cases of low state/process noise, η_n , in Eq. (17.12), where the set of the sampling points may end up comprising a single point (e.g., [1]).

Hence, avoiding resampling can be beneficial. In practice, resampling is performed only if a related metric of the variance of the weights is below a threshold. In [28, 29], the *effective number* of samples is approximated by

$$N_{eff} \approx \frac{1}{\sum_{i=1}^N (W_n^{(i)})^2}. \quad (17.40)$$

The value of this index ranges from 1 to N . Resampling is performed if $N_{eff} \leq N_T$, typically with $N_T = \frac{N}{2}$.

Algorithm 17.5 (Generic particle filtering).

- Select a prior distribution, p , to generate particles for the initial state \mathbf{x}_0 .
- Select the number of particle streams, N .
- **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x}_0^{(i)} \sim p(\mathbf{x})$; Initialize N streams.
 - set $W_0^{(i)} = \frac{1}{N}$; All initial normalized weights are equal.
- **End For**
- **For** $n = 1, 2, 3, \dots$, **Do**
 - **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x}_n^{(i)} \sim q(\mathbf{x}|\mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n)$
 - $w_n^{(i)} = w_{n-1}^{(i)} \frac{p(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)})p(\mathbf{y}_n|\mathbf{x}_n^{(i)})}{q(\mathbf{x}_n^{(i)}|\mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n)}$
 - **End For**
 - **For** $i = 1, 2, \dots, N$, **Do**
 - Compute the normalized $W_n^{(i)}$.
 - **End For**
 - Compute N_{eff} ; Eq. (17.40).
 - **If** $N_{eff} \leq N_T$; preselected value N_T .
 - Resample $\{\mathbf{x}_n^{(i)}, W_n^{(i)}\}_{i=1}^N$ to obtain $\{\bar{\mathbf{x}}_n^{(i)}, \frac{1}{N}\}_{i=1}^N$
 - $\mathbf{x}_n^{(i)} = \bar{\mathbf{x}}_n^{(i)}, w_n^{(i)} = \frac{1}{N}$
 - **End If**
- **End For**

Figure 17.7 presents a graphical illustration of the time evolution of the algorithm.

Remarks 17.2.

- A popular choice for the proposal distribution is the prior,

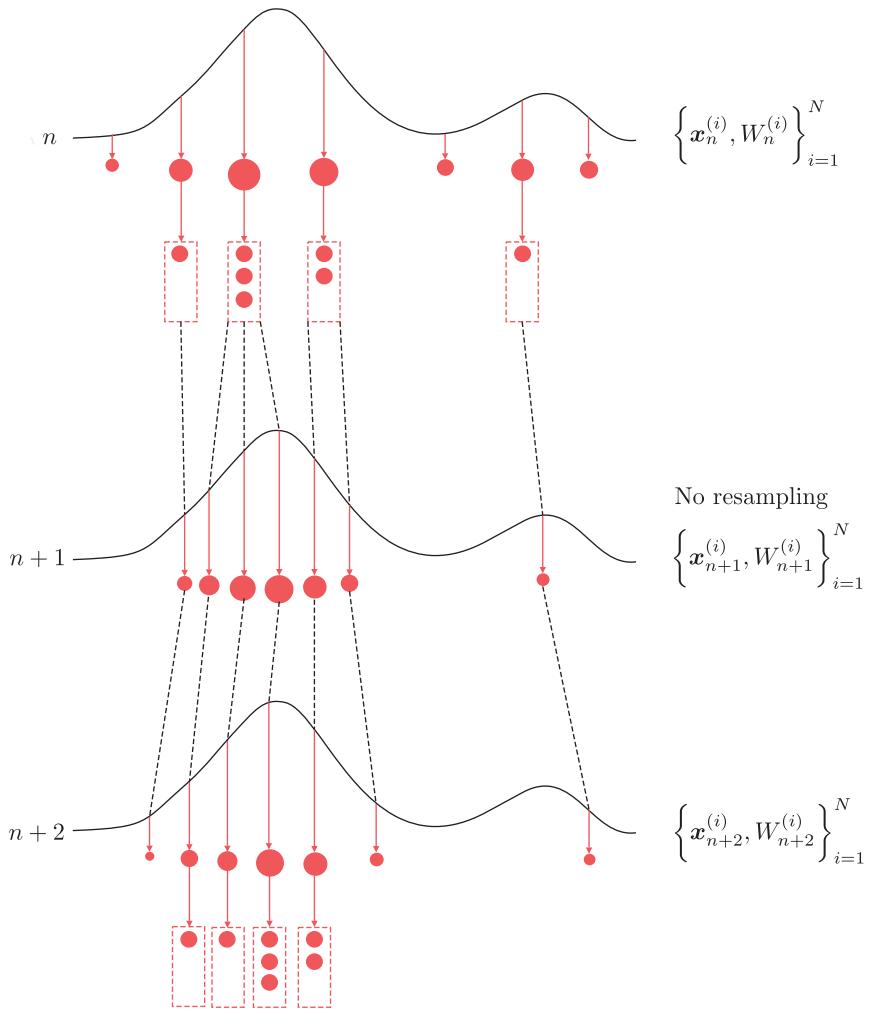
$$q(\mathbf{x}|\mathbf{x}_{n-1}^{(i)}, \mathbf{y}_n) = p(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}),$$

which yields the following weights' update recursion,

$$w_n^{(i)} = w_{n-1}^{(i)} p(\mathbf{y}_n|\mathbf{x}_n^{(i)}).$$

The resulting algorithm is known as *sampling-importance-resampling* (SIR). The great advantage of such a choice is its simplicity. However, the generation mechanism of particles ignores important information that resides in the observation sequence; the proposal distribution is independent of the observations. This may lead to poor results. A remedy can be offered by the use of auxiliary particle filtering, to be reviewed next. Another possibility is discussed in [22], via a combination of the prior and the optimal proposal distributions.

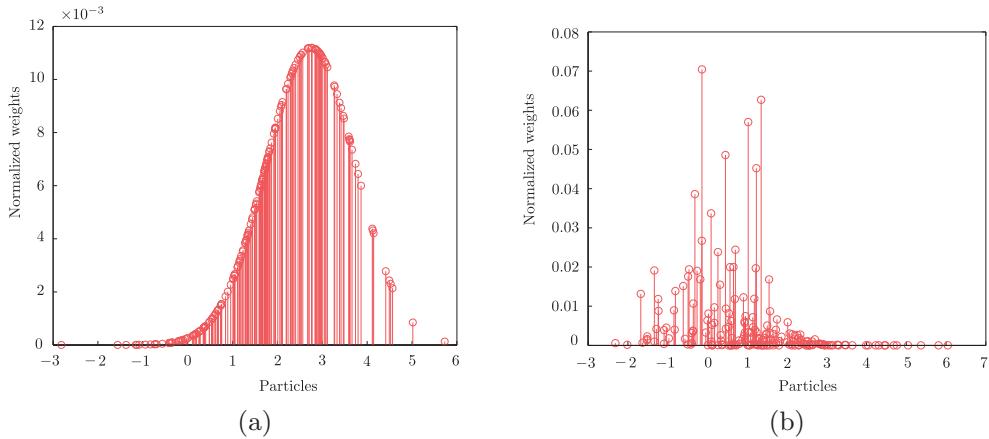
Example 17.2. Repeat example 17.1, using $N = 200$ particles, for Algorithm 17.5. Use the threshold value $N_T = 100$. Observe in Figure 17.8 that, for the corresponding time instants, more particles with significant weights are generated compared to Figure 17.5.

**FIGURE 17.7**

Three successive time iterations for $N = 7$ streams of particles corresponding to [Algorithm 17.5](#). At steps n and $n + 2$ resampling is performed. At step $n + 1$ no resampling is needed.

17.4.3 AUXILIARY PARTICLE FILTERING

Auxiliary particle filters were introduced in [34] in order to improve performance when dealing with heavy-tailed distributions. The method introduces an auxiliary variable; this is the index of a particle at the previous time instant. We allow for a particle in the i th stream at time n to be drawn using a particle from a different stream at time $n - 1$. Let the i th particle at time n be $x_n^{(i)}$ and the index of its “parent”

**FIGURE 17.8**

Plot of $N = 200$ generated particles with the corresponding (normalized) weights, for Example 17.2 using resampling, at time instants (a) $n = 3$, (b) $n = 30$. Compared to Figure 17.5c, d, more particles with significant weights survive.

particle at time $n - 1$ be i_{n-1} . The idea is to sample for the pair $(\mathbf{x}_n^{(i)}, i_{n-1})$, $i = 1, 2, \dots, N$. Employing Bayes rule, we obtain

$$\begin{aligned} p(\mathbf{x}_n, i|\mathbf{y}_{1:n}) &\propto p(\mathbf{y}_n|\mathbf{x}_n)p(\mathbf{x}_n, i|\mathbf{y}_{1:n-1}) \\ &= p(\mathbf{y}_n|\mathbf{x}_n)p(\mathbf{x}_n|i, \mathbf{y}_{1:n-1})P(i|\mathbf{y}_{1:n-1}), \end{aligned} \quad (17.41)$$

where the conditional independencies underlying the state-space model have been used. To unclutter notation, we have used \mathbf{x}_n in place of $\mathbf{x}_n^{(i)}$, and the subscript $n - 1$ has been dropped from i_{n-1} and we use i instead. Note that by the definition of the index i_{n-1} , we have

$$p(\mathbf{x}_n|i, \mathbf{y}_{1:n-1}) = p\left(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}, \mathbf{y}_{1:n-1}\right) = p\left(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}\right), \quad (17.42)$$

and also

$$P(i|\mathbf{y}_{1:n-1}) = W_{n-1}^{(i)}. \quad (17.43)$$

Thus, we can write

$$p(\mathbf{x}_n, i|\mathbf{y}_{1:n}) \propto p(\mathbf{y}_n|\mathbf{x}_n)p\left(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}\right)W_{n-1}^{(i)}. \quad (17.44)$$

The proposal distribution is chosen as

$$q(\mathbf{x}_n, i|\mathbf{y}_{1:n}) \propto p\left(\mathbf{y}_n|\boldsymbol{\mu}_n^{(i)}\right)p\left(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}\right)W_{n-1}^{(i)}. \quad (17.45)$$

Note that we have used $\boldsymbol{\mu}_n^{(i)}$ in place of \mathbf{x}_n in $p(\mathbf{y}_n|\mathbf{x}_n)$, because \mathbf{x}_n is still to be drawn. The estimate $\boldsymbol{\mu}_n^{(i)}$ is chosen in order to be easily computed and at the same time to be a good representative of \mathbf{x}_n . Typically, $\boldsymbol{\mu}_n^{(i)}$ can be the mean, the mode, a draw, or another value associated with the distribution $p\left(\mathbf{x}_n|\mathbf{x}_{n-1}^{(i)}\right)$.

For example, $\mu_n^{(i)} \sim p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)})$. Also, if the state equation is $\mathbf{x}_n = f(\mathbf{x}_{n-1}) + \eta_n$, a good choice would be $\mu_n^{(i)} = f(\mathbf{x}_{n-1}^{(i)})$.

Applying the Bayes rule in Eq. (17.45) and adopting

$$q(\mathbf{x}_n | i, \mathbf{y}_{1:n}) = p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}),$$

we obtain

$$q(i | \mathbf{y}_{1:n}) \propto p(\mathbf{y}_n | \mu_n^{(i)}) W_{n-1}^{(i)}. \quad (17.46)$$

Hence, we draw the value of the index i_{n-1} from a multinomial distribution, that is,

$$i_{n-1} \sim q(i | \mathbf{y}_{1:n}) \propto p(\mathbf{y}_n | \mu_n^{(i)}) W_{n-1}^{(i)}, \quad i = 1, 2, \dots, N. \quad (17.47)$$

The index i_{n-1} identifies the distribution from which $\mathbf{x}_n^{(i)}$ will be drawn, that is,

$$\mathbf{x}_n^{(i)} \sim p(\mathbf{x}_n | \mathbf{x}_{n-1}^{(i)}), \quad i = 1, 2, \dots, N. \quad (17.48)$$

Note that Eq. (17.47) actually performs a resampling. However, now, the resampling at time $n - 1$ takes into consideration information that becomes available at time n , via the observation \mathbf{y}_n . This information is exploited in order to determine which particles are to survive, after resampling at a given time instant, so that their “offsprings” are likely to land in regions of high-probability mass. Once sample $\mathbf{x}_n^{(i)}$ has been drawn, the index i_{n-1} is discarded, which is equivalent to marginalizing $p(\mathbf{x}_n, i | \mathbf{y}_{1:n})$ to obtain $p(\mathbf{x}_n | \mathbf{y}_{1:n})$. Each sample $\mathbf{x}_n^{(i)}$ is finally assigned a weight according to

$$w_n^{(i)} \propto \frac{p(\mathbf{x}_n^{(i)}, i_{n-1} | \mathbf{y}_{1:n})}{q(\mathbf{x}_n^{(i)}, i_{n-1} | \mathbf{y}_{1:n})} = \frac{p(\mathbf{y}_n | \mathbf{x}_n^{(i)})}{p(\mathbf{y}_n | \mu_n^{(i)})},$$

which results by dividing the right-hand sides of Eqs. (17.44) and (17.45). Note that the weight accounts for the mismatch between the likelihood $p(\mathbf{y}_n | \cdot)$ at the actual sample and at the predicted point, $\mu_n^{(i)}$. The resulting algorithm is summarized next.

Algorithm 17.6 (Auxiliary particle filtering).

- Initialization: Select a prior distribution, p , to generate the initial state \mathbf{x}_0 .
- Select N .
- **For** $i = 1, 2, \dots, N$, **Do**
 - Draw $\mathbf{x}_0^{(i)} \sim p(\mathbf{x})$; Initialize N streams of particles.
 - Set $W_0^{(i)} = \frac{1}{N}$; Set all normalized weights to equal values.
- **End For**
- **For** $n = 1, 2, \dots$, **Do**
 - **For** $i = 1, 2, \dots, N$, **Do**
 - Draw/compute $\mu_n^{(i)}$
 - $Q_i = p(\mathbf{y}_n | \mu_n^{(i)}) W_{n-1}^{(i)}$; This corresponds to $q(i | \mathbf{y}_{1:n})$ in Eq. (17.46).
 - **End For**

- **For** $i = 1, 2, \dots, N$, **Do**
 - Compute normalized Q_i
- **End For**
- **For** $i = 1, 2, \dots, N$, **Do**
 - $i_{n-1} \sim Q_i$; Eq. (17.47).
 - Draw $x_n^{(i)} \sim p(x|x_{n-1}^{(i)})$
 - Compute $w_n^{(i)} = \frac{p(y_n|x_n^{(i)})}{p(y_n|\mu_n^{(i)})}$
- **End For**
- **For** $i = 1, 2, \dots, N$, **Do**
 - Compute normalized $W_n^{(i)}$
- **End For**
- **End For**

Figure 17.9 shows $N = 200$ particles and their respective normalized weights, generated by Algorithm 17.6 for the observation sequence of Example 17.1 and using the same proposal distribution. Observe that compared to the corresponding Figures 17.5 and 17.8, a substantially larger number of particles with significant weights survive.

The previous algorithm is sometimes called the *single-stage* auxiliary particle filter as opposed to the *two-stage* one, which was originally proposed in [34]. The latter involved an extra resampling step to obtain samples with equal weights. It has been experimentally verified that the single-stage version leads to enhanced performance, and it is the one that is widely used. It has been reported that the auxiliary particle filter may lead to enhanced performance compared to Algorithm 17.5, for high signal-to-noise ratios. However, for high-noise terrains its performance degrades (see, e.g., [1]). More results concerning the performance and analysis of the auxiliary filter can be found in, for example, [13, 23, 35].

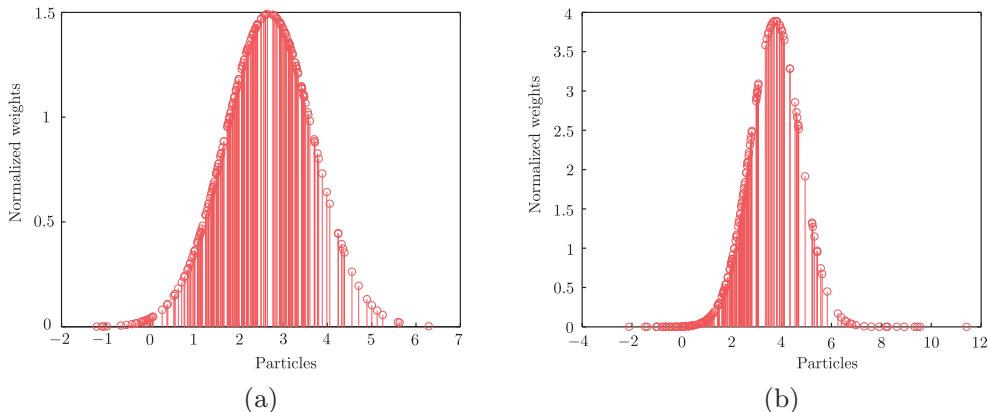


FIGURE 17.9

Plot of $N = 200$ generated particles with the corresponding (normalized) weights, for the same observation sequence as that in Example 17.1, using the auxiliary particle-filtering algorithm, at time instants (a) $n = 3$ and (b) $n = 30$. Compared to Figures 17.5c, d, and Figure 17.8, more particles with significant weights survive.

Remarks 17.3.

- Besides the algorithms presented earlier, a number of variants have been proposed over the years in order to overcome the main limitations of particle filters, associated with the increasing variance and the sample impoverishment problem. In *resample - move* [17] and *block sampling* [14], instead of just sampling for $\mathbf{x}_n^{(i)}$ at time instant, n , one also tries to modify past values, over a window $[n - 1, n - L + 1]$ of fixed size L , in light of the newly arrived observation \mathbf{y}_n . In the *regularized particle filter* [32], in the resampling stage of [Algorithm 17.5](#), instead of sampling from a discrete distribution, samples are drawn from a smooth approximation,

$$p(\mathbf{x}_n | \mathbf{y}_{1:n}) \simeq \sum_{i=1}^N W_n^{(i)} K(\mathbf{x}_n - \mathbf{x}_n^{(i)}),$$

where $K(\cdot)$ is a smooth kernel density function. In [26, 27], the posteriors are approximated by Gaussians; as opposed to the more classical extended Kalman filters, the updating and filtering is accomplished via the propagation of particles.

The interested reader may find more information concerning particle filtering in the tutorial papers [1, 9, 15].

- Rao - Blackwellization* is a technique used to reduce the variance of estimates that are obtained via Monte Carlo sampling methods (e.g., [4]). To this end, this technique has also been employed in particle filtering of dynamic systems. It turns out that, often in practice, some of the states are conditionally linear given the nonlinear ones. The main idea consists of treating the linear states differently by viewing them as nuisance parameters and *marginalizing* them out of the estimation process. The particles of the nonlinear states are propagated randomly, and then the task is treated linearly via the use of a Kalman filter (see, e.g., [10, 12, 24]).
- Smoothing* is closely related to filtering processing. In filtering, the goal lies in obtaining estimates of $\mathbf{x}_{1:n}$ (\mathbf{x}_n) based on observations taken in the interval $[1, n]$, that is, on $\mathbf{y}_{1:n}$. In smoothing, one obtains estimates of \mathbf{x}_n based on an observation set $\mathbf{y}_{1:n+k}$, $k > 0$. There are two paths to smoothing. One is known as *fixed lag* smoothing, where k is a fixed lag. The other is known as *fixed interval*, where one is interested in obtaining estimates based on observations taken over an interval $[1, T]$, that is, based on a fixed set of measurements $\mathbf{y}_{1:T}$.

There are different algorithmic approaches to smoothing. The naive one is to run the particle filtering up to time k or T and use the obtained weights for weighting the particles at time n , in order to form the random measure, that is,

$$p(\mathbf{x}_n | \mathbf{y}_{1:n+k}) \simeq \sum_{i=1}^N W_{n+k}^{(i)} \delta(\mathbf{x}_n - \mathbf{x}_n^{(i)}).$$

This can be a reasonable approximation for small values of k (or $T - n$). Other, more refined, techniques adopt a two-pass rationale. First, a particle filtering is run, and then a backward set of recursions is used to modify the weights (see, e.g., [10]).

- A summary concerning convergence results related to particle filtering can be found in, for example, [6].
- A survey on applications of particle filtering in signal processing-related tasks is given in [8, 9].

- Following the general trend for developing algorithms for distributed learning, a major research effort has been dedicated in this direction in the context of particle filtering. For a review on such schemes, see, for example, [20].
- One of the main difficulties of the particle-filtering methods is that the number of particles required to approximate the underlying distributions increases exponentially with the state dimension. To overcome this problem, several methods have been proposed. In [30], the authors propose to partition the state and estimate each partition independently. In [16], the annealed particle filter is proposed, which implements a coarse-to-fine strategy by using a series of smoothed weighting functions. The unscented particle filter, [37], proposes to use the unscented transform for each particle to avoid wasting resources in low likelihood regions. In [31], a hierarchical search strategy is proposed that uses auxiliary lower dimension models to guide the search in the higher dimensional one.

Example 17.3. *Stochastic Volatility Model.* Consider the following state-space model for generating the observations

$$\begin{aligned} x_n &= \alpha x_{n-1} + \eta_n \\ y_n &= \beta v_n \exp\left(\frac{x_n}{2}\right). \end{aligned}$$

This model belongs to a more general class known as *stochastic volatility* models, where the variance of a process is itself randomly distributed. Such models are used in financial mathematics to model derivative securities, such as options. The state variable is known as the *log-volatility*. We assume the two noise sequences to be i.i.d. and mutually independent Gaussians with zero mean and variances σ_η^2 and σ_v^2 , respectively. The model parameters, α and β , are known as the *persistence in volatility shocks* and *modal volatility*, respectively. The adopted values for the parameters are $\sigma_\eta^2 = 0.178$, $\sigma_v^2 = 1$, $\alpha = 0.97$, and $\beta = 0.69$.

The goal of the example is to generate a sequence of observations and then, based on these measurements, to predict the state, which is assumed to be unknown. To this end, we generate a sequence of $N = 2000$ particles, and the state variable at each time instant is estimated as the weighted average of the generated particles, that is

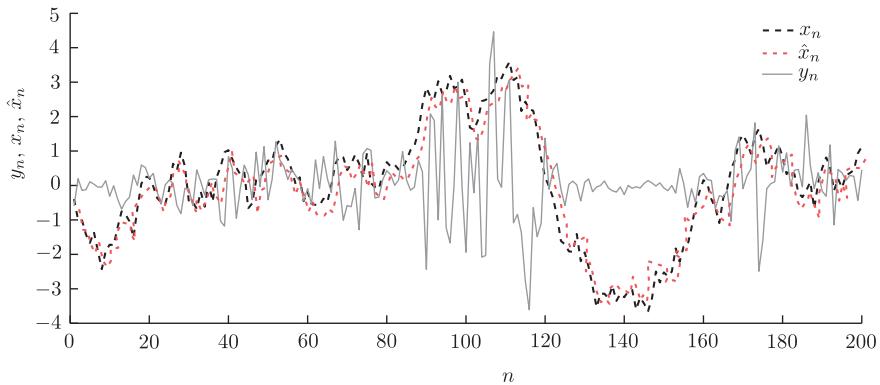
$$\hat{x}_n = \sum_{i=1}^N W_n^{(i)} x_n^{(i)}.$$

Both the SIR Algorithm 17.5 and the auxiliary filter method of Algorithm 17.6 were used. The proposal distribution was

$$q(x_n|x_{n-1}) = \mathcal{N}(x_n|\alpha x_{n-1}, \sigma_\eta^2).$$

Figure 17.10 shows the observation sequence together with the obtained estimate. For comparison reasons, the corresponding true-state value is also shown. Both methods for generating particles gave almost identical results, and we only show one of them. Observe how closely the estimates follow the true values.

Example 17.4. *Visual Tracking.* Consider the problem of visual tracking of a circle, which has a constant and known radius. We seek to track its position, that is, the coordinates of its center, $\mathbf{x} = [x_1, x_2]^T$. This vector will comprise the state variable. The model for generating the observations is given by

**FIGURE 17.10**

The observation sequence generated by the volatility model together with the true and estimated values of the state variable.

$$\begin{aligned} \mathbf{x}_n &= \mathbf{x}_{n-1} + \boldsymbol{\eta}_n, \\ \mathbf{y}_n &= \mathbf{x}_n + \mathbf{v}_n, \end{aligned} \quad (17.49)$$

where $\boldsymbol{\eta}_n$ is a uniform noise in the interval $[-10, 10]$ pixels, for each dimension. Note that, due to the uniform nature of the noise, Kalman filtering, in its standard formulation, is no longer the optimal choice, in spite of the linearity of the model. The noise \mathbf{v}_n follows a Gaussian pdf $\mathcal{N}(\mathbf{0}, \Sigma_v)$, where

$$\Sigma_v = \begin{bmatrix} 2 & 0.5 \\ 0.5 & 2 \end{bmatrix}.$$

Initially, the target circle is located in the image center. The particle filter employs $N=50$ particles and the SIS sampling method was used, (see, also, MATLAB [Exercise 17.12](#))

[Figure 17.11](#) shows the circle and the generated particles, which attempt to track the center of the circle from the noisy observations, for different time instants. Observe how closely the particles track the center of the circle as it moves around. A related video is available from the companion site of this book.

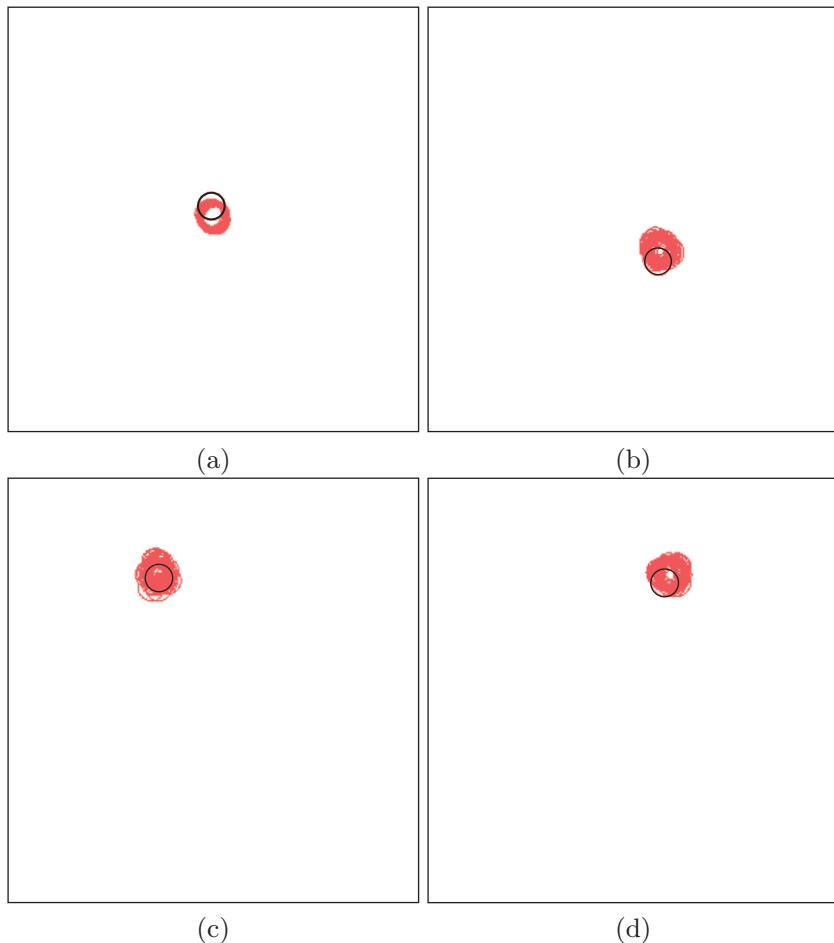
PROBLEMS

17.1 Let

$$\mu := \mathbb{E}[f(\mathbf{x})] = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$$

and $q(\mathbf{x})$ be the proposal distribution. Show that if

$$w(\mathbf{x}) := \frac{p(\mathbf{x})}{q(\mathbf{x})},$$

**FIGURE 17.11**

The circle (in gray) and the generated particles (in red) for time instants $n = 1$, $n = 30$, $n = 60$, and $n = 120$.

and

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^N w(\mathbf{x}_i) f(\mathbf{x}_i),$$

then the variance

$$\sigma_f^2 = \mathbb{E} [(\hat{\mu} - \mathbb{E} [\hat{\mu}])^2] = \frac{1}{N} \left(\int \frac{f^2(\mathbf{x}) p^2(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} - \mu^2 \right).$$

Observe that if $f^2(\mathbf{x}) p^2(\mathbf{x})$ goes to zero slower than $q(\mathbf{x})$, then for fixed N , $\sigma_f^2 \rightarrow \infty$.

17.2 In importance sampling, with weights defined as

$$w(\mathbf{x}) = \frac{\phi(\mathbf{x})}{q(\mathbf{x})},$$

where

$$p(\mathbf{x}) = \frac{1}{Z} \phi(\mathbf{x}),$$

we know from [Problem 14.6](#) that the estimate

$$\hat{Z} = \frac{1}{N} \sum_{i=1}^N w(\mathbf{x}_i)$$

defines an unbiased estimator of the normalizing constant, Z . Show that the respective variance is given by

$$\text{var}[\hat{Z}] = \frac{Z^2}{N} \left(\int \frac{p^2(\mathbf{x})}{q(\mathbf{x})} d\mathbf{x} - 1 \right).$$

17.3 Show that using resampling in importance sampling, then as the number of particles tends to infinity, the approximating, by the respective discrete random measure, distribution, \bar{p} , tends to the true (desired) one, p .

Hint: Consider the one-dimensional case.

17.4 Show that in sequential importance sampling, the proposal distribution that minimizes the variance of the weight at time n , conditioned on $\mathbf{x}_{1:n-1}$, is given by

$$q_n^{opt}(\mathbf{x}_n | \mathbf{x}_{1:n-1}) = p_n(\mathbf{x}_n | \mathbf{x}_{1:n-1}).$$

17.5 In a sequential importance sampling task, let

$$p_n(\mathbf{x}_{1:n}) = \prod_{k=1}^n \mathcal{N}(x_k | 0, 1)$$

$$\phi_n(\mathbf{x}_{1:n}) = \prod_{k=1}^n \exp\left(-\frac{x_k^2}{2}\right),$$

and let the proposal distribution be

$$q_n(\mathbf{x}_{1:n}) = \prod_{k=1}^n \mathcal{N}(x_k | 0, \sigma^2).$$

Let the estimate of $Z_n = (2\pi)^{\frac{n}{2}}$, be

$$\hat{Z}_n = \frac{1}{N} \sum_{i=1}^N w(\mathbf{x}_{1:n}^{(i)}).$$

Show that the variance of the estimator is given by

$$\text{var}[\hat{Z}_n] = \frac{Z_n^2}{N} \left(\left(\frac{\sigma^4}{2\sigma^2 - 1} \right)^{\frac{n}{2}} - 1 \right).$$

Observe that for $\sigma^2 > 1/2$, which is the range of values for the above formula makes sense and guarantees a finite value for the variance, the variance exhibits an exponential increase with respect to n . To keep the variance small, one has to make N very large, that is, to generate a very large number of particles [15].

- 17.6** Prove that the use of the optimal proposal distribution in particle filtering leads to

$$w_n(\mathbf{x}_{1:n}) = w_{n-1}(\mathbf{x}_{1:n-1})p(\mathbf{y}_n|\mathbf{x}_{n-1}).$$

MATLAB Exercises

- 17.7** For the state-space model of [Example 17.1](#), implement the generic particle filtering algorithm for different numbers of particle streams N and different thresholds of effective particle sizes N_{eff} .
Hint. Start by selecting a distribution (the normal should be a good start) and initialize. Then, update the particles in each step according to the algorithm. Finally, check whether the N_{eff} is lower than the threshold and if it is, continue with the resampling process.
- 17.8** For the same example as before, implement the SIS particle filtering algorithm and plot the resulting particles together with the normalized weights for various time instances n . Observe the degeneracy phenomenon of the weights as time evolves.
- 17.9** For [Example 17.1](#), implement the SIR particle filtering algorithm for different numbers of particle streams N and for various time instances n . Use $N_T = N/2$. Compare the performance of SIR and SIS algorithms.
- 17.10** Repeat the previous exercise, implement the auxiliary particle filtering (APF) algorithm and compare the particle-weight histogram with the ones obtained from SIS and SIR algorithms. Observe that the number of particles with significant weights that survive is substantially larger.
- 17.11** Reproduce [Figure 17.10](#) for the stochastic volatility model of [Example 17.3](#) and observe how the estimated sequence \hat{x}_n follows the true sequence x_n based on the observations y_n .
- 17.12** Develop the MATLAB code to reproduce the visual tracking of the circle of [Example 17.4](#). Because, at each time instant, we are only interested in the \mathbf{x}_n and not on the whole sequence, modify the SIS sampling in [Algorithm 17.4](#) to care for this case.

Specifically, given Eqs. (17.28)–(17.29), then in order to estimate \mathbf{x}_n instead of $\mathbf{x}_{1:n}$, Eq. (17.31) is simplified to

$$p(\mathbf{x}_n|\mathbf{y}_{1:n}) = \frac{p(\mathbf{y}_n|\mathbf{x}_n)p(\mathbf{x}_n|\mathbf{y}_{1:n-1})}{p(\mathbf{y}_n|\mathbf{y}_{1:n-1})}, \quad (17.50)$$

where

$$p(\mathbf{x}_n|\mathbf{y}_{1:n-1}) = \int_{\mathbf{x}_{n-1}} p(\mathbf{x}_n|\mathbf{x}_{n-1})p(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}) d\mathbf{x}_{n-1}. \quad (17.51)$$

The samples are now weighted as

$$w_n^{(i)} = \frac{p(\mathbf{x}_n^{(i)}|\mathbf{y}_{1:n})}{q(\mathbf{x}_n^{(i)}|\mathbf{y}_{1:n})}, \quad (17.52)$$

and a popular selection for the proposal distribution is

$$q(\mathbf{x}_n|\mathbf{y}_{1:n}) \equiv p(\mathbf{x}_n|\mathbf{y}_{1:n-1}). \quad (17.53)$$

Substituting Eqs. (17.50) and (17.53) into Eq. (17.52), we get the following rule for the weights:

$$w_n^{(i)} \propto p(\mathbf{y}_n | \mathbf{x}_n^{(i)}). \quad (17.54)$$

REFERENCES

- [1] M.S. Arulampalam, S. Maskell, N. Gordon, T. Clapp, A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking, *IEEE Trans. Signal Process.* 50 (2) (2002) 174-188.
- [2] C.M. Bishop, *Pattern Recognition and Machine Learning*, Springer, New York, 2006.
- [3] J. Carpenter, P. Clifford, P. Fearnhead, Improved particle filter for nonlinear problems, in: *Proceedings IEE, Radar, Sonar and Navigation*, vol. 146, 1999, pp. 2-7.
- [4] G. Casella, C.P. Robert, Rao-Blackwellisation of sampling schemes, *Biometrika* 83 (1) (1996) 81-94.
- [5] N. Chopin, Central limit theorem for sequential Monte Carlo methods and its application to Bayesian inference, *Ann. Stat.* 32 (2004).
- [6] D. Crisan, A. Doucet, A survey of convergence results on particle filtering methods for practitioners, *IEEE Trans. Signal Process.* 50 (3) (2002) 736-746.
- [7] P. Del Moral, *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*, Springer-Verlag, New York, 2004.
- [8] P.M. Djuric, Y. Huang, T. Ghirmai, Perfect sampling: a review and applications to signal processing, *IEEE Trans. Signal Process.* 50 (2002) 345-356.
- [9] P.M. Djuric, J.H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M.F. Bugallo, J. Miguez, Particle filtering, *IEEE Signal Process. Mag.* 20 (2003) 19-38.
- [10] P.M. Djuric, M. Bugallo, Particle filtering, in: T. Adali, S. Haykin (Eds.), *Adaptive Signal Processing: Next Generation Solutions*, John Wiley & Sons, Inc., New York, 2010.
- [11] A. Doucet, S. Godsill, C. Andrieu, On sequential Monte Carlo sampling methods for Bayesian filtering, *Stat Comput* 10 (2000) 197-208.
- [12] R. Douc, O. Cappe, E. Moulines, Comparison of resampling schemes for particle filtering, in: *4th International Symposium on Image and Signal Processing and Analysis (ISPA)*, 2005.
- [13] R. Douc, E. Moulines, J. Olsson, On the auxiliary particle filter, 2010, arXiv:0709.3448v1 [math.ST].
- [14] A. Doucet, M. Briers, S. Sénelac, Efficient block sampling strategies for sequential Monte Carlo methods, *J. Comput. Graph. Stat.* 15 (2006) 693-711.
- [15] A. Doucet, A.M. Johansen, A tutorial on particle filtering and smoothing: Fifteen years later, in: *Handbook of Nonlinear Filtering*, Oxford University Press, Oxford, 2011.
- [16] J. Deutscher, A. Blake, I. Reid, Articulated body motion capture by annealed particle filtering, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2000, pp. 126-133.
- [17] W.R. Gilks, C. Berzuini, Following a moving target—Monte Carlo inference for dynamic Bayesian models, *J. R. Stat. Soc. B* 63 (2001) 127-146.
- [18] N.J. Gordon, D.J. Salmond, A.F.M. Smith, Novel approach to nonlinear/non-Gaussian Bayesian state estimation, *Proc IEEE F* 140 (2) (1993) 107-113.
- [19] J.M. Hammersley, K.W. Morton, Poor man's Monte Carlo, *J. R. Stat. Soc. B* 16 (1) (1954) 23-38.
- [20] O. Hinka, F. Hlawatz, P.M. Djuric, Distributed particle filtering in agent networks, *IEEE Signal Process. Mag.* 30 (1) (2013) 61-81.
- [21] S. Hong, S.S. Chin, P.M. Djurić, M. Bolić, Design and implementation of flexible resampling mechanism for high-speed parallel particle filters, *J. VLSI Signal Process.* 44 (1-2) (2006) 47-62.
- [22] Y. Huang, P.M. Djurić, A blind particle filtering detector of signals transmitted over flat fading channels, *IEEE Trans. Signal Process.* 52 (7) (2004) 1891-1900.

- [23] A.M. Johansen, A. Doucet, A note on auxiliary particle filters, *Stat. Probab. Lett.* 78 (12) (2008) 1498-1504.
- [24] R. Karlsson, F. Gustafsson, Complexity analysis of the marginalized particle filter, *IEEE Trans. Signal Process.* 53 (11) (2005) 4408-4411.
- [25] G. Kitagawa, Monte Carlo filter and smoother for non-Gaussian nonlinear state space models, *J. Comput. Graph. Stat.* 5 (1996) 1-25.
- [26] J.H. Kotecha, P.M. Djurić, Gaussian particle filtering, *IEEE Trans. Signal Process.* 51 (2003) 2592-2601.
- [27] J.H. Kotecha, P.M. Djurić, Gaussian sum particle filtering, *IEEE Trans. Signal Process.* 51 (2003) 2602-2612.
- [28] J.S. Liu, R. Chen, Sequential Monte Carlo methods for dynamical systems, *J. Am. Stat. Assoc.* 93 (1998) 1032-1044.
- [29] J.S. Liu, *Monte Carlo Strategies in Scientific Computing*, Springer, New York, 2001.
- [30] J. MacCormick, M. Isard, Partitioned sampling, articulated objects, and interface-quality hand tracking, in: *Proceedings of the 6th European Conference on Computer Vision, Part II, ECCV*, Springer-Verlag, London, UK, 2000, pp. 3-19.
- [31] A. Makris, D. Kosmopoulos, S. Perantonis, S. Theodoridis, A hierarchical feature fusion framework for adaptive visual tracking, *Image Vis. Comput.* 29 (9) (2011) 594-606.
- [32] C. Musso, N. Oudjane, F. Le Gland, Improving regularised particle filters, in: A. Doucet, N. de Freitas, N.J. Gordon (Eds.), *Sequential Monte Carlo Methods in Practice*, Springer-Verlag, New York, 2001.
- [33] A. Owen, Y. Zhou, Safe and effective importance sampling, *J. Am. Stat. Assoc.* 95 (2000) 135-143.
- [34] M.K. Pitt, N. Shephard, Filtering via simulation: auxiliary particle filters, *J. Am. Stat. Assoc.* 94 (1999) 590-599.
- [35] M.K. Pitt, R.S. Silva, P. Giordani, R. Kohn, Auxiliary particle filtering within adaptive Metropolis-Hastings sampling, 2010, [http://arxiv.org/abs/1006.1914\[stat.Me\]](http://arxiv.org/abs/1006.1914[stat.Me]).
- [36] M.N. Rosenbluth, A.W. Rosenbluth, Monte Carlo calculation of the average extension of molecular chains, *J. Chem. Phys.* 23 (2) (1956) 356-359.
- [37] R. van der Merwe, N. de Freitas, A. Doucet, E. Wan, The unscented particle filter, in: *Proceedings Advances in Neural Information Processing Systems (NIPS)*, 2000.

This page intentionally left blank

NEURAL NETWORKS AND DEEP LEARNING

18

CHAPTER OUTLINE

18.1	Introduction	886
18.2	The Perceptron	887
18.2.1	The Kernel Perceptron Algorithm	891
18.3	Feed-Forward Multilayer Neural Networks	892
18.4	The Backpropagation Algorithm	896
18.4.1	The Gradient Descent Scheme	897
	<i>Speeding up the Convergence Rate</i>	903
	<i>Some Practical Hints</i>	904
18.4.2	Beyond the Gradient Descent Rationale	905
18.4.3	Selecting a Cost Function	906
18.5	Pruning the Network	907
18.6	Universal Approximation Property of Feed-Forward Neural Networks	909
18.7	Neural Networks: A Bayesian Flavor	912
18.8	Learning Deep Networks	913
18.8.1	The Need for Deep Architectures	914
18.8.2	Training Deep Networks	915
	<i>Distributed Representations</i>	917
18.8.3	Training Restricted Boltzmann Machines	918
	<i>Computation of the Conditional Probabilities</i>	920
	<i>Contrastive Divergence</i>	921
18.8.4	Training Deep Feed-Forward Networks	924
18.9	Deep Belief Networks	926
18.10	Variations on the Deep Learning Theme	928
18.10.1	Gaussian Units	928
18.10.2	Stacked Autoencoders	929
18.10.3	The Conditional RBM	930
18.11	Case Study: A Deep Network for Optical Character Recognition	933
18.12	CASE Study: A Deep Autoencoder	935
18.13	Example: Generating Data via a DBN	938
Problems		939
	<i>MATLAB Exercises</i>	941
References		942

18.1 INTRODUCTION

Neural networks have a long history that goes back to the first attempts to understand how the human (and more generally, the mammal) brain works and how what we call intelligence is formed.

From a physiological point of view, one can trace the beginning of the field back to the work of Santiago Ramon y Cajal, [68], who discovered that the basic building element of the brain is the *neuron*. The brain comprises approximately 60 to 100 billions neurons; that is, a number of the same order as the number of stars in our galaxy! Each neuron is connected with other neurons via elementary structural and functional units/links, known as *synapses*. It is estimated that there are 50 to 100 trillions of synapses. These links mediate information between connected neurons. The most common type of synapses are the chemical ones, which convert electric pulses, produced by a neuron, to a chemical signal and then back to an electrical one. Depending on the input pulse(s), a synapse is either *activated* or *inhibited*. Via these links, each neuron is connected to other neurons and this happens in a hierarchically structured way, in a layer-wise fashion.

Santiago Ramon y Cajal (1852–1934) was a Spanish pathologist, histologist, neuroscientist, and Nobel laureate. His many pioneering investigations of the microscopic structure of the brain have established him as the father of modern neuroscience.

A milestone from the learning theory's point of view occurred in 1943, when Warren McCulloch and Walter Pitts, [59], developed a computational model for the basic neuron. Moreover, they provided results that tie neurophysiology with mathematical logic. They showed that given a sufficient number of neurons and adjusting appropriately the synaptic links, each one represented by a weight, one can compute, in principle, any computable function. As a matter of fact, it is generally accepted that this is the paper that gave birth to the fields of neural networks and artificial intelligence.

Warren McCulloch (1898–1969) was an American psychiatrist and neuroanatomist who spent many years studying the representation of an event in the neural system. Walter Pitts (1923–1969) was an American logician who worked in the field of cognitive psychology. He was a mathematical prodigy and he taught himself logic and mathematics. At the age of 12, he read *Principia Mathematica* by Alfred North Whitehead and Bertrand Russell and he wrote a letter to Russell commenting on certain parts of the book. He worked with a number of great mathematicians and logicians including Wiener, Householder, and Carnap. When he met McCulloch at the University of Chicago, he was familiar with the work of Leibnitz on computing, which inspired them to study whether the nervous system could be considered to be a type of universal computing device. This gave birth to their 1943 paper, mentioned in the reference before.

Frank Rosenblatt, [73, 74], borrowed the idea of a neuron model, as suggested by McCulloch and Pitts, to build a true learning machine which learns from a set of training data. In the most basic version of operation, he used a single neuron and adopted a rule that can learn to separate data, which belong to two linearly separable classes. That is, he built a pattern recognition system. He called the basic neuron a *perceptron* and developed a rule/algorithm, the *perceptron algorithm*, for the respective training. The perceptron will be the kick-off point for our tour in this chapter.

Frank Rosenblatt (1928–1971) was educated at Cornell, where he obtained his PhD in 1956. In 1959, he took over as director of Cornell's Cognitive Systems Research Program and also as a lecturer in the psychology department. He used an IBM 704 computer to simulate his perceptron and later built a special-purpose hardware, which realized the perceptron learning rule.

Neural networks are learning machines, comprising a large number of neurons, which are connected in a layered fashion. Learning is achieved by adjusting the unknown synaptic weights to minimize a preselected cost function. It took almost 25 years, after the pioneering work of Rosenblatt, for neural networks to find their widespread use in machine learning. This is the time period needed for the basic McCulloch Pitts's model of a neuron to be generalized and lead to an algorithm for training such networks. A breakthrough came under the name *backpropagation algorithm*, which was developed for training neural networks based on a set of input-output training samples. Backpropagation is also treated in detail in this chapter.

It is interesting to note that neural networks dominated the field of machine learning for almost a decade, from 1986 until the middle of 1990s. Then they were superseded, to a large extent, by the support vector machines, which established their reign until very recently. At the time the book is being compiled, there is an aggressive resurgence in the interest on neural networks, in the context of *deep learning*. Interestingly enough, there is one name that is associated with the revival of interest on neural networks, both in the mid-1980s and now; this is the name of Geoffrey Hinton [34, 75]. Deep learning refers to learning networks with many layers of neurons, and this topic is treated at the end of this chapter.

18.2 THE PERCEPTRON

Our starting point is the simple problem of a *linearly separable* two-class (ω_1, ω_2) classification task. In other words, we are given a set of training samples, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, with $y_n \in \{-1, +1\}$, $\mathbf{x}_n \in \mathbb{R}^l$ and it is assumed that there is a hyperplane,

$$\boldsymbol{\theta}_*^T \mathbf{x} = 0$$

such that,

$$\begin{aligned}\boldsymbol{\theta}_*^T \mathbf{x} &> 0, & \text{if } \mathbf{x} \in \omega_1, \\ \boldsymbol{\theta}_*^T \mathbf{x} &< 0, & \text{if } \mathbf{x} \in \omega_2.\end{aligned}$$

In other words, such a hyperplane classifies correctly *all* the points in the training set. For notational simplification, the bias term of the hyperplane has been absorbed in $\boldsymbol{\theta}_*$ after extending the dimensionality of the problem by one, as it has been explained in [Chapter 3](#) and used in various parts of this book.

The goal now becomes that of developing an algorithm that iteratively computes a hyperplane that classifies correctly all the patterns from both classes. To this end, a cost function is adopted.

The Perceptron Cost. Let the available estimate at the current iteration step of the unknown parameters be $\boldsymbol{\theta}$. Then, there are two possibilities. The first one is that all points are classified correctly; this means that a solution has been obtained. The other alternative is that $\boldsymbol{\theta}$ classifies correctly some of the points and the rest are misclassified. Let \mathcal{Y} be the set of all misclassified samples. The perceptron cost is defined as

$$J(\boldsymbol{\theta}) = - \sum_{n: \mathbf{x}_n \in \mathcal{Y}} y_n \boldsymbol{\theta}^T \mathbf{x}_n : \quad \text{Perceptron Cost},$$

(18.1)

where,

$$y_n = \begin{cases} +1, & \text{if } \mathbf{x} \in \omega_1, \\ -1, & \text{if } \mathbf{x} \in \omega_2. \end{cases} \quad (18.2)$$

Observe that the cost function is nonnegative. Indeed, because the sum is over the misclassified points, if $\mathbf{x}_n \in \omega_1$ (ω_2) then $\theta^T \mathbf{x}_n < (>) 0$ rendering the product $-y_n \theta^T \mathbf{x}_n > 0$. The cost function becomes zero, if there are no misclassified points, that is, $\mathcal{Y} = \emptyset$, which corresponds to a solution.

The perceptron cost function is not differentiable at all points. It is a *continuous piece-wise linear* function. Indeed, let us write it in a slightly different way,

$$J(\theta) = \left(- \sum_{n: \mathbf{x}_n \in \mathcal{Y}} y_n \mathbf{x}_n^T \right) \theta,$$

This is a linear function with respect to θ , as long as the number of misclassified points remains the same. However, as one slowly changes the value of θ , which corresponds to a change of the position of the respective hyperplane, there will be a point where the number of misclassified samples in \mathcal{Y} suddenly changes; this is the time, where a sample in the training set changes its relative position with respect to the (moving) hyperplane and as a consequence the set \mathcal{Y} is modified. After this change, $J(\theta)$ will correspond to a new linear function.

The Perceptron Algorithm. It can be shown, for example, [61, 74], that, starting from an arbitrary point, $\theta^{(0)}$, the following iterative update,

$$\theta^{(i)} = \theta^{(i-1)} + \mu_i \sum_{n: \mathbf{x}_n \in \mathcal{Y}} y_n \mathbf{x}_n : \quad \text{The Perceptron Rule,} \quad (18.3)$$

converges after a *finite number of steps*. The parameter μ_i is the user-defined step size, judiciously chosen to guarantee convergence. Note that this is the same algorithm as the one derived in Section 8.10.2, for minimizing the hinge loss function via the notion of subgradient.

Besides the previous scheme, another version of the algorithm considers one sample per iteration in a cyclic fashion, until the algorithm converges. Let us denote by $y_{(i)}$, $\mathbf{x}_{(i)}$, $(i) \in \{1, 2, \dots, N\}$, the training pair that is presented in the algorithm at the i th iteration step.¹ Then, the update iteration becomes

$$\theta^{(i)} = \begin{cases} \theta^{(i-1)} + \mu_i y_{(i)} \mathbf{x}_{(i)}, & \text{if } \mathbf{x}_{(i)} \text{ is misclassified by } \theta^{(i-1)}, \\ \theta^{(i-1)}, & \text{otherwise.} \end{cases} \quad (18.4)$$

In other words, starting from an initial estimate, usually taken to be equal to zero, $\theta^{(0)} = \mathbf{0}$, we test each one of the samples, \mathbf{x}_n , $n = 1, 2, \dots, N$. Every time a sample is misclassified, action is taken for a correction. Otherwise no action is required. Once all samples have been considered, we say that one *epoch* has been completed. If no convergence has been attained, all samples are reconsidered in a

¹ The symbol (i) has been adopted to denote the time index of the samples, instead of i , because we do not know which point will be presented to the algorithm at the i th iteration. Recall that each training point is considered many times, till convergence is achieved.

second epoch and so on. This is known as *pattern-by-pattern* or *online* mode of operation. However, note that, in contrast to how we have used the term “online” in previous chapters, here we mean that the total number of data samples is fixed and the algorithm considers them in a *cyclic fashion*, epoch after epoch.

After a successive *finite* number of epochs, the algorithm is guaranteed to converge. Note that for convergence, the sequence μ_i must be appropriately chosen. This is pretty familiar to us by now. However for the case of the perceptron algorithm, convergence is still guaranteed even if μ_i is a positive constant, $\mu_i = \mu > 0$, usually taken to be equal to one ([Problem 18.1](#)).

The formulation in (18.4) brings the perceptron algorithm under the umbrella of the so-called *reward-punishment* philosophy of learning. If the current estimate succeeds in predicting the class of the respective pattern, no action is taken. Otherwise, the algorithm is punished to perform an update.

[Figure 18.1](#) provides a geometric interpretation of the perceptron rule. Assume that sample x is misclassified by the hyperplane, $\theta^{(i-1)}$. As we know from geometry, $\theta^{(i-1)}$ corresponds to a vector that is perpendicular to the hyperplane that defines; see also [Figure 11.15](#) in [Section 11.10.1](#). Because x lies in the $(-)$ side of the hyperplane and it is misclassified, it belongs to class ω_1 . Hence, assuming $\mu = 1$, the applied correction by the algorithm is

$$\theta^{(i)} = \theta^{(i-1)} + x,$$

and its effect is to turn the hyperplane to the direction toward x to place it in the $(+)$ side of the new hyperplane, which is defined by the updated estimate $\theta^{(i)}$. The perceptron algorithm in its pattern-by-pattern mode of operation is summarized in [Algorithm 18.1](#).

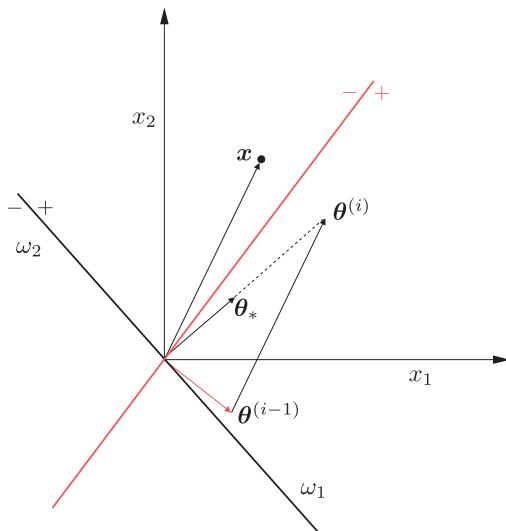


FIGURE 18.1

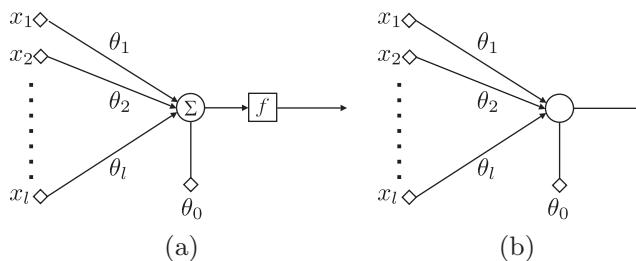
Pattern x is misclassified by the red line. The action of the perceptron rule is to turn the hyperplane toward the point x , in an attempt to include it in the correct side of the new hyperplane and classify it correctly.

Algorithm 18.1 (The online perceptron algorithm).

- Initialization
 - $\theta^{(0)} = \mathbf{0}$.
 - Select μ ; usually it is set equal to one.
 - $i = 0$.
- **Repeat:** Each iteration corresponds to an epoch.
 - counter = 0; Counts the number of updates per epoch.
 - **For** $n = 1, 2, \dots, N$, **Do**; For each epoch, all samples are presented.
 - **If** $(y_n \mathbf{x}_n^T \theta^{(i-1)} \leq 0) **Then**
 - $i = i + 1$
 - $\theta^{(i)} = \theta^{(i-1)} + \mu y_n \mathbf{x}_n$
 - counter=counter+1$
 - **End For**
 - **Until** counter=0

Once the perceptron algorithm has run and converged, we have the weights, θ_i , $i = 1, 2, \dots, l$, of the synapses of the associated neuron/perceptron as well as the bias term θ_0 . These can now be used to classify unknown patterns. Figure 18.2a shows the corresponding architecture of the basic neuron element. The features, x_i , $i = 1, 2, \dots, l$, are applied to the input nodes. In turn, each feature is multiplied by the respective synapse (weight), and then the bias term is added on their linear combination. The outcome of this operation then goes through a nonlinear function, f , known as the *activation function*. Depending on the form of the nonlinearity, different types of neurons occur. In the more classical one, known as the McCulloch-Pitts neuron, the activation function is the Heaviside one, that is,

$$f(z) = \begin{cases} 1 & \text{if } z > 0, \\ 0 & \text{if } z \leq 0. \end{cases} \quad (18.5)$$

**FIGURE 18.2**

- (a) In the basic neuron/perceptron architecture the input features are applied to the input nodes and are weighted by the respective weights of the synapses. The bias term is then added on their linear combination and the result is pushed through the nonlinearity. In the McCulloch-Pitts neuron, the output fires a 1 for patterns in class ω_1 or a zero for the other class. (b) The summation and nonlinear operation are merged together for graphical simplicity.

Usually, the summation operation and the nonlinearity are merged to form a node and the architecture in Figure 18.2b occurs.

Remarks 18.1.

- **ADALINE:** Soon after Rosenblatt proposed the perceptron, Widrow and Hopf proposed the *adaptive line element* (ADALINE), which is a linear version of the perceptron [98]. That is, during training the nonlinearity of the activation function is not involved. The resulting algorithm is the LMS algorithm, treated in detail in Chapter 5. It is interesting to note that the LMS was readily adopted and widely used for online learning within the signal processing and communications communities.

18.2.1 THE KERNEL PERCEPTRON ALGORITHM

In Chapter 11² we discussed the kernelization of various linear algorithms to exploit Cover's theorem and solve a linear task in a reproducing kernel Hilbert space (RKHS), although the original problem is not (linearly) separable in the original space. The perceptron algorithm is not an exception and a kernelized version can be developed. Our starting point is the pattern-by-pattern version of the perceptron algorithm. Every time a pattern, $\mathbf{x}_{(i)}$, is misclassified, a correction to the parameter vector is performed as in (18.4). The difference now is that the place of $\mathbf{x}_{(i)}$ is taken by the image $\phi(\mathbf{x}_{(i)})$, where ϕ denotes the (implicit) mapping into the respective RKHS, as explained in Chapter 11. Let us introduce a new variable, a_n , which counts how many times the corresponding feature vector, \mathbf{x}_n , has participated in the correction update. Then, after convergence (we assume that after the mapping in the RKH space, the classes have become linearly separable and the algorithm is guaranteed to converge) the parameter,³ $\boldsymbol{\theta}$, as well as the bias term, can be written as

$$\boldsymbol{\theta} = \sum_{n=1}^N a_n y_n \phi(\mathbf{x}_n), \quad (18.6)$$

$$\theta_0 = \sum_{n=1}^N a_n y_n, \quad (18.7)$$

where we have considered the bias term separately and the dimension of the input vectors has not been extended. Then, given an unknown pattern, \mathbf{x} , classification is performed according to the sign of

$$f(\mathbf{x}) := \langle \boldsymbol{\theta}, \phi(\mathbf{x}) \rangle + \theta_0 = \sum_{n=1}^N a_n y_n \kappa(\mathbf{x}, \mathbf{x}_n) + \sum_{n=1}^N a_n y_n, \quad (18.8)$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in the RKHS, $\kappa(\cdot, \cdot)$ is the kernel associated with the RKHS, and we have used the kernel trick. The kernelized version of the perceptron algorithm is summarized in Algorithm 18.2.

² The readers who have not read Chapter 11, can bypass this section.

³ Note that $\boldsymbol{\theta}$ may now be function, but we keep the same symbol as in the perceptron algorithm.

Algorithm 18.2 (The kernel perceptron algorithm).

- **For** $n = 1, 2, \dots, N$, **Do**
 - $a_n = 0$
- **End For**
- **Repeat**
 - counter=0
 - **For** $i = 1, 2, \dots, N$, **Do**
 - **If** $(y_n(\sum_{n=1}^N a_n y_n \kappa(x_i, x_n) + \sum_{n=1}^N a_n y_n) \leq 0) **Then**
 - $a_i = a_i + 1$
 - counter=counter+1$
 - **End For**
- **Until** counter=0

18.3 FEED-FORWARD MULTILAYER NEURAL NETWORKS

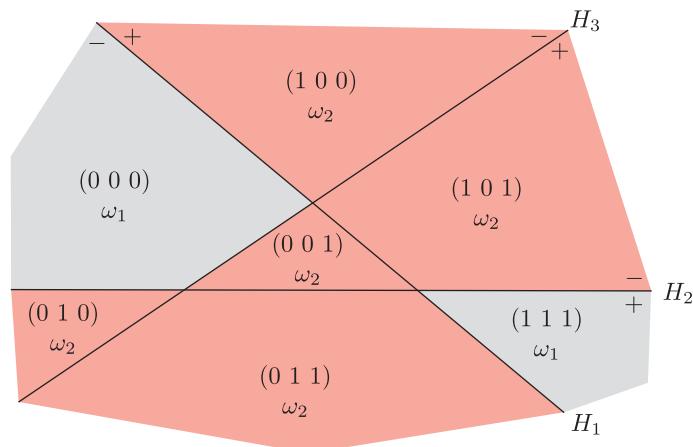
A single neuron is associated with a hyperplane

$$H : \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_l x_l + \theta_0 = 0,$$

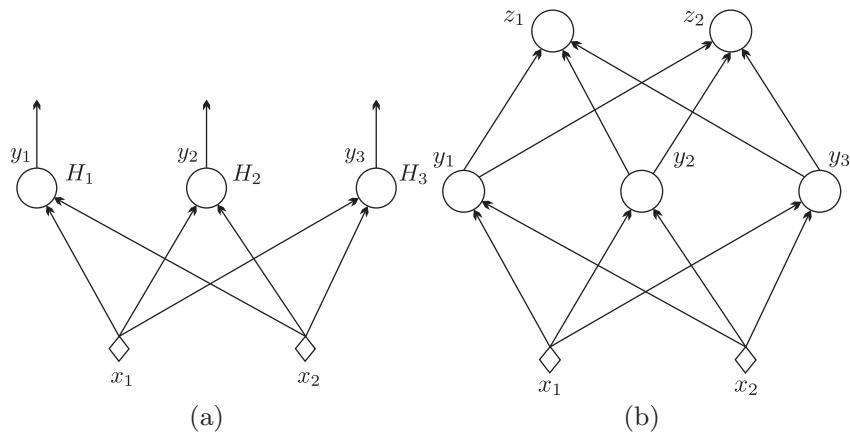
in the input (feature) space. Moreover, classification is performed via the nonlinearity, which fires an one or stays at zero, depending on which side of H a point lies. We will now show how to combine neurons, in a layer-wise fashion, to construct nonlinear classifiers. We will follow a simple constructive proof, which will unveil certain aspects of neural networks. These will be useful later on, when dealing with deep architectures.

As a starting point, we consider the case where the classes in the feature space are formed by unions of polyhedral regions. This is shown in [Figure 18.3](#), for the case of the two-dimensional feature space. Polyhedral regions are formed as intersections of half-spaces, each one associated with a hyperplane. In [Figure 18.3](#), there are three hyperplanes (straight lines in \mathbb{R}^2), indicated as H_1, H_2, H_3 , giving rise to seven polyhedral regions. For each hyperplane, the (+) and (-) sides (half-spaces) are indicated. In the sequel, each one of the regions is labeled using a triplet of binary numbers, depending on which side it is located with respect to H_1, H_2, H_3 . For example, the region labeled as (101), lies in the (+) side of H_1 , the (-) side of H_2 , and the (+) side of H_3 . [Figure 18.4a](#) shows three neurons, realizing the three hyperplanes, H_1, H_2, H_3 , of [Figure 18.3](#), respectively. The associated outputs, denoted as y_1, y_2, y_3 , form the label of the region in which the corresponding input pattern lies. Indeed, if the weights of the synapses have been appropriately set, then if a pattern originates from the region, say, (010), then the first neuron on the left will fire a zero ($y_1 = 0$), the second an one ($y_2 = 1$) and the right-most a zero ($y_3 = 0$). In other words, combining the outputs of the three neurons together, we have achieved a *mapping* of the input feature space into the three-dimensional space. More specifically, the mapping is performed on the vertices of the *unit cube* in \mathbb{R}^3 , as shown in [Figure 18.5](#). In the more general case, where p neurons are employed, the mapping will be on the vertices of the unit hypercube in \mathbb{R}^p . This layer of neurons comprises the first *hidden layer* of the network, which we are developing.

An alternative way to view this mapping is as a new representation of the input patterns in terms of code words. For three neurons/hyperplanes we can form 2^3 binary code-words, each corresponding to

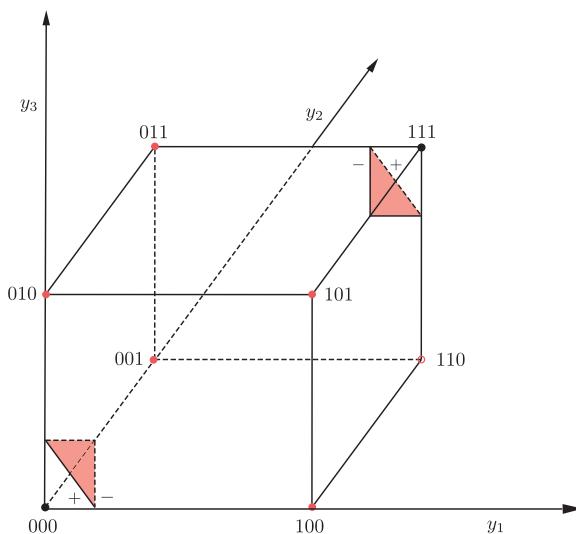
**FIGURE 18.3**

Classes are formed by the union of polyhedral regions. Regions are labeled according to the side on which they lie, with respect to the three lines, H_1 , H_2 , H_3 . The number 1 indicates the (+) side and the 0 the (-) side. Class ω_1 consists of the union of the (000) and (111) regions.

**FIGURE 18.4**

(a) The neurons of the first hidden layer are excited by the feature values applied at the input nodes and form the polyhedral regions. (b) The neurons of the second layer have as inputs the outputs of the first layer, and they thereby form the classes. To simplify the figure, the bias terms for each neuron are not shown.

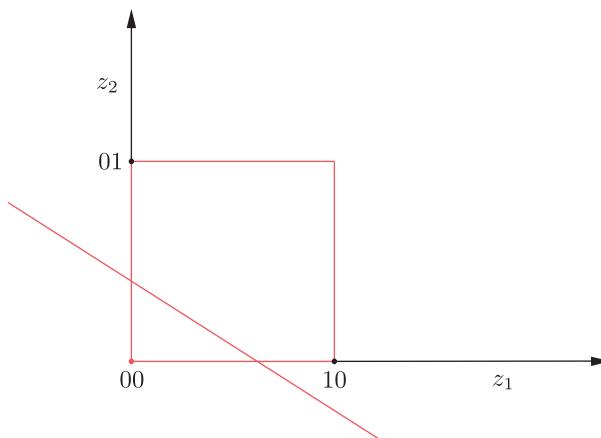
a vertex of the unit cube, which can represent $2^3 - 1 = 7$ regions (there is one remaining vertex i.e., (110), which does not correspond to any region). Note, however, that this mapping encodes information concerning some *structure* of the input data; that is, information relating to how the input patterns are grouped together in the feature space in different regions.

**FIGURE 18.5**

The neurons of the first hidden layer perform a mapping from the input feature space to the vertices of a unit hypercube. Each region is mapped to a vertex. Each vertex of the hypercube is now linearly separable from all the rest and can be separated by a hyperplane realized by a neuron. The vertex 110, denoted as an unshaded circle does not correspond to any region.

We will now use this new representation, as it is provided by the outputs of the neurons of the first hidden layer, as input which feeds the neurons of a second hidden layer, which is constructed as follows. We choose all regions that belong to one class. For the sake of our example in Figure 18.3, we select the two regions that correspond to class ω_1 , that is, (000) and (111). Recall that all the points from these regions are mapped to the respective vertices of the unit cube in the \mathbb{R}^3 . However, in this new transformed space, each one of the vertices is now *linearly separable* from the rest. This means that we can use a neuron/perceptron in the transformed space, which will place a single vertex in the (+) side and the rest in the (-) one, of the associated hyperplane. This is shown in Figure 18.5, where two planes are shown, which separate the respective vertices from the rest. Each of these planes is realized by a neuron, operating in \mathbb{R}^3 , as shown in Figure 18.4b, where a second layer of *hidden* neurons has been added.

Note that the output z_1 of the left neuron will fire a 1 only if the input pattern originates from the region 000 and it will be at 0 for all other patterns. For the neuron on the right, the output z_2 will be 1 for all the patterns coming from region (111) and zero for all the rest. Note that this second layer of neurons has performed a second mapping, this time to the vertices of the unit rectangle in the \mathbb{R}^2 . This mapping provides a new representation of the input patterns, and this representation encodes information related to the classes of the regions. Figure 18.6 shows the mapping to the vertices of the unit rectangle in the (z_1, z_2) space. Note that all the points originating from class ω_2 are mapped to (00) and the points from class ω_1 are mapped either to (10) or to (01). This is very interesting; by successive mappings, we have transformed our originally nonlinearly separable task, to one that is linearly separable. Indeed, the point (00) can be linearly separated from (01) and (10) and this can be

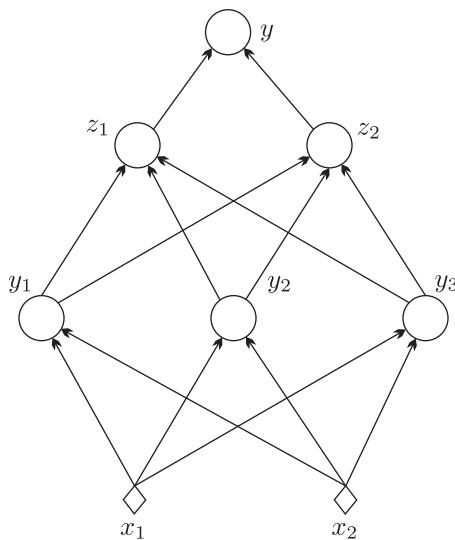
**FIGURE 18.6**

Patterns form class ω_1 are mapped either to (01) or to (10) and patterns from class ω_2 are mapped to (00). Thus the classes have now become linearly separable and can be separated via a straight line realized by a neuron.

realized by an extra neuron operating in the (z_1, z_2) space; it is known as the *output neuron*, because it provides the final classification decision. The final resulting network is shown in Figure 18.7. We call this network *feed-forward*, because information flows forward from the input to the output layer. It comprises the input layer, which is a nonprocessing one, two hidden layers (the term “hidden” is self-explained) and one output layer. We call such a NN an three-layer network, without counting the input layer of nonprocessing nodes.

We have constructively shown that a three-layer feed-forward NN can, in principle, solve *any* classification task whose classes are formed by the union of polyhedral regions. Although we focused on the two-class case, the generalization to multiclass cases is straightforward, by employing more output neurons depending on the number of classes. Note that in some cases, one hidden layer of nodes may be sufficient. This depends on whether the vertices on which the regions are mapped, are assigned to classes so linear separability is possible. For example, this would be the case if class ω_1 was the union of (000) and (100) regions. Then, these two corners could be separated from the rest via a single plane and a second hidden layer of neurons would not be required (check why). In any case, we will not take our discussion any further. The reason is that such a construction is important to demonstrate the power of building a multilayer NN, in analogy to what is happening in our brain. However, from a practical point of view, such a construction has not much to offer. In practice, when the data live in high-dimensional spaces, there is no chance of determining the parameters that define the neurons analytically to realize the hyperplanes, which form the polyhedral regions. Furthermore, in real life, classes are not necessarily formed by the union of polyhedral regions and more important classes do overlap. Hence, one needs to devise a training procedure based on a cost function.

All we will keep from our previous discussion is the structure of the multilayer network, and we will seek ways for estimating the unknown weights of the synapses and biases of the neurons. Moreover, from a conceptual point of view, we have to remember that each layer performs a mapping into a new

**FIGURE 18.7**

A three layer feed-forward neural network. It comprises the input (non-processing) layer, two hidden layers and one output layer of neurons. Such a three layer NN can solve *any* classification task, where classes are formed by unions of polyhedral regions.

space, and each mapping provides a different, hopefully more informative, representation of the input data, until the last layer where the task has been transformed into one that it is easy to solve.

18.4 THE BACKPROPAGATION ALGORITHM

A feed-forward neural network (NN) consists of a number of layers of neurons, and each neuron is determined by the corresponding set of synaptic weights and its bias term. From this point of view, an NN realizes a nonlinear parametric function, $\hat{y} = f_{\theta}(x)$ where θ stands for all the weights/biases present in the network. Thus, training an NN seems not to be any different from training any other parametric prediction model. All that is needed is (a) a set of training samples, (b) a loss function, $\mathcal{L}(y, \hat{y})$, and (c) an iterative scheme, for example, gradient descent, to perform the optimization of the associated empirical loss,

$$J(\theta) = \sum_{n=1}^N \mathcal{L}(y_n, f_{\theta}(x_n)).$$

The difficulty with training NNs lies in their multilayer structure that complicates the computation of the involved gradients, which are needed for the optimization. Moreover, the McCulloch-Pitts neuron involves the discontinuous Heaviside activation function, which is not differentiable. A first step in developing a practical algorithm for training an NN is to replace the Heaviside activation function with a differentiable approximation of it.

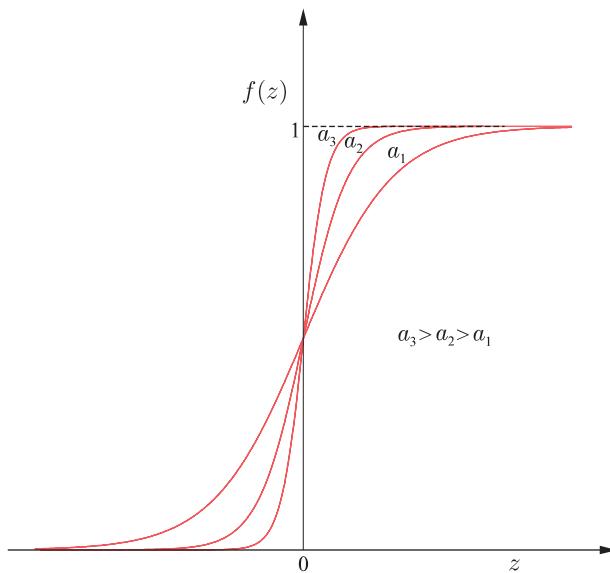


FIGURE 18.8

The logistic sigmoid function for different values of the parameter a .

The logistic sigmoid neuron: One possibility is to adopt the logistic sigmoid function, that is,

$$f(z) = \sigma(z) := \frac{1}{1 + \exp(-az)}. \quad (18.9)$$

The graph of the function is shown in Figure 18.8. Note that the larger the value of the parameter a , the closer the corresponding graph becomes to that of the Heaviside function. Another possibility would be to use

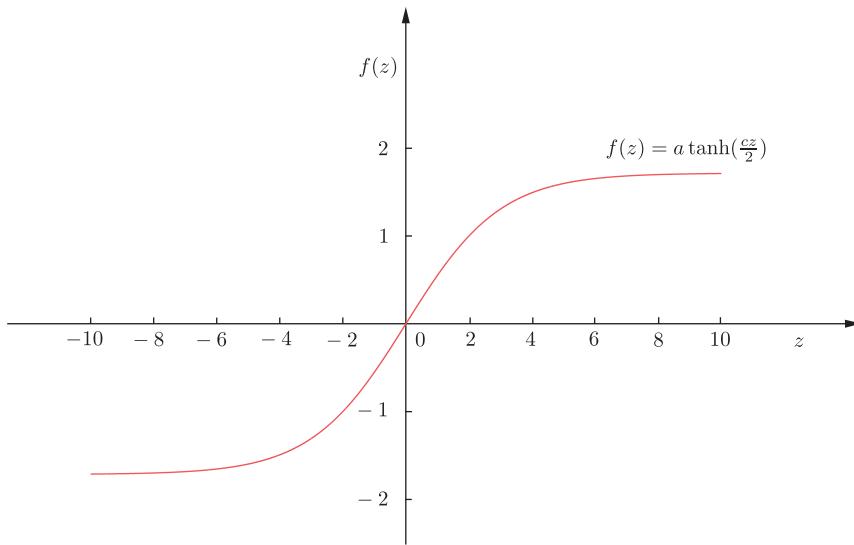
$$f(z) = a \tanh\left(\frac{cz}{2}\right), \quad (18.10)$$

where c and a are controlling parameters. The graph of this function is shown in Figure 18.9. Note that in contrast to the logistic sigmoid one, this is an antisymmetric function, that is, $f(-z) = -f(z)$. All these functions are also known as *squashing* functions, because they limit the output to a finite range of values.

18.4.1 THE GRADIENT DESCENT SCHEME

Having adopted a differentiable activation function, we are ready to proceed with developing the gradient descent iterative scheme for the minimization of the cost function. We will formulate the task in a general framework.

Let $(\mathbf{y}_n, \mathbf{x}_n)$, $n = 1, 2, \dots, N$, be the set of training samples. Note that we have assumed multiple output variables, assembled as a vector. We assume that the network comprises L layers; $L - 1$ hidden and one output layers. Each layer consists of k_r , $r = 1, 2, \dots, L$, neurons. Thus, the output vectors are

**FIGURE 18.9**

The hyperbolic tangent squashing function for $a = 1.7159$ and $c = 4/3$.

$$\mathbf{y}_n = [y_{n1}, y_{n2}, \dots, y_{nk_L}]^T \in \mathbb{R}^{k_L}, \quad n = 1, 2, \dots, N. \quad (18.11)$$

For the sake of the mathematical derivations, we also denote the number of input nodes as k_0 ; that is, $k_0 = l$, where l is the dimensionality of the input feature space.

Let $\boldsymbol{\theta}_j^r$ denote the synaptic weights associated with the j th neuron in the r th layer, with $j = 1, 2, \dots, k_r$ and $r = 1, 2, \dots, L$, where the bias term is included in $\boldsymbol{\theta}_j^r$, that is,

$$\boldsymbol{\theta}_j^r := [\theta_{j0}^r, \theta_{j1}^r, \dots, \theta_{jk_{r-1}}^r]^T. \quad (18.12)$$

The synaptic weights link the respective neuron to all neurons in layer k_{r-1} , see Figure 18.10. The basic iterative step for the gradient descent scheme is written as

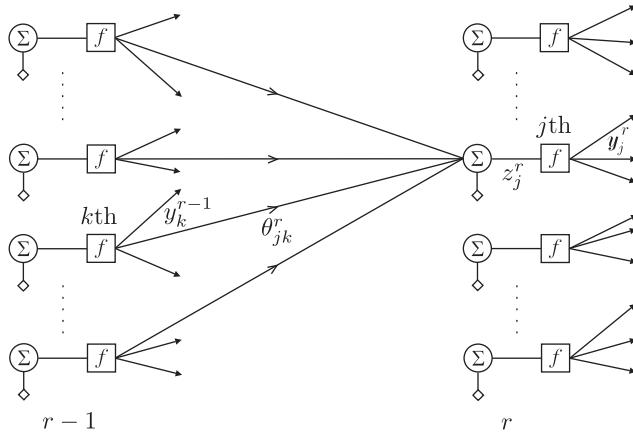
$$\boldsymbol{\theta}_j^r(\text{new}) = \boldsymbol{\theta}_j^r(\text{old}) + \Delta\boldsymbol{\theta}_j^r, \quad (18.13)$$

where

$$\Delta\boldsymbol{\theta}_j^r = -\mu \frac{\partial J}{\partial \boldsymbol{\theta}_j^r} \Bigg|_{\boldsymbol{\theta}_j^r(\text{old})}. \quad (18.14)$$

The parameter μ is the user-defined step-size (it can also be iteration-dependent) and J denotes the cost function. For example, if the squared error loss is adopted, we have

$$J(\boldsymbol{\theta}) = \sum_{n=1}^N J_n(\boldsymbol{\theta}), \quad (18.15)$$

**FIGURE 18.10**

The links and the associated variables of the j th neuron at the r th layer.

and

$$J_n(\boldsymbol{\theta}) = \frac{1}{2} \sum_{k=1}^{k_L} (\hat{y}_{nk} - y_{nk})^2, \quad (18.16)$$

where \hat{y}_{nk} , $k = 1, 2, \dots, k_L$, are the estimates provided at the corresponding output nodes of the network. We will consider them as the elements of a corresponding vector, $\hat{\mathbf{y}}_n$.

Computation of the gradients: Let z_{nj}^r denote the output of the linear combiner of the j th neuron in the r th layer at time instant n , when the pattern \mathbf{x}_n is applied at the input nodes. Then, we can write that

$$z_{nj}^r = \sum_{m=1}^{k_{r-1}} \theta_{jm}^r y_{nm}^{r-1} + \theta_{j0}^r = \sum_{m=0}^{k_{r-1}} \theta_{jm}^r y_{nm}^{r-1} = \boldsymbol{\theta}_j^r \mathbf{y}_n^{r-1}, \quad (18.17)$$

where by definition

$$\mathbf{y}_n^{r-1} := [1, y_{n1}^{r-1}, \dots, y_{nk_{r-1}}^{r-1}]^T, \quad (18.18)$$

and $y_{n0}^r \equiv 1$, $\forall r, n$. For the neurons at the output layer, $r = L$, $y_{nm}^L = \hat{y}_{nm}$, $m = 1, 2, \dots, k_L$, and for $r = 1$, we have $y_{nm}^0 = x_{nm}$, $m = 1, 2, \dots, k_0$; that is, y_{nm}^0 are set equal to the input feature values.

Hence, we can now write that

$$\frac{\partial J_n}{\partial \boldsymbol{\theta}_j^r} = \frac{\partial J_n}{\partial z_{nj}^r} \frac{\partial z_{nj}^r}{\partial \boldsymbol{\theta}_j^r} = \frac{\partial J_n}{\partial z_{nj}^r} \mathbf{y}_n^{r-1}. \quad (18.19)$$

Let us now define

$$\delta_{nj}^r := \frac{\partial J_n}{\partial z_{nj}^r}. \quad (18.20)$$

Then we have

$$\boxed{\Delta \theta_j^r = -\mu \sum_{n=1}^N \delta_{nj}^r y_n^{r-1}, \quad r = 1, 2, \dots, L.} \quad (18.21)$$

Computation of δ_{nj}^r : Here is where the heart of the backpropagation algorithm beats. For the computation of the gradients, δ_{nj}^r , one starts at the last layer, $r = L$, and proceeds *backwards* toward $r = 1$; this philosophy justifies the name given to the algorithm.

1. $r = L$: We have that

$$\delta_{nj}^L = \frac{\partial J_n}{\partial z_{nj}^L}. \quad (18.22)$$

For the squared error loss function,

$$J_n = \frac{1}{2} \sum_{k=1}^{k_L} \left(f(z_{nk}^L) - y_{nk} \right)^2. \quad (18.23)$$

Hence,

$$\begin{aligned} \delta_{nj}^L &= (\hat{y}_{nj} - y_{nj})f'(z_{nj}^L), \\ &= e_{nj}f'(z_{nj}^L), \quad j = 1, 2, \dots, k_L, \end{aligned} \quad (18.24)$$

where f' denotes the derivative of f , and e_{nj} is the error associated with the j th output variable at time n . Note that for the last layer, the computation of the gradient is straightforward.

2. $r < L$: Due to the successive dependence between the layers, the value of z_{nj}^{r-1} influences all the values z_{nk}^r , $k = 1, 2, \dots, k_r$ of the next layer. Employing the chain rule for differentiation, we get

$$\delta_{nj}^{r-1} = \frac{\partial J_n}{\partial z_{nj}^{r-1}} = \sum_{k=1}^{k_r} \frac{\partial J_n}{\partial z_{nk}^r} \frac{\partial z_{nk}^r}{\partial z_{nj}^{r-1}}, \quad (18.25)$$

or

$$\frac{\partial J_n}{\partial z_{nj}^{r-1}} = \sum_{k=1}^{k_r} \delta_{nk}^r \frac{\partial z_{nk}^r}{\partial z_{nj}^{r-1}}. \quad (18.26)$$

However,

$$\frac{\partial z_{nk}^r}{\partial z_{nj}^{r-1}} = \frac{\partial \left(\sum_{m=0}^{k_{r-1}} \theta_{km}^r y_{nm}^{r-1} \right)}{\partial z_{nj}^{r-1}}, \quad (18.27)$$

where,

$$y_{nm}^{r-1} = f(z_{nm}^{r-1}), \quad (18.28)$$

which leads to,

$$\frac{\partial z_{nk}^r}{\partial z_{nj}^{r-1}} = \theta_{kj}^r f'(z_{nj}^{r-1}), \quad (18.29)$$

and combining with (18.25)–(18.26), we obtain the recursive rule

$$\boxed{\delta_{nj}^{r-1} = \left(\sum_{k=1}^{k_r} \delta_{nk}^r \theta_{kj}^r \right) f'(z_{nj}^{r-1}), \quad j = 1, 2, \dots, k_{r-1}.} \quad (18.30)$$

For uniformity with (18.24), define

$$e_{nj}^{r-1} := \sum_{k=1}^{k_r} \delta_{nk}^r \theta_{kj}^r, \quad (18.31)$$

and we finally get

$$\delta_{nj}^{r-1} = e_{nj}^{r-1} f'(z_{nj}^{r-1}). \quad (18.32)$$

The only remaining computation is the derivative of f , which is easily shown to be equal to ([Problem 18.2](#))

$$f'(z) = af(z)(1 - f(z)). \quad (18.33)$$

The derivation has been completed and the backpropagation scheme is summarized in [Algorithm 18.3](#).

Algorithm 18.3 (The gradient descent backpropagation algorithm).

- Initialization
 - Initialize all synaptic weights and biases randomly with small, but not very small, values.
 - Select step size μ .
 - Set $y_{nj}^0 = x_{nj}$, $j = 1, 2, \dots, k_0 = l$, $n = 1, 2, \dots, N$.
- **Repeat;** Each repetition completes an epoch.
 - **For** $n = 1, 2, \dots, N$, **Do**
 - **For** $r = 1, 2, \dots, L$, **Do**; Forward computations.
 - **For** $j = 1, 2, \dots, k_r$, **Do**
 - Compute z_{nj}^r from (18.17).
 - Compute $y_{nj}^r = f(z_{nj}^r)$.
 - **End For**
 - **End For**
 - **For** $j = 1, 2, \dots, k_L$, **Do**
 - Compute δ_{nj}^L from (18.24).
 - **End For**
 - **For** $r = L, L-1, \dots, 2$, **Do**; Backward computations.
 - **For** $j = 1, 2, \dots, k_r$, **Do**
 - Compute δ_{nj}^{r-1} from (18.32).
 - **End For**
 - **End For**
 - **End For**
 - **For** $r = 1, 2, \dots, L$, **Do**; Update the weights.

- **For** $j = 1, 2, \dots, k_r$, **Do**
 - Compute $\Delta\theta_j^r$ from (18.21)
 - $\theta_j^r = \theta_j^r + \Delta\theta_j^r$
- **End For**
- **End For**
- **Until** a stopping criterion is met.

The backpropagation algorithm can claim a number of fathers. The popularization of the algorithm is associated with the classical paper [75], where the derivation of the algorithm is provided. However, the algorithm had been derived much earlier in [97]. The idea of backpropagation also appears in [14] in the context of optimal control.

Remarks 18.2.

- A number of criteria have been suggested for terminating the backpropagation algorithm. One possibility is to track the value of the cost function, and stop the algorithm when this gets smaller than a preselected threshold. An alternative path is to check for the gradient values and stop when these become small; this means that the values of the weights do not change much from iteration to iteration, see, for example, [48].
- As it is the case with all gradient descent schemes, the choice of the step-size, μ , is very critical; it has to be small to guarantee convergence, but not too small; otherwise convergence speed slows down. The choice depends a lot on the specific problem at hand. Adaptive values of μ , whose value depends on the iteration, are more appropriate, and they will be discussed soon.
- Due to the highly nonlinear nature of the NN problem, the cost function in the parameter space is in general of a complicated form and there are many local minima, where the algorithm can be trapped. If such a local minimum is deep enough, the obtained solution can be acceptable. However, this may not be the case and the solution can be trapped in a shallow minimum resulting in a bad solution. In practice, one reinitializes randomly the weights a number of times and keeps the best solution. Initialization has to be performed with care; we discuss this later on. A more recent direction for initialization will be discussed in Section 18.8, in the context of deep learning.
- *Pattern-by-pattern operation:* The scheme discussed in Algorithm 18.3 is of the *batch* type of operation, where the weights are updated once per epoch; that is, after all N training patterns have been presented to the algorithm. The alternative route is the *pattern-by-pattern/online* mode of operation; for this case, the weights are updated at every time instant when a new pattern appears in the input. An intermediate way, where the update is performed every $N_1 < N$ samples, has also been considered, which is referred to as a *mini-batch* mode of operation. Batch and mini-batch modes of operation have an averaging effect on the computation of the gradients. In [78], it is advised to add a small white noise sequence to the training data, which may have a beneficial effect for the algorithm to escape from a poor local minimum.

The pattern-by-pattern mode leads to a less smooth convergence trajectory; however, such a randomness may have the advantage of helping the algorithm to escape from a local minimum. To exploit randomness even further in the pattern-by-pattern mode, it is advisable that prior to the pass of a new epoch, the sequence in which data are presented to the algorithm is randomized, see, for example, [28]. This has no meaning in the batch mode, because updates take place once all data have been considered. In practice, the pattern-by-pattern version of the backpropagation seems to converge faster and give better solutions.

Online versions exploit better the training set, when redundancies in the data are present or training samples are very similar. Averaging, as it is done in the batch mode, wastes resources, because averaging the contribution to the gradient of similar patterns does not add much information. In contrast, in the pattern-by-pattern mode of operation, all examples are equally exploited, inasmuch as an update takes place for each one of the patterns.

Speeding up the convergence rate

The basic gradient descent scheme inherits all the advantages (low computational demands) and all the disadvantages (slow convergence rate) of the gradient descent algorithmic family, as it was first presented in this book in [Chapter 5](#). To speed up the convergence rate, a large research effort has been invested in the late 1980s and early 1990s and a number of variants of the basic gradient descent backpropagation scheme have been proposed. In this section, we provide some directions that seem to be more popular in practice.

Gradient descent with a momentum term: One way to improve the convergence rate, while remaining within the gradient descent rationale, is to employ the so-called *momentum term* [24, 99]. The correction term in (18.21) is now modified as

$$\Delta\theta_j^r(\text{new}) = a\Delta\theta_j^r(\text{old}) - \mu \sum_{n=1}^N \delta_{np}^r \mathbf{y}_n^{r-1}, \quad (18.34)$$

where a is the momentum factor. In other words, the algorithm takes into account the correction used in the previous iteration step as well as the current gradient computations. Its effect is to increase the step size, in regions where the cost function exhibits low curvature. Assuming that the gradient is approximately constant over, say I , successive iterations, it can be shown ([Problem 18.3](#)) that using the momentum term the updates are equivalent to

$$\Delta\theta_j^r(I) \approx -\frac{\mu}{1-a} \mathbf{g}, \quad (18.35)$$

where \mathbf{g} is the gradient value over the I successive iteration steps. Typical values of a are in the range of 0.1 to 0.8. It has been reported that the use of a momentum term can speed up the convergence rate up to a factor of two [79]. Experience seems to suggest that the use of a momentum factor helps the batch mode of operation more than the online version.

Iteration-dependent step-size: A heuristic variant of the previous backpropagation versions results if the step-size is left to vary as iterations progress. A rule is to change its value according to whether the cost function in the current iteration step is larger or smaller compared to the previous one. Let $J^{(i)}$ be the computed cost value at the current iteration. Then if $J^{(i)} < J^{(i-1)}$, the learning rate is increased by a factor of r_i . If, on the other hand, the new value is larger than the previous one by a factor larger than c , then the learning rate is reduced by a factor of r_d . Otherwise, the same value is kept. Typical values for the involved parameters are $r_i = 1.05$, $r_d = 0.7$, and $c = 1.04$. For iteration steps where the value of the cost increases, it is advisable to set the momentum factor equal to zero. Another possibility is not to perform the update whenever the cost increases. Such techniques, also known as *adaptive momentum*, are more appropriate for batch processing, because for online versions the values of the cost tend to oscillate from iteration to iteration.

Using different step-size for each weight: It is beneficial for improving the convergence rate, to employ a *different step-size* for each individual weight; this gives the freedom to the algorithm to exploit better the dependence of the cost function on each direction in the parameter space. In [46], it is suggested to increase the learning rate, associated with a weight, if the respective gradient value has the same sign for two successive iterations. Conversely, the learning rate is decreased if the sign changes, because this is indicative of possible oscillation.

Some practical hints

Training an NN still has a lot of practical engineering flavor compared to mathematical rigorousness. In this section, I will present some practical hints that experience has shown to be useful in improving the performance of the backpropagation algorithm; see, for example, [51] for a more detailed discussion.

Preprocessing the input features/variables: It is advisable to preprocess the input variables so they have (approximately) zero mean over the training set. Also, one should scale them so they all have similar variances, assuming that all variables are equally important. Their variance should also match the range of values of the activation (squashing) function. For example, for the hyperbolic tangent activation function, a variance of the order of one seems to be a good choice. Moreover, it is beneficial for the convergence of the algorithm if the input variables are uncorrelated. This can be achieved via an appropriate transform, for example, PCA.

Selecting symmetric activation functions: For the same reason that it is beneficial for the convergence when the inputs have zero mean, it is desirable that the outputs of the neurons assume equally likely positive and negative values. After all, the outputs of one layer become inputs to the next. To this end, the hyperbolic activation function in Eq. (18.10) can be used. Recommended values are $a = 1.7159$ and $c = 4/3$. These values guarantee that if the inputs are preprocessed as suggested before, that is, to be normalized to variances equal to one, then the variance at the output of the activation function is also equal to one and the respective mean value equal to zero.

Target values: The target values should be carefully chosen to be in line with the activation function used. The values should be selected to offset by some small amount the limiting value of the squashing function. Otherwise, the algorithm tends to push the weights to large values and this slows down the convergence; the activation function is driven to saturation, making the derivative of the activation function very small, which in turn renders small gradient values. For the hyperbolic tangent function, using the parameters discussed before, the choice of ± 1 for the target class labels seems to be the right one. Note that in this case, the saturation values are $a = \pm 1.7158$.

Initialization: The weights should be initialized randomly to values of small magnitude. If they are initialized to large values, then all activation functions will operate in their saturation point, making the gradients small, which slows down convergence. The effect on the gradients is the same, when the weights are initialized to very small values. Initialization must be done so that the operation in each neuron takes place in the (approximate) linear region of the graph of the activation function and not in the saturated one. It can be shown ([51], Problem 18.4) that if the input variables are preprocessed to zero mean and unit variance, and the tangent hyperbolic function is used with parameter values as discussed before, then the best choice for initializing the weights is to assign values drawn from a distribution with zero mean and standard deviation equal to

$$\sigma = m^{-1/2},$$

where m is the number of synaptic connections in the corresponding neuron.

18.4.2 BEYOND THE GRADIENT DESCENT RATIONALE

The other path to follow to improve upon the convergence rate of the gradient descent-based backpropagation algorithm, at the expense of increased complexity, is to resort to schemes that involve, in one way or another, information related to the second order derivatives. We have already discussed such families in this book, for example, the Newton family introduced in [Chapter 6](#). For each one of the available families, a backpropagation version can be derived to serve the needs of the NN training. We will not delve into details, because the concept remains the same as that discussed for the gradient descent. The difference is that now second order derivatives have to be propagated backwards. The interested reader can look at the respective references and also in [12, 16, 26, 51, 102] for more details.

In [4, 47, 48] schemes based on the conjugate gradient philosophy have been developed and members of the Newton family have been proposed in, for example, [6, 69, 95]. In all these schemes, the computation of the elements of the Hessian matrix, that is,

$$\frac{\partial^2 J}{\partial \theta_{jk}^r \partial \theta_{j'k'}^{r'}},$$

is required. To this end, various simplifying assumptions are employed in the different papers (see, also, [Problems 18.5](#) and [18.6](#)).

A popular algorithm, which is loosely based on Newton's scheme, has been proposed in [20], and it is known as the *quickprop* algorithm. It is a heuristic method that treats the synaptic weights as if they were quasi-independent. It then approximates the error surface, as a function of each weight, via a quadratic polynomial. If this has its minimum at a sensible value, the latter is used as the new weight for the iterations; otherwise, a number of heuristics are mobilized. A common formulation for the resulting updating rule is given by,

$$\Delta \theta_{ij}^r(\text{new}) = \begin{cases} a_{ij}^r(\text{new}) \Delta \theta_{ij}^r(\text{old}), & \text{if } \Delta \theta_{ij}^r(\text{old}) \neq 0, \\ -\mu \frac{\partial J}{\partial \theta_{ij}^r}, & \text{if } \Delta \theta_{ij}^r(\text{old}) = 0, \end{cases} \quad (18.36)$$

where

$$a_{ij}^r(\text{new}) = \min \left\{ \frac{\frac{\partial J(\text{new})}{\partial \theta_{ij}^r}}{\frac{\partial J(\text{old})}{\partial \theta_{ij}^r} - \frac{\partial J(\text{new})}{\partial \theta_{ij}^r}}, a_{\max}^r \right\}, \quad (18.37)$$

with typical values of the parameters used being $0.01 \leq \mu \leq 0.6$, and $a_{\max} \approx 1.75$. An algorithm similar in spirit with the quickprop has been proposed in [70].

In practice, when large networks and data sets are involved, simpler methods, such as carefully tuned gradient descent schemes, seem to work better than more complex second order techniques. The latter can offer improvements in smaller networks, especially in the context of regression tasks. The careful tuning of NNs, especially when they are large, is of *paramount importance*. The deep learning techniques to be discussed soon, when used as part of a pre-training phase of NNs, can be seen as an attempt for well-tuned initialization.

18.4.3 SELECTING A COST FUNCTION

As we have already commented, a feed-forward NN belongs to the more general class of parametric modeling; thus, in principle, any loss function we have met so far in this book can be employed to replace the least-squares one. Over the years, certain loss functions have gained in popularity in the context of NNs for classification tasks.

If one adopts as targets the 0, 1 values, then the true and predicted values, y_{nm}, \hat{y}_{nm} , $n = 1, 2, \dots, N$, $m = 1, 2, \dots, k_L$, can be interpreted as probabilities and a commonly used cost function is the *cross-entropy*, which is defined as,

$$J = - \sum_{n=1}^N \sum_{k=1}^{k_L} (y_{nk} \ln \hat{y}_{nk} + (1 - y_{nk}) \ln(1 - \hat{y}_{nk})) : \text{ Cross-Entropy Cost,} \quad (18.38)$$

which takes its minimum value when $y_{nk} = \hat{y}_{nk}$, which for binary target values is equal to zero.

An interpretation of the cross-entropy cost comes from the following observation: the vector of target values, $\mathbf{y}_n \in \mathbb{R}^{k_L}$, has a single element equal to one, which indicates the class of the corresponding input pattern, \mathbf{x}_n ; the rest of the elements are zero. Viewing each component, \hat{y}_{nm} , as the probability of obtaining a one at the respective node (class), then the probability $P(\mathbf{y}_n)$ is given by

$$P(\mathbf{y}_n) = \prod_{k=1}^{k_L} (\hat{y}_{nk})^{y_{nk}} (1 - \hat{y}_{nk})^{1-y_{nk}}. \quad (18.39)$$

Then, it is straightforward to see that the cross-entropy cost function in (18.38) is the negative log-likelihood of the training samples. It can be shown ([Problem 18.7](#)) that, the cross-entropy function depends on the relative errors and not on the absolute errors, as is the case in the LS loss; thus, small and large error values are equally weighted during the optimization. Furthermore, it can be shown that the cross-entropy belongs to the so-called well-formed loss functions, in the sense that if there is a solution that classifies correctly all the training data, the gradient descent scheme will find it [[2](#)]. In [[80](#)], it is pointed out that the cross-entropy loss function may lead to improved generalization and faster training for classification, compared to the LS loss.

An alternative cost results if the similarity between y_{nk} and \hat{y}_{nk} is measured in terms of the *relative entropy* or KL divergence,

$$J = - \sum_{n=1}^N \sum_{k=1}^{k_L} y_{nk} \ln \frac{\hat{y}_{nk}}{y_{nk}} : \text{ Relative Entropy Cost.} \quad (18.40)$$

Although we have interpreted the outputs of the nodes as probabilities, there is no guarantee that these add to one. This can be enforced by selecting the activation function in the last layer of nodes to be

$$\hat{y}_{nk} = \frac{\exp(z_{nk}^L)}{\sum_{m=1}^{k_L} \exp(z_{nm}^L)} : \text{ Softmax Activation Function,} \quad (18.41)$$

which is known as the *softmax activation* function [13]. It is easy to show that in this case, δ_{nj}^L required by the backpropagation algorithm is equal to $\hat{y}_j - y_j$ (Problem 18.9). A more detailed discussion on various cost functions can be found in, for example, [88].

18.5 PRUNING THE NETWORK

A crucial factor in training NNs is to decide the size of the network. The size is directly related to the number of weights to be estimated. We know that in any parametric modeling method, if the number of free parameters is large enough with respect to the number of training data, overfitting will occur.

Concerning feed-forward neural networks, two issues are involved. The first concerns the number of layers and the other the number of neurons per layer. As we will discuss in Section 18.8, a number of factors support the use of more than two hidden layers. However, experience has shown that trying to train such NNs via algorithms inspired by the backpropagation philosophy alone, will fail to obtain a reasonably good solution, due to the complicated shape of the cost function in the parameter space. Thus, in practice, one has to use at most two hidden layers. Otherwise, it seems that more sophisticated training techniques have to be adopted. Coming to the second issue, there is no theoretically supported model to assist the prediction of the number of neurons per layer. In practice, the most common technique is to start with a large enough number of neurons and then use a regularization technique to push the less informative weights to low values. A number of different regularization approaches have been proposed over the years. A brief presentation and some guidelines are given in the sequel.

Weight decay: This path refers to a typical cost function regularization via the Euclidean norm of the weights. Instead of minimizing a cost function, $J(\boldsymbol{\theta})$, its regularized version is used, such that,

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|^2. \quad (18.42)$$

Although this simple type of regularization helps in improving the generalization performance of the network, and it can be sufficient for some cases, in general it is not the most appropriate way to go. We have already discussed in Chapter 3 in the context of ridge regression that, involving the bias terms in the regularizing norm is not a good practice, because it affects the translation invariant property of the estimator. A more sensible way to regularize is to remove the bias terms from the norm. Moreover, it is even better if one groups the parameters of different layers together and employs different regularizing constants for each group.

Weight elimination: Instead of employing the norm of the weights, another approach involves more general functions for the regularization term, that is,

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda h(\boldsymbol{\theta}). \quad (18.43)$$

For example, in [96] the following is used

$$h(\boldsymbol{\theta}) = \sum_{k=1}^K \frac{\theta_k^2}{\theta_h^2 + \theta_k^2}, \quad (18.44)$$

where K is the total number of the weights involved and θ_h is a preselected threshold value. A careful look at this function reveals that if $\theta_k < \theta_h$ the penalty term goes to zero very fast. In contrast, for values $\theta_k > \theta_h$, the penalty term tends to unity. In this way, less significant weights are pushed toward to zero. A number of variants of this method have also appeared, for example, [76].

Methods based on sensitivity analysis: In [50], the so-called *optimal brain damage* technique is proposed. A perturbation analysis of the cost function in terms of the weights is performed, via the second order Taylor expansion, that is,

$$\delta J = \sum_{i=1}^K g_i \delta \theta_i + \frac{1}{2} \sum_{i=1}^K h_{ii} \delta \theta_i^2 + \frac{1}{2} \sum_{i=1}^K \sum_{j=1, j \neq i}^K h_{ij} \delta \theta_i \delta \theta_j, \quad (18.45)$$

where,

$$g_i := \frac{\partial J}{\partial \theta_i} \quad h_{ij} := \frac{\partial^2 J}{\partial \theta_i \partial \theta_j}.$$

Then, assuming the Hessian matrix to be diagonal and if the algorithm operates near the optimum (zero gradient), we can approximately set

$$\delta J \approx \frac{1}{2} \sum_{i=1}^K h_{ii} \delta \theta_i^2. \quad (18.46)$$

The method works as follows:

- The network is trained using the backpropagation algorithm. After a few iteration steps, the training is frozen.
- The so called *saliencies*, defined as

$$s_i = \frac{h_{ii} \theta_i^2}{2},$$

are computed for each weight, and weights with a small saliency are removed. Basically, the saliency measures the effect on the cost function, if one removes (sets equal to zero) the respective weight.

- Training is continued and the process is repeated, until a stopping criterion is satisfied.

In [25], the full Hessian matrix is computed, giving rise to the *optimal brain surgeon* method.

Early stopping: An alternative primitive technique to avoid overfitting is the so-called *early stopping*. The idea is to stop the training when the test error starts increasing. Training the network over many epochs can lead the training error to converge to small values. However, this is an indication of overfitting rather than indicative of a good solution. According to the early stopping method, training is performed for some iterations and then it is frozen. The network, using the currently available estimates of the weights/biases, is used with a validation/test data set and the value of the cost function is computed. Then training is resumed, and after some iterations the previous process is repeated. When the value of the cost function, computed on the test set, starts increasing then training is stopped.

Remarks 18.3.

- *Weight Sharing:* One major issue encountered in many classification tasks is that of transformation invariance. This means that the classifier should classify correctly, independent of transformations performed on the input space, such as translation, rotation, and scaling. For example, the character 5 should “look the same” to an OCR system, irrespective of its position, orientation, and size. There are various ways to approach this problem. One is to choose appropriate feature vectors, which are invariant under such transformations, see, for example, [88]. Another way is to make the

classifier responsible for it in the form of *built-in constraints*. Weight sharing is such a constraint, which forces certain connections in the network to have the same weights, for example, [65].

- *Convolutional Networks:* This is a very successful example of networks that are built around the weight-sharing rationale. Convolutional networks have been inspired by the structural architecture of our visual system, for example, [42], and have been particularly successful in machine vision and optical character recognition schemes, where the inputs are images. Networks developed on these ideas are based on local connectivities between neurons and on hierarchically organized transformations of the image. Nodes form groups of two-dimensional arrays known as *feature maps*. Each node in a given map receives inputs from a specific window area of the previous layer, known as its *receptive field*. Translation invariance is imposed by forcing corresponding nodes in the same map, looking at different receptive fields, to share weights. Thus, if an object moves from one input receptive field to the other, the network responds in the same way.

The first such architecture was proposed in [22] and it works in an unsupervised training mode. A supervised version of it was proposed in [49]; see also [52]. It turns out that such architectures closely resemble the physiology of our visual system, at least as far as the quick recognition of objects is concerned [77].

A very interesting aspect of these networks is that they can have many hidden layers, without facing problems in their training. Training a general-purpose feed-forward NN with many layers, using standard backpropagation-type algorithms and random initialization, would be impossible; we will come back to this issue soon. Thus, convolutional networks are notable early successful examples of deep architectures.

18.6 UNIVERSAL APPROXIMATION PROPERTY OF FEED-FORWARD NEURAL NETWORKS

In Section 18.3, the classification power of a three-layer feed-forward NN, built around the McCulloch-Pitts neuron, was discussed. Then we moved on to employ smooth versions of the activation function, for the sake of differentiability. The issue now is whether we can say something more concerning the prediction power of such networks. It turns out that some strong theoretical results have been produced, which provide support for the use of NNs in practice, see, for example, [17, 23, 39, 45].

Let us consider a *two-layer* network, with one hidden layer and with a single output *linear* node. The output of the network is then written as

$$\hat{g}(\mathbf{x}) = \sum_{k=1}^K \theta_k^o f(\boldsymbol{\theta}_k^{hT} \mathbf{x}) + \theta_0^o, \quad (18.47)$$

where $\boldsymbol{\theta}_k^h$ denotes the synaptic weights and bias term defining the k th hidden neuron and the superscript “ o ” refers to the output neuron. Then, the following theorem holds true.

Theorem 18.1. *Let $g(\mathbf{x})$ be a continuous function defined in a compact⁴ subset $S \subset \mathbb{R}^l$ and any $\epsilon > 0$. Then there is a two layer network with $K(\epsilon)$ hidden nodes of the form in Eq. (18.47), so that*

$$|g(\mathbf{x}) - \hat{g}(\mathbf{x})| < \epsilon, \quad \forall \mathbf{x} \in S. \quad (18.48)$$

⁴ Closed and bounded.

In [5], it is shown that the approximation error decreases according to an $\mathcal{O}(1/K)$ rule. In other words, the input dimensionality does not enter into the scene and the error depends on the number of neurons used. The theorem states that a two-layer NN network is sufficient to approximate any continuous function; that is, it can be used to realize any nonlinear discriminant surface in a classification task or any nonlinear function for prediction in a general regression problem. This is a strong theorem indeed. However, what the theorem does not say is how big such a network can be in terms of the required number of neurons in the single layer. It may be that a very large number of neurons are needed to obtain a good enough approximation. This is where the use of more layers can be advantageous. Using more layers, the overall number of neurons needed to achieve certain approximation may be much smaller. We will come to this issue soon, when discussing deep architectures.

Remarks 18.4.

- *Extreme Learning Machines* (ELMs): These are single-layered feed-forward networks (SLFNs) with output of the form [40]:

$$g_K(\mathbf{x}) = \sum_{i=1}^K \theta_i^o f(\theta_i^h T \mathbf{x} + b_i), \quad (18.49)$$

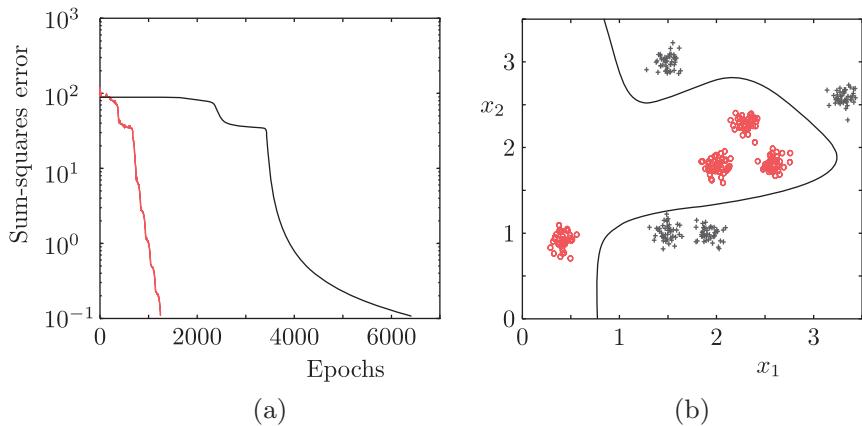
where f is the respective activation function and K is the number of hidden nodes. The main difference with standard SLFNs is that the weights of each node (i.e., θ_i^h and b_i) are generated *randomly*, whereas the weights of the output function (i.e., θ_i^o) are selected so that the squared error over the training points is minimized. This implies solving:

$$\min_{\theta} \sum_{n=1}^N (y_n - g_K(\mathbf{x}_n))^2, \quad (18.50)$$

Hence, according to the ELM rationale, we do not need to compute the values of the parameters for the hidden layer. It turns out that such a training philosophy has a solid theoretical foundation, as convergence to a unique solution is guaranteed. It is interesting to note that, although the node parameters are randomly generated, for infinitely differentiable activation functions, the training error can become arbitrarily small, if K approaches N (it becomes zero if $K = N$). Furthermore, the universal approximation theorem ensures that for sufficiently large values of K and N , g_K can approximate any nonconstant piece-wise continuous function [41]. A number of variations and generalizations of this simple idea can be found in the respective literature. The interested reader is referred to, for example, [44, 67], for related reviews.

Example 18.1. In this example, the capability of a multilayer perceptron to classify nonlinearly separable classes is demonstrated. The classification task consists of two classes, each being the union of four regions in the two-dimensional space. Each region consists of normally distributed random vectors with statistically independent components and each with variance $\sigma^2 = 0.08$. The mean values are different for each of the regions. Specifically, the regions of the class denoted by a red \circ (see Figure 18.11) are formed around the mean vectors

$$[0.4, 0.9]^T, [2.0, 1.8]^T, [2.3, 2.3]^T, [2.6, 1.8]^T$$

**FIGURE 18.11**

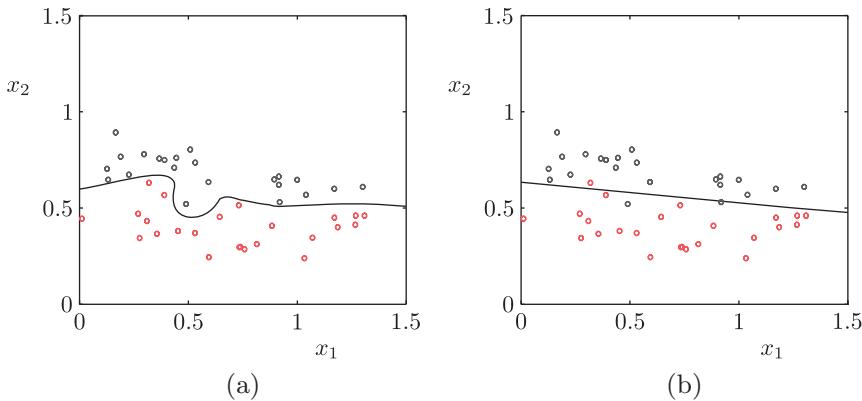
(a) Error convergence curves for the adaptive momentum (red line) and the momentum algorithms, for [Example 18.1](#). Note that the adaptive momentum leads to faster convergence. (b) The classifier formed by the multilayer perceptron.

and those of the class denoted by a black + around the values

$$[1.5, 1.0]^T, [1.9, 1.0]^T, [1.5, 3.0]^T, [3.3, 2.6]^T.$$

A total of 400 training vectors were generated, 50 from each distribution. A multilayer perceptron with three neurons in the first and two neurons in the second hidden layer were used, with a single output neuron. The activation function was the logistic one with $a = 1$ and the desired outputs 1 and 0, respectively, for the two classes. Two different algorithms were used for the training, namely the momentum and the adaptive momentum; see discussion after [Remarks 18.2](#). After some experimentation, the parameters employed were (a) for the momentum $\mu = 0.05$, $\alpha = 0.85$ and (b) for the adaptive momentum $\mu = 0.01$, $\alpha = 0.85$, $r_i = 1.05$, $c = 1.05$, $r_d = 0.7$. The weights were initialized by a uniform pseudorandom distribution between 0 and 1. [Figure 18.11a](#) shows the respective output error convergence curves for the two algorithms as a function of the number of epochs. The respective curves can be considered typical and the adaptive momentum algorithm leads to faster convergence. Both curves correspond to the batch mode of operation. [Figure 18.11b](#) shows the resulting classifier using the weights estimated from the adaptive momentum training.

A second experiment was conducted in order to demonstrate the effect of the pruning. [Figure 18.12](#) shows the resulting decision lines separating the samples of the two classes, denoted by black and red o respectively. [Figure 18.12a](#) corresponds to a multilayer perceptron with two hidden layers and 20 neurons in each of them, amounting to a total of 480 weights. Training was performed via the backpropagation algorithm. The overfitting nature of the resulting curve is readily observed. [Figure 18.12b](#) corresponds to the same multilayer perceptron trained with a pruning algorithm. Specifically, the method based on parameter sensitivity was used, testing the saliency values of the weights every 100 epochs and removing weights with saliency value below a chosen threshold. Finally, only 25 of the 480 weights survived and the curve is simplified to a straight line.

**FIGURE 18.12**

Decision curve (a) before pruning and (b) after pruning.

18.7 NEURAL NETWORKS: A BAYESIAN FLAVOR

In Chapter 12, the (generalized) linear regression and the classification tasks were treated in the framework of Bayesian learning. Because a feed-forward neural network realizes a parametric input-output mapping, $f_{\theta}(\mathbf{x})$, there is nothing to prevent us from looking at the problem from a fully statistical point of view.

Let us focus on the regression task and assume that the noise variable is a zero mean Gaussian one. Then, the output variable, given the value of $f_{\theta}(\mathbf{x})$, is described in terms of a Gaussian distribution,

$$p(y|\theta; \beta) = \mathcal{N}(y|f_{\theta}(\mathbf{x}), \beta^{-1}), \quad (18.51)$$

where β is the noise precision variable. Assuming successive training samples, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$, to be independent, we can write

$$p(\mathbf{y}|\theta; \beta) = \prod_{n=1}^N \mathcal{N}(y_n|f_{\theta}(\mathbf{x}_n), \beta^{-1}). \quad (18.52)$$

Adopting a Gaussian prior for θ , that is,

$$p(\theta|\alpha) = \mathcal{N}(\theta|\mathbf{0}, \alpha^{-1}I), \quad (18.53)$$

the posterior distribution, given the output values \mathbf{y} , can be written as

$$p(\theta|\mathbf{y}) \propto p(\theta|\alpha)p(\mathbf{y}|\theta; \beta). \quad (18.54)$$

However, in contrast to Eq. (12.16), the posterior is not a Gaussian one, owing to the nonlinearity of the dependence on θ . Here is where complications arise and one has to employ a series of approximations to deal with it.

Laplacian approximation: The Laplacian approximation method, introduced in Chapter 12, is adopted to approximate $p(\theta|\mathbf{y})$ to a Gaussian one. To this end, the maximum, θ_{MAP} , has to be computed,

which is carried out via an iterative optimization scheme. Once this is found, the posterior can be replaced by a Gaussian approximation, denoted as, $q(\theta|y)$.

Taylor expansion of the neural network mapping: The final goal is to compute the predictive distribution,

$$p(y|x,y) = \int p(y|f_\theta(x))q(\theta|y) d\theta. \quad (18.55)$$

However, although the involved pdfs are Gaussians, the integration is intractable, because of the nonlinear nature of f_θ . In order to carry this out, a first order Taylor expansion is performed,

$$f_\theta(x) \approx f_{\theta_{\text{MAP}}}(x) + g^T(\theta - \theta_{\text{MAP}}), \quad (18.56)$$

where g is the respective gradient computed at θ_{MAP} , which can be computed using backpropagation arguments. After this linearization, the involved pdfs become linear with respect to θ and the integration leads to an approximate Gaussian predictive distribution as in Eq. (12.21).

For the classification, instead of the Gaussian pdf, the logistic regression model as in Section 13.7.1 of Chapter 13 is adopted and similar approximations as before are employed.

More on the Bayesian approach to NNs can be obtained in [56, 57]. In spite of their theoretical interest, the Bayesian approach has not been widely adopted in practice, compared to their backpropagation-based algorithmic relatives.

18.8 LEARNING DEEP NETWORKS

In our tour so far in this chapter, we have discussed various aspects of learning networks with more than two layers of nodes. The backpropagation algorithm, in its various formulations, was introduced as a popular scheme for training multilayer architectures. We also established some very important features of the multilayer perceptrons concerning their universal approximation property and also their power to solve any classification task comprising classes formed by the union of polyhedra regions in the input space. Two or three layers were, theoretically, enough to perform such tasks. Thus, it seems that everything has been said. Unfortunately (or maybe fortunately) this is far from the truth.

Multilayer perceptrons, after almost two decades of intense research, lost their initial glory and were superseded, to a large extent, by other techniques, such as kernel-based schemes, boosting and boosted trees, and Bayesian learning methods. A major reason for this loss of popularity was that their training can become difficult and often backpropagation-related algorithms are stuck in local minima. Although improvements can be obtained by trying different practical “tricks,” such as multiple training using random initialization, still their generalization performance may not be competitive with other methods. This drawback becomes more severe if more than two hidden layers are used. The more layers one uses, the more difficult the training becomes and the probability to recover solutions corresponding to poor local minima is increased. As a matter of fact, efforts to use more than two hidden layers were soon abandoned.

In this section, we are going to focus on the following two issues:

- Is there any need for networks with more than two or three layers?
- Is there a training scheme, beyond or complementary to the backpropagation algorithm, to assist the optimization process to settle in a “good” local minimum, by extracting and exploiting more information from the input data?

Answers to both these points will be presented, starting with the first one.

18.8.1 THE NEED FOR DEEP ARCHITECTURES

In Section 18.3, we discussed how each layer of a neural network provides a different description of the input patterns. The input layer describes each pattern as a point in the feature space. The first hidden layer of nodes (using the Heaviside activation) forms a partition of the input space and places the input point in one of the regions, using a coding scheme of zeros and ones at the outputs of the respective neurons. This can be considered as a more abstract representation of our input patterns. The second hidden layer of nodes, based on the information provided by the previous layer, encodes information related to the classes; this is a further representation abstraction, which carries some type of “semantic meaning.” For example, it provides the information of whether a tumor is malignant or benign, in a related medical application.

The previously reported hierarchical type of representation of the input patterns mimics the way that a mammal’s brain follows to “understand” and “sense” the world around us; in the case of humans, this is the physical mechanism in the brain on which *intelligence* is built. The brain of the mammals is organized in a number of layers of neurons, and each layer provides a different representation of the input percept. In this way, different levels of abstraction are formed, via a hierarchy of transformations. For example, in the primate visual system, this hierarchy involves detection of edges, primitive shapes, and as we move to higher hierarchy levels, more complex visual shapes are formed, until finally a semantics concept is established; for example, a car moving in a video scene, a person sitting in an image. The cortex of our brain can be seen as a multilayer architecture with 5-10 layers dedicated only to our visual system, [77].

An issue that is now raised is whether one can obtain an equivalent input-output representation via a relatively simple functional formulation (such as the one implied by the support vector machines) or via networks with less than three layers of neurons/processing elements, maybe at the expense of more elements per layer.

The answer to the first point is yes, as long as the input-output dependence relation is simple enough. However, for more complex tasks, where more complex concepts have to be learned, for example, recognition of a scene in a video recording, language and speech recognition, the underlying functional dependence is of a very complex nature so that we are unable to express it analytically in a simple way.

The answer to the second point, concerning networks, lies in what is known as *compactness* of representation. We say that a network, realizing an input-output functional dependence, is compact if it consists of relatively few free parameters (few computational elements) to be learned/tuned during the training phase. Thus, for a given number of training points, we expect compact representations to result in better generalization performance.

It turns out that using networks with more layers, one can obtain more compact representations of the input-output relation. Although there are not theoretical findings for general learning tasks to prove such a claim, theoretical results from the theory of circuits of Boolean functions suggest that a function, which can compactly be realized by, say, k layers of logic elements, may need an exponentially large number of elements if it is realized via $k - 1$ layers. Some of these results have been generalized and are valid for learning algorithms in some special cases. For example, the parity function with l inputs requires $\mathcal{O}(2^l)$ training samples and parameters to be represented by a Gaussian support vector machine, $\mathcal{O}(l^2)$ parameters for a neural network with one hidden layer, $\mathcal{O}(l)$ parameters and nodes for a multilayer network with $\mathcal{O}(\log_2 l)$ layers; see, for example, [7, 8, 64].

Such arguments may seem a bit confusing, because we have already stated that networks with two layers of nodes are universal approximators for a certain class of functions. However, this theorem

does not say how one can achieve this in practice. For example, any continuous function can be approximated arbitrarily close by a sum of monomials. Nevertheless, a huge number of monomials may be required, which is not practically feasible. In any learning task, we have to be concerned with what is feasibly “learnable” in a given representation. The interested reader may refer to, for example, [90] for a discussion on the benefits one is expected to get when using many-layer architectures.

Let us now elaborate a bit more on the aforementioned issues and also make bridges to schemes discussed in previous chapters. Recall from [Chapter 11](#) that nonparametric techniques, modeling the input-output relation in RKH spaces, establish a functional dependence of the form,

$$f(\mathbf{x}) = \sum_{n=1}^N \theta_n \kappa(\mathbf{x}, \mathbf{x}_n) + \theta_0. \quad (18.57)$$

This can be seen as a network with one hidden layer, whose processing nodes perform kernel computations and the output node performs a linear combination. As already commented in [Section 11.10.4](#), the kernel function, $\kappa(\mathbf{x}, \mathbf{x}_n)$, can be thought of as a measure of similarity between \mathbf{x} and the respective training sample, \mathbf{x}_n . For kernels such as the Gaussian one, the action of the kernel function is of a local nature, in the sense that the contribution of $\kappa(\mathbf{x}, \mathbf{x}_n)$ in the summation tends to zero as the distance of \mathbf{x} from \mathbf{x}_n increases (the rate of decreasing influence depends on the variance σ^2 of the Gaussian). Thus, if the true input-output functional dependence undergoes fast variations, then a large number of such local kernels will be needed to model sufficiently well the input-output relation. This is natural, as one attempts to approximate a fast-changing function in terms of smooth bases of a local extent. Similar arguments hold true for the Gaussian processes discussed in [Chapter 13](#). Besides the kernel methods, other widely used learning schemes are also of a local nature, as is the case for the decision trees, discussed in [Chapter 7](#). This is because the input space is partitioned into regions via rules that are local for each one of the regions.

In contrast, assuming that these variations are not random in nature but that there exist underlying (unknown) regularities, resorting to models with a more compact representation, such as networks with many layers, one expects to learn the regularities and exploit them to improve the performance. As stated in [72], exploiting the regularities that are hidden in the training data is likely to design an excellent predictor for future events. The interested reader may explore more on these issues from the insightful tutorial [9].

From now on, we will refer to the number of layers in a network as the *depth* of the network. Networks with up to three (two hidden) layers, are known as *shallow*, whereas those with more than three are called *deep* networks. The main issue associated with a deep architecture is its training. As said before, the use of backpropagation fails to provide satisfactory generalization performance. A breakthrough that paved the way for training such large networks was proposed in [34].

18.8.2 TRAINING DEEP NETWORKS

A new philosophy for training deep networks was proposed in [34]. The main idea is to *pre-train* each layer, via an *unsupervised* learning algorithm, one layer at a time, in a *greedy-like* rationale. Different options are open for selecting the unsupervised learning scheme. The most popular is the one suggested in [34] that builds upon a special type of Boltzmann machines known as the *restricted*

Boltzmann machine (RBM), which will be treated in more detail in the next subsection. Needless to say that this is a new field of research with an intense happening in various application disciplines. New techniques are still being developed, and new experimental evidence is added to the existing one. Thus, the terrain may change as new information concerning these networks becomes available. Our goal is to point out the major ideas and techniques that are currently used. The reader must be flexible and engaged in following new developments in this fast-growing area.

Figure 18.13 presents a block diagram of a deep neural network with three hidden layers. The vector of the input random variables is denoted as \mathbf{x} and those associated with the hidden ones as \mathbf{h}^i , $i = 1, 2, 3$. The vector of the output nodes is denoted as \mathbf{y} . Pre-training evolves in a sequential fashion, starting from the weights connecting the input nodes to the nodes of the first hidden layer. As we will see soon, this is achieved by maximizing the likelihood of the observed samples of the input observations, \mathbf{x} , and treating the variables associated with the first layer as hidden ones. Once the weights corresponding to the first layer have been computed, the respective nodes are allowed to fire an output value and a vector

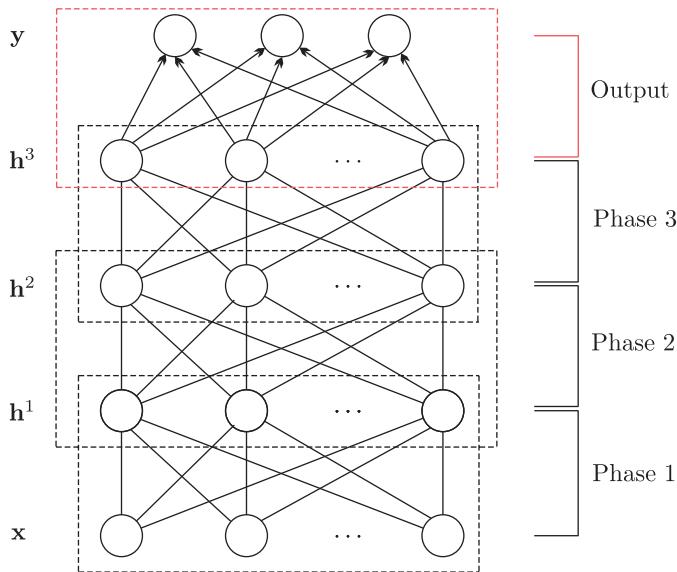


FIGURE 18.13

Block diagram of a deep neural network architecture, with three hidden layers and one output layer. The vector of the input random variables at the input layer is denoted as \mathbf{x} . The vector of the variables associated with the nodes of the i th hidden layer is denoted as \mathbf{h}^i , $i = 1, 2, 3$. The output variables are denoted as \mathbf{y} . At each phase of the pre-training, the weights associated with one hidden layer are computed, one at a time. For the network of the figure, comprising three hidden layers, pre-training consists of three stages of unsupervised learning. Once pre-training of the hidden units has been completed, the weights associated with the output nodes are pre-trained via a supervised learning algorithm. During the final fine-tuning, all the parameters are estimated via a supervised learning rule, such as the backpropagation scheme, using as initial values those obtained during pre-training.

of values, \mathbf{h}^1 , is formed. This is the reason that a generative model for the unsupervised pre-training is adopted (such as the RBM), to be able to generate *in a probabilistic way* outputs at the hidden nodes. These values are in turn used as observations for the pre-training of the next hidden layer, and so on.

Once pre-training has been completed, a supervised learning rule, such as the backpropagation, is then employed to obtain the values of weights leading to the output nodes, as well as to fine tune the weights associated with the hidden layers, using as initial weight values those obtained during the pre-training phase.

Before proceeding into mathematical details, some further comments regarding the adopted approach can be helpful in better understanding the philosophy behind this type of training procedure.

We can interpret each one of the hidden layers as creating a feature vector and our deep architecture as a scheme for learning a *hierarchy of features*. The higher the layer is the higher the abstraction of the representation, associated with the respective feature vector. Using many layers of nodes, we leave the network to decide and generate a hierarchy of features, in an effort to capture the regularities underlying the data. Using deep architectures, one can provide as input to the network a “coding” scheme, which is as close as possible to the raw data, without being necessary for the designer to intervene and generate the features. This is very natural, because in complex tasks, which have to learn and predict nontrivial concepts, it is difficult for a human to know and generate good features that encode efficiently the relevant information, which resides in the data; grasping this information is vital for the generalization power of the model during prediction. Hence, the idea in deep learning is to leave the feature generation task, as much as possible, to the network itself.

It seems that unsupervised learning is a way to discover and unveil information hidden in the data, by learning the underlying regularities and the statistical structure of the data. In this way, pre-training can be thought of as a data-dependent *regularizer* that pushes the unknown parameters to regions where good solutions exist, by exploiting the extra information acquired by the unsupervised learning, see, for example, [19]. It is true to say that, some more formal and theoretically pleasing arguments, which can justify the good generalization performance obtained by such networks, are still to come.

Distributed representations

A notable characteristic of the features generated internally, layer by layer, in a multilayer neural network is that they offer what is known in machine learning as *distributed representation* of the input patterns. Some of the node outputs are 1 and the rest are 0. Interpreting each node as a feature, that provides information with respect to the input patterns, a distributed representation is spread among all these possible features, which are not mutually exclusive. In the antipodal of such a representation would be to have a single neuron firing each time. Moreover, it turns out that such a distributed representation is sparse, because only a few of the neurons are active each time. This is in line with what we believe happens in the human brain, where at each time less than 5% of the neurons, in each layer, fire, and the rest remain inactive.

Sparsity is another welcome characteristic, which is strongly supported by the more general learning theory, for example, [91]. Following information theoretic arguments, it can be shown that to get good generalization performance, the number of bits needed to encode the whole training set should be small with respect to the number of training data. Moreover, sparsity offers the luxury of encoding different examples with different binary codes, as required in many applications.

At the other extreme of representation is the one offered by local methods, where a different model is attached to each region in space and parameters are optimized locally. However, it turns out that

distributed representations can be exponentially more compact, compared to local representations. Take as an example the representation of integers in the interval $[1, 2, \dots, N]$. One way is to use a vector of length N and for each integer to set the respective position equal to 1. However, a more efficient way in terms of the number of bits would be to employ a distributed representation; that is, use a vector of size $\log_2 N$ and encode each integer via ones and zeros positioned to express the number as a sum of powers of two. An early discussion on the benefits of distributed representation in learning tasks can be found in [30]. A more detailed treatment of these issues is provided in [9].

18.8.3 TRAINING RESTRICTED BOLTZMANN MACHINES

A *restricted Boltzmann machine* (RBM) is a special type of the more general class of Boltzmann machines (BMs), which were introduced in Chapter 15, [1, 82]. Figure 18.14 shows the probabilistic graphical model corresponding to an RBM. There are no connections among nodes of the same layer. Moreover, the upper level comprises nodes corresponding to hidden variables and the lower level consists of visible nodes. That is, observations are applied to the nodes of the lower layer only. Following the general definition of a Boltzmann machine, the joint distribution of the involved random variables is of the form,

$$P(v_1, \dots, v_J, h_1, \dots, h_I) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})), \quad (18.58)$$

where we have used different symbols for the J visible (v_j , $j = 1, 2, \dots, J$) and the I hidden variables (h_i , $i = 1, 2, \dots, I$). The *energy* is defined in terms of a set of unknown parameters,⁵ that is,

$$P(v_1, \dots, v_J, h_1, \dots, h_I) = \frac{1}{Z} \exp(-E(\mathbf{v}, \mathbf{h})), \quad (18.59)$$

where b_i and c_j are the bias terms for the hidden and visible nodes, respectively. The normalizing constant is obtained as,

$$Z = \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})). \quad (18.60)$$

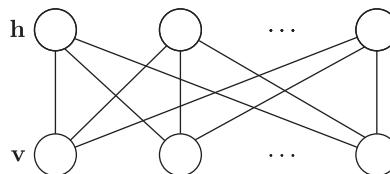


FIGURE 18.14

An RBM is an undirected graphical model with no connections among nodes of the same layer. In the context of the deep networks, the lower level comprises visible nodes and the upper layer consists of hidden nodes only.

⁵ Compared to the notation used in Section 15.4.2 we use a negative sign. This is only to suit better the needs of the section, and it is obviously of no importance for the derivations.

We will focus on discrete variables, hence the involved distributions are probabilities. More specifically, we will focus on variables of a binary nature, that is, $v_j, h_i \in \{0, 1\}$, $j = 1, \dots, J$, $i = 1, \dots, I$. Observe from Eq. (18.59) that, in contrast to a general Boltzmann machine, only products between hidden and visible variables are present in the energy term.

The goal in this section is to derive a scheme for training an RBM; that is, to learn the set of unknown parameters, θ_{ij} , b_i , c_j , which will be collectively denoted as Θ , \mathbf{b} , and \mathbf{c} , respectively. The method to follow is to maximize the log-likelihood, using N observations of the visible variables, denoted as \mathbf{v}_n , $n = 1, 2, \dots, N$, where

$$\mathbf{v}_n := [v_{1n}, \dots, v_{Jn}]^T,$$

is the vector of the corresponding observations at time n . We will say that the visible nodes are *clamped* on the respective observations. Once we establish a training scheme for an RBM, we will see how this mechanism can be embedded into a deep network for pre-training.

The corresponding (average) log-likelihood is given by,

$$\begin{aligned} L(\Theta, \mathbf{b}, \mathbf{c}) &= \frac{1}{N} \sum_{n=1}^N \ln P(\mathbf{v}_n; \Theta, \mathbf{b}, \mathbf{c}) \\ &= \frac{1}{N} \sum_{n=1}^N \ln \left(\frac{1}{Z} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}_n, \mathbf{h}; \Theta, \mathbf{b}, \mathbf{c})) \right) \\ &= \frac{1}{N} \sum_{n=1}^N \ln \left(\sum_{\mathbf{h}} \exp(-E(\mathbf{v}_n, \mathbf{h}; \Theta, \mathbf{b}, \mathbf{c})) \right) \\ &\quad - \ln \sum_{\mathbf{v}} \sum_{\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h})), \end{aligned}$$

where the index n in the energy refers to the respective observations onto which the visible nodes have been clamped, and Θ has explicitly been brought into the notation.

Taking the derivative of $L(\Theta, \mathbf{b}, \mathbf{c})$ with respect to θ_{ij} (similar to the case with respect to b_i and c_j), and applying standard properties of derivatives, it is not difficult to show (Problem 18.10), that

$$\frac{\partial L(\Theta, \mathbf{b}, \mathbf{c})}{\partial \theta_{ij}} = \frac{1}{N} \sum_{n=1}^N \left(\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}_n) h_i v_{jn} \right) - \sum_{\mathbf{v}} \sum_{\mathbf{h}} P(\mathbf{v}, \mathbf{h}) h_i v_j, \quad (18.61)$$

where we have used that,

$$P(\mathbf{h}|\mathbf{v}) = \frac{P(\mathbf{v}, \mathbf{h})}{\sum_{\mathbf{h}'} P(\mathbf{v}, \mathbf{h}')},$$

The gradient in (18.61) involves two terms. The first one can be computed once $P(\mathbf{h}|\mathbf{v})$ is available; we will derive it shortly. Basically, this term is the mean *firing rate* or *correlation* when the RBM is operating in its clamped phase; often we call it the *positive* phase, and the term is denoted as $\langle h_i v_j \rangle^+$. The second term is the corresponding correlation when the RBM is working in its *free running* or *negative* phase and it is denoted as $\langle h_i v_j \rangle^-$. Thus, a gradient ascent scheme for maximizing the log-likelihood will be of the form,

$$\theta_{ij}(\text{new}) = \theta_{ij}(\text{old}) + \mu (\langle h_i v_j \rangle^+ - \langle h_i v_j \rangle^-).$$

Before going any further, let's take a minute to justify why we have named the two phases of operation as positive and negative, respectively. These terms appear in the seminal papers on Boltzmann machines by Hinton and Sejnowski [29, 31]. The first one, corresponding to the clamped condition, can be thought of as a form of a *Hebbian* learning rule. Hebb was a neurobiologist and stated the first ever (to my knowledge) learning rule [27]: “If two neurons on either side of a synapse are activated simultaneously, the strength of this synapse is selectively increased.” Note that this is exactly the effect of the positive phase correlation in the parameter’s update recursion. On the contrary, the effect of the negative phase correlation term is the opposite. Thus, the latter term can be thought of as a *forgetting* or *unlearning* contribution; it can be considered as a control condition of a purely “internal” nature (note that it does not depend on the observations), compared to the “external” information received from the environment (observations).

Computation of the conditional probabilities

From the respective definitions, we get that

$$P(\mathbf{h}|\mathbf{v}) = \frac{1}{Z} \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{P(\mathbf{v})} = \frac{\exp(-E(\mathbf{v}, \mathbf{h}))}{\sum_{\mathbf{h}'} \exp(-E(\mathbf{v}, \mathbf{h}'))}, \quad (18.62)$$

and plugging in the definition of the energy in Eq. (18.59), we obtain

$$\begin{aligned} P(\mathbf{h}|\mathbf{v}) &= \frac{\exp\left(\sum_{i=1}^I \sum_{j=1}^J \theta_{ij} h_i v_j + \sum_{i=1}^I b_i h_i\right) \exp\left(\sum_{j=1}^J c_j v_j\right)}{\sum_{\mathbf{h}'} \exp\left(\sum_{i=1}^I \sum_{j=1}^J \theta_{ij} h'_i v_j + \sum_{i=1}^I b_i h'_i\right) \exp\left(\sum_{j=1}^J c_j v_j\right)} \\ &= \prod_{i=1}^I \frac{\exp\left(\sum_{j=1}^J \theta_{ij} v_j + b_i\right) h_i}{\sum_{h'_i} \left[\exp\left(\sum_{j=1}^J \theta_{ij} v_j + b_i\right) h'_i \right]}. \end{aligned} \quad (18.63)$$

The factorization is a direct consequence of the RBM modeling, where no connections among hidden nodes are present (Problem 18.11). The previous formula readily suggests that,

$$P(h_i|\mathbf{v}) = \frac{\exp\left(\sum_{j=1}^J \theta_{ij} v_j + b_i\right) h_i}{\sum_{h'_i} \left[\exp\left(\sum_{j=1}^J \theta_{ij} v_j + b_i\right) h'_i \right]}, \quad (18.64)$$

which for the binary case becomes

$$P(h_i = 1|\mathbf{v}) = \frac{\exp\left(\sum_{j=1}^J \theta_{ij} v_j + b_i\right)}{1 + \exp\left(\sum_{j=1}^J \theta_{ij} v_j + b_i\right)}, \quad (18.65)$$

which can compactly be written as

$P(h_i = 1|\mathbf{v}) = \text{sigm}\left(\sum_{j=1}^J \theta_{ij} v_j + b_i\right),$

(18.66)

and recalling the definition of the logistic sigmoid function (18.9), we have that

$$\text{sigm}(z) = 1 - \sigma(z).$$

Due to the symmetry involved in the defining equations, it can be similarly shown that,

$$P(v_j = 1|\mathbf{h}) = \text{sigm}\left(\sum_{i=1}^I \theta_{ij} h_i + c_j\right). \quad (18.67)$$

Contrastive divergence

To train the RBM, one has to obtain the positive and negative phase correlations. However, the computation of the latter is intractable. A way to approach it is via Gibbs sampling techniques (see Chapter 14). The fact that we know analytically the conditionals, $P(h_i|\mathbf{v})$ and $P(v_j|\mathbf{h})$, allows us to apply Gibbs sampling by sequentially drawing samples, that is, $\mathbf{h}^{(1)} \sim P(\mathbf{h}|\mathbf{v}^{(1)})$, $\mathbf{v}^{(2)} \sim P(\mathbf{v}|\mathbf{h}^{(1)})$, $\mathbf{h}^{(2)} \sim P(\mathbf{h}|\mathbf{v}^{(2)})$, and so on. However, one has to wait long until the chain converges to a distribution representative of the true one. This is a reason that such networks had not been widely used in practical applications. In [33, 34], the method known as *contrastive divergence* (CD) was introduced. The maximum likelihood loss function was approximated as a difference of two Kullback-Leibler divergences. The end result, from an algorithmic point of view, can be conceived as a stochastic approximation attempt, where expectations are replaced by samples. There is, however, a notable difference: no samples are available for the hidden variables. According to the CD method, these samples are *generated* via Gibbs sampling, starting the chain from the observations available for the visible nodes. The most important feature is that, in practice, only a few iterations of the chain are sufficient.

Following this rationale, a first primitive version of this algorithmic scheme can be cast as:

- Step 1: Start the Gibbs sampler at $\mathbf{v}^{(1)} := \mathbf{v}_n$ and generate samples for the hidden variables, that is,

$$\mathbf{h}^{(1)} \sim P(\mathbf{h}|\mathbf{v}^{(1)}).$$

- Step 2: Use $\mathbf{h}^{(1)}$ to generate samples for the visible nodes,

$$\mathbf{v}^{(2)} \sim P(\mathbf{v}|\mathbf{h}^{(1)}).$$

These are known as *fantasy data*.

- Step 3: Use $\mathbf{v}^{(2)}$ to generate the next set of hidden variables,

$$\mathbf{h}^{(2)} \sim P(\mathbf{h}|\mathbf{v}^{(2)}).$$

The scheme based on these steps is known as CD-1, because only one up-down-up Gibbs sweep is used. If k such steps are employed, the resulting scheme is referred to as CD- k . Once the samples have been generated, the parameter update can be written as

$$\theta_{ij}(n) = \theta_{ij}(n-1) + \mu \left(h_i^{(1)} v_{jn} - h_i^{(2)} v_j^{(2)} \right). \quad (18.68)$$

Note that in the first term in the parenthesis the clamped value of the j th visible node is used; in the second one, we use the sample that is obtained after running the Gibbs sampling on the model itself. It is common to represent the contrastive divergence update rule as

$$\Delta \theta_{ij} \propto \langle h_i v_j \rangle_{\text{data}} - \langle h_i v_j \rangle_{\text{recon}}, \quad (18.69)$$

where the first expectation (over the hidden unit activations) is with respect to the data distribution and the second expectation is with respect to the distribution of the “reconstructed” data, via Gibbs sampling.

A more refined scheme results if the estimates of the gradients are not obtained via a single observation sample, but they are instead averaged over a number of observations. In this vein, the training input examples are divided in a number of disjoint chunks, each one comprising, say, L examples. These blocks of data are also known as *mini-batches*. The previous steps are performed for each observation, but now the update is carried out only once per block of L samples, by averaging out the obtained estimates of the gradient, that is,

$$\theta_{ij}^{(t)} = \theta_{ij}^{(t-1)} + \frac{\mu}{L} \sum_{l=1}^L g_{ij}^{(l)}, \quad i = 1, \dots, I, j = 1, \dots, J, \quad (18.70)$$

where

$$g_{ij}^{(l)} := h_i^{(1)} v_{j(l)} - h_i^{(2)} v_j^{(2)},$$

denotes the gradient approximation associated with the corresponding observation $v_{j(l)}$, $(l) \in \{1, 2, \dots, N\}$, which is currently considered by the algorithm (and gives birth to the associated Gibbs samples). Recursion (18.70) can be written in a more compact form as

$$\Theta^{(t)} = \Theta^{(t-1)} + \frac{\mu}{L} \sum_{l=1}^L G_{ij}^{(l)}, \quad (18.71)$$

where

$$G_{ij}^{(l)} := \mathbf{h}^{(1)} \mathbf{v}_{(l)}^T - \mathbf{h}^{(2)} \mathbf{v}^{(2)T}.$$

Once all blocks have been considered, this corresponds to one epoch of training. The process continues for a number of successive epochs until a convergence criterion is met.

Another version of the scheme results if we replace the obtained samples of the hidden variables with their respective mean values. This, in turn, leads to estimates with lower variance [85]. This is in accordance to what is known as Rao-Blackwellization, where the generated samples are replaced by their expected values. In our current context, where the variables are of a binary nature, it is readily seen that

$$\mathbb{E}[h_i^{(1)}] = P(h_i = 1 | v_{j(l)}) = \text{sigm}\left(\sum_{j=1}^J \theta_{ij}^{(t-1)} v_{j(l)} + b_i^{(t-1)}\right), \quad (18.72)$$

$$\mathbb{E}[h_i^{(2)}] = P(h_i = 1 | v_j^{(2)}) = \text{sigm}\left(\sum_{j=1}^J \theta_{ij}^{(t-1)} v_j^{(2)} + b_i^{(t-1)}\right). \quad (18.73)$$

In this case, the updates become

$$\Theta^{(t)} = \Theta^{(t-1)} + \frac{\mu}{L} \sum_{l=1}^L G_{ij}^{(l)}, \quad (18.74)$$

$$G_{ij}^{(l)} := \mathbb{E}[\mathbf{h}^{(1)}] \mathbf{v}_{(l)}^T - \mathbb{E}[\mathbf{h}^{(2)}] \mathbf{v}^{(2)T}. \quad (18.75)$$

The updates of the bias terms are derived in a similar way (one can also assume that there are fictitious extra nodes of a fixed value +1, and incorporate the bias terms in θ_{ij}), and we get

$$\mathbf{b}^{(t)} = \mathbf{b}^{(t-1)} + \frac{\mu}{L} \sum_{l=1}^L \mathbf{g}_b^{(l)}, \quad (18.76)$$

$$\mathbf{g}_b^{(l)} := \mathbb{E}[\mathbf{h}^{(1)}] - \mathbb{E}[\mathbf{h}^{(2)}], \quad (18.77)$$

and

$$\mathbf{c}^{(t)} = \mathbf{c}^{(t-1)} + \frac{\mu}{L} \sum_{l=1}^L \mathbf{g}_c^{(l)}, \quad (18.78)$$

$$\mathbf{g}_c^{(l)} := \mathbf{v}^{(l)} - \mathbf{v}^{(2)}. \quad (18.79)$$

The resulting scheme, using the expected values version, is summarized in [Algorithm 18.4](#).

Algorithm 18.4 (RBM learning via CD-1 for binary variables).

- Initialization
 - Initialize $\Theta^{(0)}, \mathbf{b}^{(0)}, \mathbf{c}^{(0)}$, randomly.
- **For** each epoch **DO**
 - **For** each block of size L **Do**
 - $G = O, \mathbf{g}_b = \mathbf{0}, \mathbf{g}_c = \mathbf{0}$; set gradients to zero.
 - **For** each \mathbf{v}_n in the block **Do**
 - $\mathbf{h}^{(1)} \sim P(\mathbf{h}|\mathbf{v}_n)$
 - $\mathbf{v}^{(2)} \sim P(\mathbf{v}|\mathbf{h}^{(1)})$
 - $\mathbf{h}^{(2)} \sim P(\mathbf{h}|\mathbf{v}^{(2)})$
 - $G = G + \mathbb{E}[\mathbf{h}^{(1)}]\mathbf{v}_n^T - \mathbb{E}[\mathbf{h}^{(2)}]\mathbf{v}^{(2)}$
 - $\mathbf{g}_b = \mathbf{g}_b + \mathbb{E}[\mathbf{h}^{(1)}] - \mathbb{E}[\mathbf{h}^{(2)}]$
 - $\mathbf{g}_c = \mathbf{g}_c + \mathbf{v}_n - \mathbf{v}^{(2)}$
 - **End for**
 - $\Theta = \Theta + \frac{\mu}{L}G$
 - $\mathbf{b} = \mathbf{b} + \frac{\mu}{L}\mathbf{g}_b$
 - $\mathbf{c} = \mathbf{c} + \frac{\mu}{L}\mathbf{g}_c$
 - **End for**
 - If a convergence criterion is met, Stop.
- **End For**

Remarks 18.5.

- *Persistent contrastive divergence:* To present the contrastive divergence technique, we made a comment related to the stochastic approximation method; the main point is that it is a result of approximating the likelihood via the difference of two KL divergences. However, there is indeed a strong relation of the method with stochastic approximation arguments. As a matter of fact, a version very similar to contrastive divergence was derived in [101], in the context of the general Boltzmann machines, based entirely on stochastic approximation arguments for minimizing the log-likelihood cost function. The main idea is traced back to [62]. The difference with the

contrastive divergence lies in the fact that instead of resetting the chain to the data after each parameter update, the previous state of the chain is used for the next iteration of the algorithm. This initialization is often fairly close to the model distribution, even though the model has changed a bit in the parameter update. The algorithm is known as *persistent contrastive divergence* (PCD) to emphasize that the Markov chain is not reset between parameter updates. It can be shown that this algorithm generates a consistent estimator, even with one Gibbs cycle per iteration.

The PCD algorithm can be used to obtain gradient estimates in an *online* mode of operation or using mini-batches, using only a few training data points for the positive correlation term of each gradient estimate and only a few samples for the negative correlation term. It was demonstrated in [89] that this scheme can lead to enhanced performance, compared to CD.

A treatment of stochastic approximation techniques for minimizing the log-likelihood in the context of RBMs is given in [85]. This bridge paves the way of using the various “tricks” developed for the more general stochastic approximation methods, such as weight decaying or using of momentum terms to smooth out the convergence trajectory in the parameter space.

Moreover, general tools from the stochastic approximation theory, concerning the convergence of such algorithms, can be mobilized.

- Since the advent of the contrastive divergence method, a number of papers have been dedicated to its theoretical analysis. In [15], it was shown that, in general, the fixed points of CD will differ from those of maximum likelihood; however, assuming the data is generated via an RBM, then asymptotically they both share the maximum likelihood solution as a fixed point. Conditions in [100] are derived to guarantee convergence of CD; however, they are difficult to satisfy in practice. An analysis of CD in terms of an expansion of the log-probability is given in [10]. In [43], contrastive divergence is related to the gradient of the log pseudo-likelihood of the model. In [84], the focus of the analysis is on the CD-1 and it is pointed out that this is not related to the gradient of any cost function. Furthermore, it is shown that a regularized CD update has a fixed point for a large class of regularization functions.

18.8.4 TRAINING DEEP FEED-FORWARD NETWORKS

Figure 18.13 illustrates a multilayer perceptron with three hidden layers. As is always the case with any supervised learning task, the kick-off point is a set of training examples, (y_n, \mathbf{x}_n) , $n = 1, 2, \dots, N$. Training a deep multilayer perceptron, employing what we have said before, involves two major phases: (a) pre-training and (b) supervised fine-tuning. Pre-training the weights associated with hidden nodes involves unsupervised learning via the RBM rationale. Assuming K hidden layers, \mathbf{h}^k , $k = 1, 2, \dots, K$, we look at them in pairs, that is, $(\mathbf{h}^{k-1}, \mathbf{h}^k)$, $k = 1, 2, \dots, K$, with $\mathbf{h}^0 := \mathbf{x}$, being the input layer. Each pair will be treated as an RBM, in a hierarchical manner, with the outputs of the previous one becoming the inputs to the next. It can be shown, for example, in [34], that adding a new layer each time increases a variational lower bound on the log-probability of the training data.

Pre-training of the weights leading to the output nodes is performed via a supervised learning algorithm. The last hidden layer together with the output layer are not treated as an RBM, but as a one layer feed-forward network. In other words, the input to this *supervised* learning task are the features formed in the last hidden layer.

Finally, fine-tuning involves retraining in a typical backpropagation algorithm rationale, using the values obtained during pre-training for initialization. This is very important for getting a better

feeling and understanding of how deep learning works. The label information is used in the hidden layers *only* at the fine-tuning stage. During pre-training, the feature values in each layer grasp information related to the input distribution and the underlying regularities. The label information does not participate in the process of discovering the features. Most of this part is left to the unsupervised phase, during pre-training. Note that this type of learning can also work even if some of the data are unlabeled. Unlabeled information is useful, because it provides valuable extra information concerning the input data. As a matter of fact, this is at the heart of semisupervised learning, see, e.g., [88].

The methodology is summarized in [Algorithm 18.5](#).

Algorithm 18.5 (Training deep neural networks).

- Initialization.
 - Initialize randomly all the weights for the hidden nodes, $\Theta^k, \mathbf{b}^k, \mathbf{c}^k, k = 1, 2, \dots, K$.
 - Initialize randomly the weights leading to the output nodes.
 - Set $\mathbf{h}^0(n) := \mathbf{x}_n, n = 1, 2, \dots, N$.

Phase I: Unsupervised Pre-training of Hidden Units

- **For** $k = 1, 2, \dots, K$, **Do**:
 - Treat \mathbf{h}^{k-1} as visible nodes and \mathbf{h}^k as hidden nodes to an RBM.
 - Train the RBM with respect to $\Theta^k, \mathbf{b}^k, \mathbf{c}^k$, via [Algorithm 18.4](#).
 - Use the obtained values of the parameters to generate in the layer, \mathbf{h}^k, N vectors, corresponding to the N observations.
 - Option 1:
 - $\mathbf{h}^k(n) \sim P(\mathbf{h}|\mathbf{h}^{k-1}(n)), n = 1, 2, \dots, N$; Sample from the distribution.
 - Option 2:
 - $\mathbf{h}^k(n) = [P(h_1^n|\mathbf{h}^{k-1}(n), \dots, P(h_{I_k}^n|\mathbf{h}^{k-1}(n))]^T, n = 1, 2, \dots, N$; that is, propagate the respective probabilities. I_k is the number of nodes in the layer.
- **End For**

Phase II: Supervised Pre-training of Output Nodes

- Train the parameters of the pair $(\mathbf{h}^K, \mathbf{y})$, associated with the output layer, via any *supervised* learning algorithm. Treat $(y_n, \mathbf{h}^K(n)), n = 1, 2, \dots, N$, as the training data.

Phase III: Fine-Tuning of All Nodes via Supervised Training

- Use the obtained values for all the parameters as initial values and train the whole network via the backpropagation, using $(y_n, \mathbf{x}_n), n = 1, 2, \dots, N$ as training examples.

Remarks 18.6.

- Training a deep network has a lot of engineering flavor and one needs to acquire some experience by playing with such networks. This was also the case with the “shallow” networks treated in the beginning of the chapter, where some practical hints concerning the training of such networks were summarized in [Section 18.4.1](#). Some of these hints can also be used for deep networks, when using the backpropagation algorithm, during the final fine-tuning phase. For training deep architectures, we also have to deal with some practical “tricks” concerning the unsupervised pre-training. A list of very useful suggestions are summarized in [36].

18.9 DEEP BELIEF NETWORKS

In line with the emphasis given in this chapter so far, we focused our discussion on deep learning on multilayer perceptrons for supervised learning. Our focus was on the information flow in the feed-forward or bottom-up direction. However, this is only part of the whole story. The other part concerns training *generative* models. The goal of such learning tasks is to “teach” the model to generate data. This is basically equivalent with learning probabilistic models that relate a set of variables, which can be observed, with another set of hidden ones. RBMs are just an instance of such models. Moreover, it has to be emphasized that, RBMs can represent any discrete distribution if enough hidden units are used, [21, 55].

In our discussion up to now in this section, we viewed a deep network as a mechanism forming layer-by-layer features of features, that is, more and more abstract representations of the input data. The issue now becomes whether one can start from the last layer, corresponding to the most abstract representation, and follow a *top-down* path with the new goal of generating data. Besides the need in some practical applications, there is an additional reason to look at this reverse direction of information flow.

Some studies suggest that such top-down connections exist in our visual system to generate lower level features of images starting from higher level representations. Such a mechanism can explain the creation of vivid imagery during dreaming, as well as the disambiguating effect on the interpretation of local image regions by providing contextual prior information from previous frames, for example, [53, 54, 60].

A popular way to represent statistical generative models is via the use of probabilistic graphical models, which were treated in [Chapters 15](#) and [16](#). A typical example of a generative model is that of sigmoidal networks, introduced in [Section 15.3.4](#), which belong to the family of parametric Bayesian (belief) networks. A sigmoidal network is illustrated in [Figure 18.15a](#), which depicts a directed acyclic

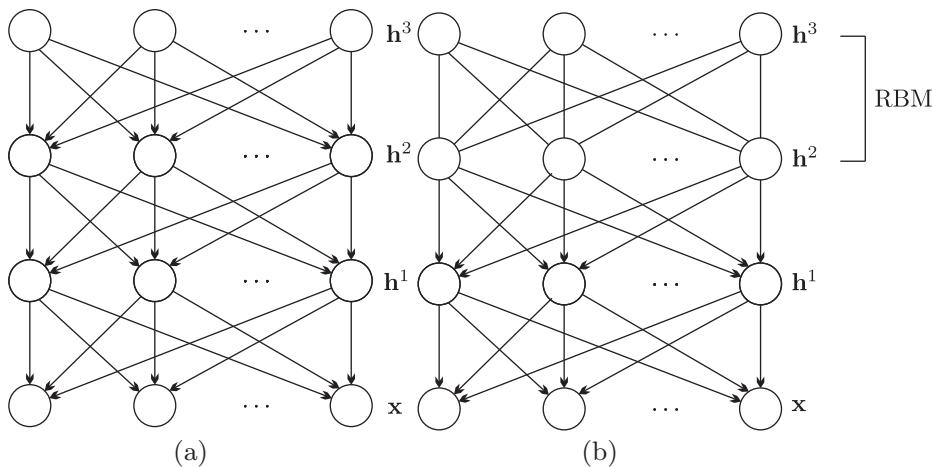


FIGURE 18.15

(a) A graphical model corresponding to a sigmoidal belief (Bayesian) network. (b) A graphical model corresponding to a deep belief network. It is a mixture of directed and undirected edges connecting nodes. The top layer involves undirected connections and it corresponds to an RBM.

graph (Bayesian). Following the theory developed in [Chapter 15](#), the joint probability of the observed (\mathbf{x}) and hidden variables, distributed in K layers, is given by,

$$P(\mathbf{x}, \mathbf{h}^1, \dots, \mathbf{h}^K) = P(\mathbf{x}|\mathbf{h}^1) \left(\prod_{k=1}^{K-1} P(\mathbf{h}^k|\mathbf{h}^{k+1}) \right) P(\mathbf{h}^K),$$

where the conditionals for each one of the I_k nodes of the k th layer are defined as,

$$P(h_i^k | \mathbf{h}^{k+1}) = \sigma \left(\sum_{j=1}^{I_{k+1}} \theta_{ij}^{k+1} h_j^{k+1} \right), \quad k = 1, 2, \dots, K-1, \quad i = 1, 2, \dots, I_k.$$

A variant of the sigmoidal network was proposed in [34], which has become known as *deep belief network*. The difference with a sigmoidal one is that the top two layers comprise an RBM. Thus, it is a mixed type of network consisting of both directed as well as undirected edges. The corresponding graphical model is shown in [Figure 18.15b](#). The respective joint probability of all the involved variables is given by

$$P(\mathbf{x}, \mathbf{h}^1, \dots, \mathbf{h}^K) = P(\mathbf{x}|\mathbf{h}^1) \left(\prod_{k=1}^{K-2} P(\mathbf{h}^k|\mathbf{h}^{k+1}) \right) P(\mathbf{h}^{K-1}, \mathbf{h}^K). \quad (18.80)$$

It is known that learning Bayesian networks of relatively large size is intractable, because of the presence of converging edges (explaining away), see [Section 15.3.3](#). To this end, one has to resort to variational approximation methods to bypass this obstacle, see [Section 16.3](#). However, variational methods often lead to poor performance owing to simplified assumptions.

In [34], it is proposed that we employ the scheme summarized in [Algorithm 18.5](#), Phase 1. In other words, all hidden layers, starting from the input one, are treated as RBMs, and a greedy layer-by-layer pre-training bottom-up philosophy is adopted. We should emphasize that the conditionals, which are recovered by such a scheme can only be thought of as approximations of the true ones. After all, the original graph is a directed one and is not undirected, as the RBM assumption imposes. The only exception lies at the top level, where the RBM assumption is a valid one.

Once the bottom-up pass has been completed, the estimated values of the unknown parameters are used for initializing another fine-tuning training algorithm, in place of the Phase III step of the [Algorithm 18.5](#); however, this time the fine-tuning algorithm is an unsupervised one, as no labels are available. Such a scheme has been developed in [32] for training sigmoidal networks and is known as *wake-sleep* algorithm. The scheme has a variational approximation flavor, and if initialized randomly takes a long time to converge. However, using the values obtained from the pre-training for initialization, the process can significantly be speeded up [37]. The objective behind the wake-sleep scheme is to adjust the weights during the top-down pass, so as to maximize the probability of the network to generate the observed data.

Once training of the weights has been completed, data generation is achieved by the scheme summarized in [Algorithm 18.6](#).

Algorithm 18.6 (Generating samples via a DBN).

- Obtain samples \mathbf{h}^{K-1} , for the nodes at level $K-1$. This can be done via running a Gibbs chain, by alternating samples, $\mathbf{h}^K \sim P(\mathbf{h}|\mathbf{h}^{K-1})$ and $\mathbf{h}^{K-1} \sim P(\mathbf{h}|\mathbf{h}^K)$. This can be carried out as explained in [subsection 18.8.3](#), as the top two layers comprise an RBM. The convergence of the Gibbs chain can be speeded up by initializing the chain with a feature vector formed at the $K-1$ layer by one

of the input patterns; this can be done by following a bottom-up pass to generate features in the hidden layers, as the one used during pre-training.

- **For** $k = K - 2, \dots, 1$, **Do**; Top-down pass.
 - **For** $i = 1, 2, \dots, I_k$, **Do**
 - $h_i^{k-1} \sim P(h_i | h^k)$; Sample for each one of the nodes.
 - **End For**
- **End For**
- $x = h^0$; Generated pattern.

18.10 VARIATIONS ON THE DEEP LEARNING THEME

Besides the basic schemes, which have been presented so far, a number of variants have also been proposed. It is anticipated that in the years to come more and more versions will add on to the existing palette of methods. We will now report the main directions that were currently available at the time this book was published.

18.10.1 GAUSSIAN UNITS

In real-world applications such as speech recognition, data often consist of real-valued features, so the choice of binary visible units can be a modeling restriction. To deal with such types of data, we can use Gaussian visible units instead, that is, linear, real-valued units, with Gaussian noise [21], [58], [87]. If v_j , $j = 1, \dots, J$ and h_i , $i = 1, \dots, I$ are the (Gaussian) visible and (binary) hidden units of the RBM, respectively, the energy function, $E(\mathbf{v}, \mathbf{h})$, of the RBM becomes

$$E(\mathbf{v}, \mathbf{h}) = \sum_{j=1}^J \frac{(v_j - c_j)^2}{2\sigma_j^2} + \sum_{i=1}^I b_i h_i - \sum_{i,j} \theta_{ij} \frac{v_j}{\sigma_j} h_i, \quad (18.81)$$

where c_j , $j = 1, \dots, J$ and b_i , $i = 1, \dots, I$ are the biases of the visible and hidden units, respectively, σ_j , $j = 1, \dots, J$, are the standard deviations of the Gaussian visible units, and θ_{ij} , $i = 1, \dots, I, j = 1, \dots, J$, are the weights connecting the visible and hidden units. The conditional probability, $P(h_i = 1 | \mathbf{v})$, of turning “on” a hidden unit is again the output of the logistic function, as in a standard RBM, that is,

$$P(h_i = 1 | \mathbf{v}) = \text{sigm}\left(b_i + \sum_{j=1}^J \theta_{ij} v_j\right), \quad i = 1, 2, \dots, I. \quad (18.82)$$

However, the conditional pdf, $p(v_j | \mathbf{h})$, of a visible unit now becomes,

$$p(v_j | \mathbf{h}) = \mathcal{N}\left(c_j + \sum_{i=1}^I \theta_{ij} h_i, \sigma_j\right), \quad j = 1, 2, \dots, J. \quad (18.83)$$

Ideally, the contrastive divergence algorithm should be modified, so as to be able to learn the σ_j 's in addition to the c_i 's, b_i 's, and θ_{ij} 's (for a treatment of this topic, the reader is referred to [58]). However, in practice, the estimation of the σ_j 's with the contrastive divergence algorithm is quite an unstable

procedure, and it is therefore preferable to normalize the data to zero mean and unit variance, prior to the RBM training stage. If this normalization step takes place, we no longer need to learn the variances, and we need to make only slight modifications to the contrastive divergence in [Algorithm 18.4](#). More specifically, the sampling step

$$\mathbf{v}^{(2)} \sim P(\mathbf{v} \mid \mathbf{h}^{(1)})$$

is now performed by simply adding Gaussian noise of zero mean and unit variance, $\mathcal{N}(0, 1)$, to the accumulated input of each visible node. Furthermore, the output of each visible unit is no longer binary; however, this does not affect the sampling stages, $\mathbf{h}^{(1)}$ and $\mathbf{h}^{(2)}$ of the contrastive divergence algorithm due to the existence of the sigmoid function at the hidden units.

In practice, if Gaussian visible units are used, the step-size for weights and biases should be kept rather small during the training stage, compared to a standard RBM. For example, step-sizes of the order of 0.001 are not unusual. This is because we want to reduce the risk of the training algorithm to diverge, due to the linear nature of the visible units, which can receive large input values and do not possess a squashing function, which is capable of bounding the output to a predefined range of values.

It is also interesting to note that it is possible to have binary visible and Gaussian hidden units. This can be particularly useful for problems where one does not want to restrict the activation output of the hidden units to fall in the range [0, 1]. For example, this is the case in a deep encoder, used for dimensionality reduction purposes [\[35\]](#).

Finally, it is also possible to have the more general case of Gaussian units for both the visible as well as the hidden layers, although, in practice, such networks are very hard to train [\[58\]](#). Note that in this more general case Eq. [\(18.81\)](#) becomes

$$E(\mathbf{v}, \mathbf{h}) = \sum_{j=1}^J \frac{(v_j - c_j)^2}{2\sigma_j^2} + \sum_{i=1}^I \frac{(h_i - b_i)^2}{2\sigma_i^2} - \sum_{i,j} \theta_{ij} \frac{v_j}{\sigma_j} \frac{h_i}{\sigma_i}. \quad (18.84)$$

18.10.2 STACKED AUTOENCODERS

Instead of building a deep network architecture by hierarchically training layers of RBMs, one can replace RBMs with autoencoders. *Autoencoders* have been proposed in [\[3, 75\]](#) as methods for dimensionality reduction. An autoencoder consists of two parts, the *encoder* and the *decoder*. The output of the encoder is the reduced representation of the input pattern, and it is defined in terms of a vector function,

$$\mathbf{f} : \mathbf{x} \in \mathbb{R}^l \longmapsto \mathbf{h} \in \mathbb{R}^m, \quad (18.85)$$

where,

$$h_i := f_i(\mathbf{x}) = \phi_e(\boldsymbol{\theta}_i^T \mathbf{x} + b_i^e), \quad i = 1, 2, \dots, m, \quad (18.86)$$

with ϕ_e being the activation function; the latter is usually taken to be the logistic sigmoid function, $\phi_e(\cdot) = \sigma(\cdot)$.

The decoder is another function \mathbf{g} ,

$$\mathbf{g} : \mathbf{h} \in \mathbb{R}^m \longmapsto \hat{\mathbf{x}} \in \mathbb{R}^l, \quad (18.87)$$

where

$$\hat{x}_j = g_j(\mathbf{h}) = \phi_d(\boldsymbol{\theta}_j^T \mathbf{h} + b_j^d), \quad j = 1, 2, \dots, l. \quad (18.88)$$

The activation ϕ_d is, usually, taken to be either the identity (linear reconstruction) or the logistic sigmoid one. The task of training is to estimate the parameters,

$$\Theta := [\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_m], \mathbf{b}^e, \quad \Theta' := [\boldsymbol{\theta}'_1, \dots, \boldsymbol{\theta}'_l], \mathbf{b}^d.$$

It is common to assume that $\Theta' = \Theta^T$. The parameters are estimated so the reconstruction error, $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$, over the available input samples are to be minimum in some sense. Usually, the least-squares cost is employed, but other choices are also possible. Regularized versions, involving a norm of the parameters, is also a possibility, for example, [71]. If the activation ϕ_e is chosen to be the identity (linear representation), and $m < l$ (to avoid triviality), the autoencoder is equivalent with the PCA technique [3]. PCA is treated in more detail in [Chapter 19](#).

Another version of autoencoders results if one adds noise to the input [92, 93]. This is a stochastic counterpart, known as the *denoising autoencoder*. For reconstruction, the uncorrupted input is employed. The idea behind this version is that by trying to undo the effect of noise, one captures statistical dependencies between inputs. More specifically, in [92], the corruption process randomly sets some of the inputs (as many as half of them) to zero. Hence, the denoising autoencoder is forced to predict the missing values from the nonmissing ones, for randomly selected subsets of missing patterns.

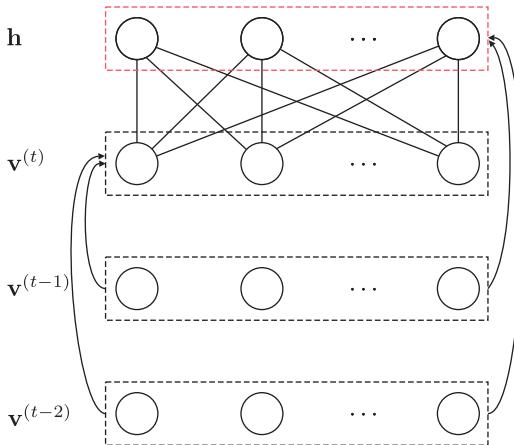
Training a deep multilayer perceptron employing autoencoders consists of the following phases:

- Phase 1: Train the first hidden layer of nodes as an autoencoder; that is, by minimizing an adopted reconstruction error.
- Following the same rationale as for the RBMs, the hidden units' outputs of the autoencoder are used as inputs to feed the layer above. Training is done by treating the two layers as an autoencoder.
- Keep adding as many layers as is required by the depth of the network.
- The output of the last hidden layer is then used as input to the top output layer. The associated parameters are estimated in a supervised manner, using the available labels. Note that this is the first time during pre-training that one uses the label information.
- Employ an algorithm, for example, backpropagation for fine-tuning.

In [35], a different technique for fine tuning is suggested. The hierarchy of autoencoders is unfolded (i.e., both encoder and decoder are used) to reproduce a reconstruction of the input and an algorithm is used to minimize the error.

18.10.3 THE CONDITIONAL RBM

The *conditional restricted Boltzmann machine* (CRBM) [87] is an extension to the standard RBM, capable of modeling temporal dependencies among successive feature vectors (assuming that the training set consists of a set of feature sequences). [Figure 18.16](#) presents the structure of a CRBM, where the hidden layer consists of binary stochastic neurons as is also the case with the standard RBM. However, it can be seen that there exist multiple layers of visible nodes, which, in the case of the figure, correspond to frames $\mathbf{v}^{t-2}, \mathbf{v}^{t-1}, \mathbf{v}^t$. Each visible layer corresponds to the feature vector at the respective time instant and consists of linear (Gaussian) units with zero mean and unity variance.

**FIGURE 18.16**

Structure of a conditional restricted Boltzmann machine.

The visible layers, which correspond to past time instants, are linked with *directional* links to the hidden layer and to the visible layer representing the t th (current) frame. Note that the t th layer of visible nodes is linked to the hidden layer with *undirected* connections, as in a standard RBM. The links (autoregressive weights) among visible nodes model the short-term temporal structure of the sequence of feature vectors, whereas the hidden units model longer (mid-term) characteristics of the feature sequence. As a result, the visible layers of previous time instants introduce a dynamically changing bias to the nodes in \mathbf{v}^t and \mathbf{h} .

To proceed, let θ_{ij} be the undirectional weight connecting v_j^t with h_i , α_{ki}^{t-q} the directional weight connecting v_k^{t-q} with v_i^t , and d_{ij}^{t-q} the directional weight connecting v_j^{t-q} with h_i , where $q = 1, 2, \dots, Q$ and Q is the length of the temporal context. The probability of turning the hidden unit h_i on is computed as follows [87]:

$$P(h_i | \mathbf{v}^t, \mathbf{v}^{t-1}, \dots, \mathbf{v}^{t-Q}) = \text{sigm} \left(b_i + \sum_j \theta_{ij} v_j^t + \sum_{q=1}^Q \sum_j d_{ij}^{t-q} v_j^{t-q} \right), \quad (18.89)$$

where b_i is the bias of the i th hidden unit, and the last summation term is the dynamically changing bias of the i th hidden node, due to the temporal context.

The conditional pdf, of the linear unit v_j^t , is given by

$$p(v_j^t | \mathbf{h}, \mathbf{v}^t, \mathbf{v}^{t-1}, \dots, \mathbf{v}^{t-Q}) = \mathcal{N} \left(c_j + \sum_i \theta_{ij} h_i + \sum_{q=1}^Q \sum_k v_k^{t-q} \alpha_{kj}^{t-q}, 1 \right) \quad (18.90)$$

where c_j is the bias of the j th visible unit and $\mathcal{N}(\mu, 1)$ is the normal distribution with mean μ and unit variance. The last summation term plays the role of the dynamically changing bias of the visible nodes.

Despite the insertion of the aforementioned two types of directed connections, it is still possible to use the contrastive divergence algorithm to update both the undirected (θ_{ij}) and directed connections (α_{ki}^{t-q} and d_{ij}^{t-q}) [87]. The learning rule for θ_{ij} is the same as in the case of an RBM with binary hidden units. Specifically, recalling the notation used in Eq. (18.69), we can write

$$\Delta\theta_{ij} \propto \langle h_i v_j \rangle_{\text{data}} - \langle h_i v_j \rangle_{\text{recon}}, \quad (18.91)$$

where the angular brackets denote expectations with respect to the distributions of the data and reconstructed data, respectively. In this line of thinking, the learning rules for the hidden biases (b_j) and visible biases (c_i), are

$$\Delta b_i \propto \langle h_i \rangle_{\text{data}} - \langle h_i \rangle_{\text{recon}} \quad (18.92)$$

and

$$\Delta c_j \propto v_j - \langle v_j \rangle_{\text{recon}}, \quad (18.93)$$

respectively. The learning rule for the directed connections, d_{ij}^{t-q} , is

$$\Delta d_{ij}^{t-q} \propto v_j^{t-q} (\langle h_i \rangle_{\text{data}} - \langle h_i \rangle_{\text{recon}}), \quad (18.94)$$

Finally, the learning rule for the autoregressive weights, α_{ki}^{t-q} , is

$$\Delta \alpha_{kj}^{t-q} \propto v_k^{t-q} (v_j^t - \langle v_j^t \rangle_{\text{recon}}). \quad (18.95)$$

In the sequel, we introduce the term *temporal pattern* to denote the sequence of feature vectors $\{\mathbf{v}^{t-Q}, \mathbf{v}^{t-Q+1}, \dots, \mathbf{v}^{t-1}, \mathbf{v}^t\}$. After the temporal patterns have been generated from the training set, and before the training stage begins, as it is customary with the contrastive divergence algorithm, they are shuffled and grouped to form mini-batches (usually 100 patterns per mini-batch) to minimize the risk that the training algorithm is trapped in a local minimum of the cost function. The step-size for Eqs. (18.91)–(18.95) needs to be set equal to a small value (e.g., 0.0001). This small value is important to achieve convergence. All weights and biases are initialized with small values using a Gaussian generator.

Remarks 18.7.

- At the time this book is being compiled there is much research activity on the deep learning topic, and it is hard to think of an application area of machine learning in which deep learning has not been applied. I will provide just a few samples of papers, and there is no claim that this covers the whole happening.

A very successful application of deep networks has been reported in the area of speech recognition, and significant performance improvements have been reported, compared to previously available state-of-the art methods, see, for example, [38]. An application concerning speech-music discrimination is given in [66]. In [94], a visual tracking application is considered. An application on object recognition is reported in [86]. In [63], an application on context-based music recommendation is discussed. The case of large-scale image classification is considered in [81]. These are just a few samples of diverse areas in which deep learning has been applied.

Some more recent review articles are [11, 18].

18.11 CASE STUDY: A DEEP NETWORK FOR OPTICAL CHARACTER RECOGNITION

The current example demonstrates how a deep neural network can be adopted to classify printed characters. Such classifiers constitute an integral part of what is known as an optical character recognition (OCR) system. For pedagogical purposes, and to keep the system simple, we focus on a four-class scenario (the extension to more classes is straightforward). The characters (classes) that are involved are the Greek letters α , ν , σ , and τ , extracted from old historical documents.

Each one of the classes comprises a number of *binarized* images. Each binary image is the result of a segmentation and binarization procedure from scanned documents and all binary images have been reshaped to the same dimensions, that is, 28×28 pixels. This specific dimension has been chosen to comply with the format of the MNIST⁶ data set of handwritten digits. Note that, as is common with real-world problems, due to segmentation and binarization errors, several images contain noisy or partially complete characters.

In our example, the class volumes are 1735, 1850, 2391, and 2264 images for classes α , ν , σ , and τ , respectively. Examples of these characters can be seen in Figure 18.17. The complete data set can be downloaded from the companion site of this book.

Each binary image is converted to a binary feature vector by scanning it row-wise and concatenating the rows to form a $28 \times 28 = 784$ -dimensional binary representation. In the sequel, 80% of the resulting patterns, per class, are randomly chosen to form the training set and the remaining patterns serve testing purposes. The class labels are represented by 4-digit binary codewords. For example, the first class (letter α) is assigned the binary code (1000), the second class is assigned the codeword (0100), and so on.

Because of the binary nature of the patterns, the use of RBMs with binary stochastic units as the building blocks of a deep network is a natural choice. In our case, the adopted deep architecture follows closely the block diagram of Figure 18.13, and consists of five layers in total: an input layer, \mathbf{x} , of 784 binary visible units, three layers, namely \mathbf{h}^1 , \mathbf{h}^2 , and \mathbf{h}^3 , of hidden binary units (consisting of 500, 500, and 2000 nodes respectively) and, finally, an output layer, \mathbf{y} , of four softmax units, which provide the posterior probability estimates of the patterns for each one of the classes.

Our main goal is to pre-train the weights of this network, via the contrastive divergence (CD) algorithm and use the resulting weight values to initialize the backpropagation algorithm; the latter will eventually provide a fine-tuning of the network's weights. As has been explained in the text, we proceed in a layer-wise mode. We are going to repeat some of the comments made before: “repetitio est mater studiorum.”⁷



FIGURE 18.17

Examples of the letters α , ν , σ , and τ of the data set under study.

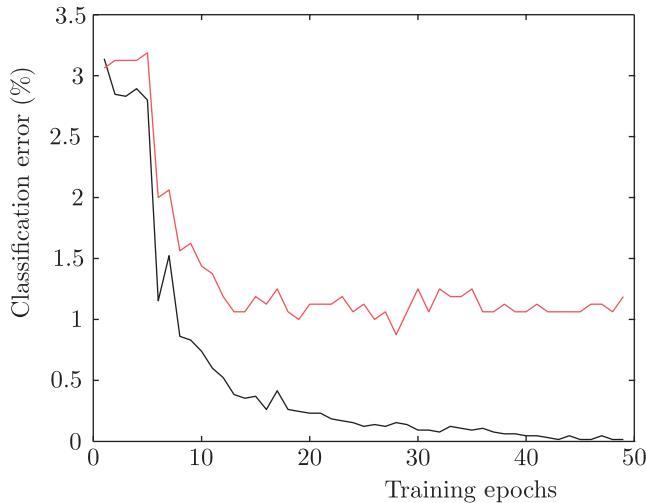
⁶ <http://yann.lecun.com/exdb/mnist/>.

⁷ Repetition is mother of study, in Latin.

- We treat nodes $(\mathbf{x}, \mathbf{h}^1)$ as an RBM and use the CD Algorithm in (18.4) for 50 epochs to compute the weights θ_{ij}^1 , $i = 1, \dots, 500$, $j = 1, \dots, 784$, and biases, b_i^1 , $i = 1, \dots, 500$, of the hidden nodes, and c_j , $j = 1, 2, \dots, 784$.
- After the first RBM has been trained, we compute the activation outputs of the nodes in layer \mathbf{h}^1 , for all the patterns in the training set and use the respective activation probabilities as the visible input data of the second RBM, $(\mathbf{h}^1, \mathbf{h}^2)$. As an alternative, we could binarize the activation outputs of the nodes in \mathbf{h}^1 to use binary data as inputs in the second RBM; however, in practice, using binarization tends to yield inferior performance. After the second RBM has been trained, the values of weights θ_{ij}^2 , $i = 1, \dots, 500$, $j = 1, \dots, 500$ and hidden biases, b_i^2 , become available.
- We proceed in a manner similar to the third RBM, $(\mathbf{h}^2, \mathbf{h}^3)$, whose weights and hidden node biases are denoted as θ_{ij}^3 , $i = 1, \dots, 2000$, $j = 1, \dots, 500$, and b_i^3 , $i = 1, \dots, 2000$, respectively. This time, the activation outputs of the nodes in \mathbf{h}^2 become the visible input data of the third RBM.
- In the previous stages, three RBMs were trained in total for the first four layers of the network $(\mathbf{x}, \mathbf{h}^i, i = 1, \dots, 3)$. In all three training stages, the respective training procedure was unsupervised in the sense that we did not use the information regarding the class labels. The class labels are used for the first time for the supervised training of the weights, θ_{ij}^4 , $i = 1, \dots, 4$, $j = 1, \dots, 2000$, connecting layer \mathbf{h}^3 with the softmax nodes (Eq. 18.41), associated with the output nodes, \mathbf{y} . Specifically, we use the backpropagation algorithm to pre-train the θ_{ij}^4 's, using the activation outputs (\mathbf{h}^3) of the third RBM for each one of the training patterns, as the input vectors and the codeword of the respective class label as the desired output. Although this type of backpropagation procedure lasts for only a few epochs (10 epochs in our experiment), it helps the initialization of the weights in θ^4 , for the backpropagation in the final fine-tuning stage, which will follow; otherwise, initialization should be performed randomly.
- After θ^4 has been pre-trained, a standard backpropagation algorithm that minimizes the cross-entropy cost function (Section 18.4.3) is employed, using fifty epochs to fine-tune all network weights and biases.

During the testing stage, each unknown pattern is “clamped” on the visible nodes of the input layer, \mathbf{x} , and the network operates in a feed-forward mode to propagate the results until the output layer, \mathbf{y} , has been reached. During this feed-forward operation, the nodes of the hidden layers propagate activation outputs, that is, the probabilities at the output of their logistic functions. Also, note that each output (softmax) node, y_i , $i = 1, \dots, 4$, emits normalized values in the range $[0, 1]$ and $\sum_{i=1}^4 y_i = 1$; this allows the interpretation of the corresponding values as posterior probabilities. For each input pattern, the softmax node corresponding to the maximum value is chosen as the winner, and the pattern is assigned to the respective class. For example, if node y_2 wins, the pattern that was “clamped” in the input layer is assigned to class 2 (letter v).

Figure 18.18 presents the training and testing error curves at the end of each training epoch. Note that due to the small number of classes and network size, the resulting errors become really small after just a few epochs. In this case, the errors are mainly due to seriously distorted characters. Furthermore, observe that the training error (as a general trend) keeps decreasing, while the test error reaches a minimum level, which can be interpreted as an indication that, in this case, we have avoided overfitting. Note, however, that this does not mean that deep networks are free from overfitting problems, in general;

**FIGURE 18.18**

Training error (gray) and testing error (red) versus number of epochs for the data set of case study (Section 18.11).

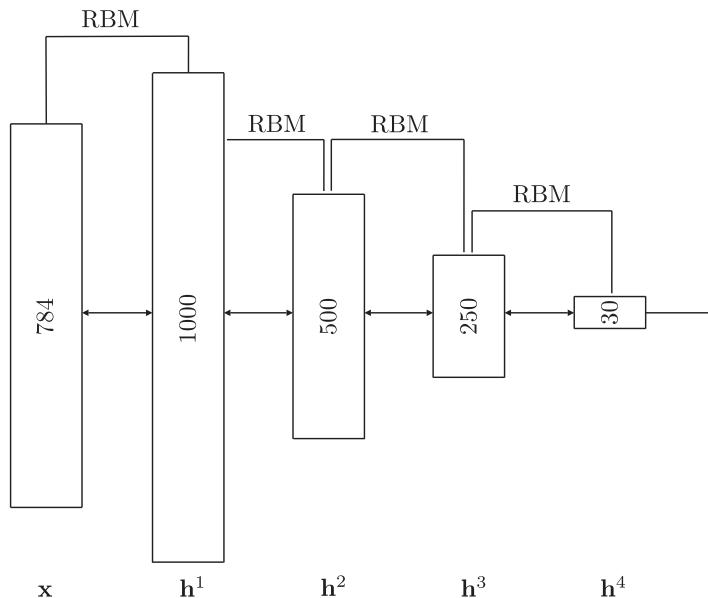
see, for example [83], and the references therein. Observe that the probability of error, after convergence, settles close to 1%. This experimental setup can be readily extended to cover more classes, such as, all the letters of the Greek or Latin alphabets.

18.12 CASE STUDY: A DEEP AUTOENCODER

In Section 18.10, we gave the definition of an autoencoder and discussed the use of autoencoders as building blocks for designing deep networks, as an alternative to using RBMs. Now we turn our attention to the use of RBS in designing deep autoencoders for dimensionality reduction and data compression. The idea was first proposed in [35].

The goal of the encoder is to gradually reduce the dimensionality of the input vectors, which will be achieved by using a multilayer neural network, where the hidden layers decrease in size [35]. We demonstrate the method via an example, using the database of the Greek letters discussed before and following the same procedure concerning the partition in training and test data.

Figure 18.19 shows the block diagram of the encoder. It comprises four hidden layers, \mathbf{h}^i , $i = 1, \dots, 4$, with 1000, 500, 250, and 30 hidden nodes, respectively. The first three hidden layers consist of binary units, whereas the last layer consists of linear (Gaussian) units. We then proceed by pre-training the weights connecting every pair of successive layers using the contrastive divergence algorithm (for 20 epochs), starting from $(\mathbf{x}, \mathbf{h}^1)$ and proceeding with $(\mathbf{h}^1, \mathbf{h}^2)$, and so on. This is in

**FIGURE 18.19**

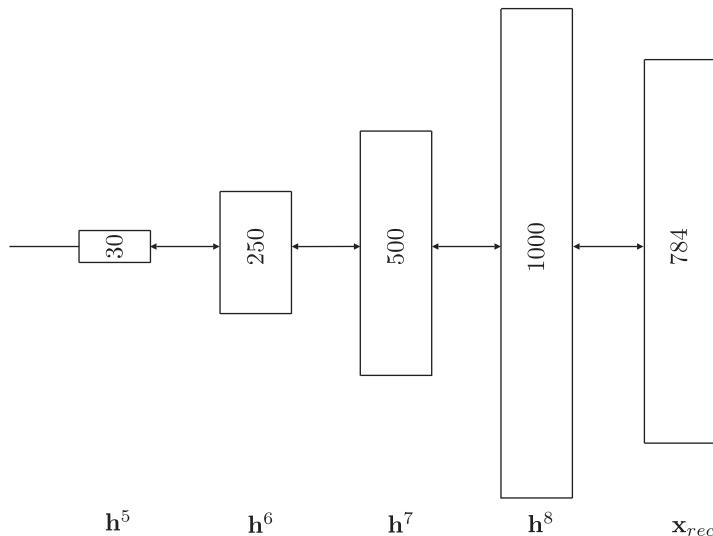
The block diagram for the encoder.

line with what we discussed so far for training deep networks. For the RBM training stage, the whole training data set was used and divided into mini-batches (consisting of 100 patterns), as is common practice.

The decoder is the reverse structure, that is, its input layer receives the 30-dimensional representation at the output of \mathbf{h}^4 and consists of four hidden layers of increasing size, whose dimensions reflect exactly the hidden layers of the encoder, plus an output layer. This is shown in Figure 18.20. It is important to note that the weights of the decoder are not pre-initialized separately; we employ the transpose of the respective weights of the encoder. For example, the weights connecting \mathbf{h}^5 with \mathbf{h}^6 are initialized with $\Theta_{h_3h_4}^T$, where $\Theta_{h_3h_4}$ are the weights connecting layers \mathbf{h}^3 and \mathbf{h}^4 of the encoder. The layer denoted as \mathbf{x}_{rec} is the output layer, which provides the reconstructed version of the input.

After all the weights have been initialized as previously described, the whole encoder-decoder network is treated as a multilayer feed-forward network and the weights are fine-tuned via the backpropagation algorithm (for 200 epochs); for each input pattern, the desired output is the pattern itself. In this way, the backpropagation algorithm tries to minimize the reconstruction error. During the backpropagation training procedure, ten mini-batches are grouped together to form a larger batch, and the weights are updated at the end of the processing of each one of these batches. This is a recipe that has proven to provide better convergence in practice.

Figure 18.21 presents some of the patterns of the training set along with their reconstructions before the fine-tuning stage (backpropagation training). Similarly, Figure 18.22 presents the reconstruction results for the same patterns after the fine-tuning stage has been completed. It can be

**FIGURE 18.20**

The block diagram for the decoder.

**FIGURE 18.21**

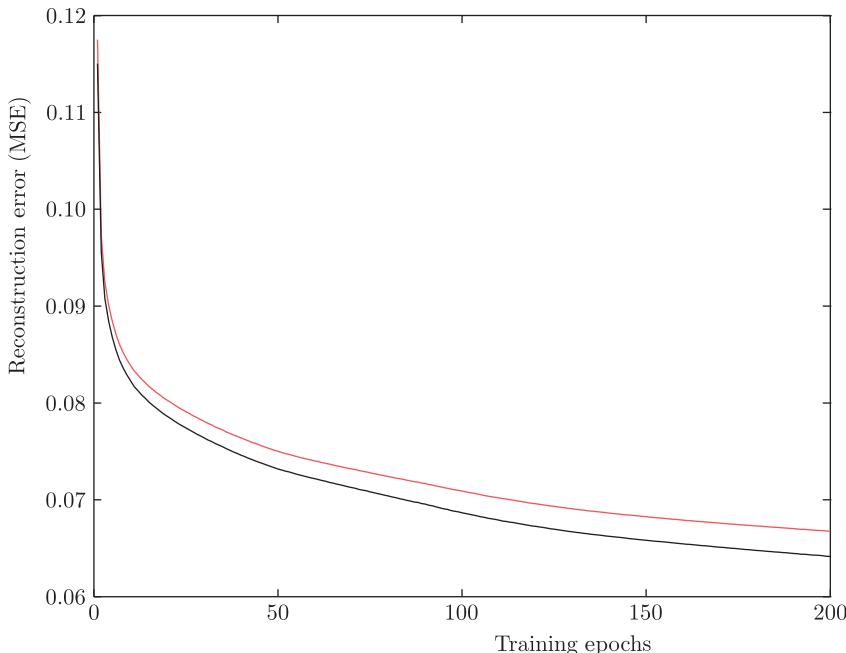
Input patterns and respective reconstructions. The top row shows the original patterns. The bottom row shows the corresponding reconstructed patterns, prior to the application of the backpropagation algorithm for fine-tuning.

**FIGURE 18.22**

Input patterns and respective reconstructions. The top row shows the original patterns. The bottom row shows the corresponding reconstructed patterns after the fine-tuning stage.

readily observed that the application of the backpropagation algorithm yields improved (less noisy) reconstructions.

Finally, [Figure 18.23](#) presents the mean-square reconstruction error (MSE) over all pixels of the images of the training set (black curve) and the testing set (red curve). The MSE is computed during the fine-tuning stage in the beginning of each epoch.

**FIGURE 18.23**

Mean-square error during the fine-tuning stage, using the backpropagation algorithm for the case study in [Section 18.12](#).

18.13 EXAMPLE: GENERATING DATA VIA A DBN

The current example demonstrates the potential of a deep belief network (DBN) to generate data. Our example evolves around the previously introduced data set of the Greek letters α , ν , o and τ .

The proposed DBN follows the architecture of [Figure 18.15b](#), where \mathbf{h}^1 , \mathbf{h}^2 and \mathbf{h}^3 contain 500, 500, and 2000 nodes respectively.

To generate the samples, we follow the steps of [Algorithm 18.6](#). To speed up data generation, each time, we feed a pattern of the data set to the input layer and we propagate the results until layer \mathbf{h}^2 . The activation probabilities of this layer serve to initialize an alternating Gibbs sampling procedure that runs for 5000 iterations. After the sampling chain has been completed, we perform a single down-pass to generate the data at the input layer.

[Figure 18.24](#) presents the data generation results (even rows), along with the patterns that were used to initialize the procedure each time (odd rows). For the sake of clarity of presentation, the generated data are the activation outputs of the visible layer, that is, we do not binarize the final data generation step.

**FIGURE 18.24**

Generated data via a DBN. Odd rows show the data used in the input and even rows show the corresponding data generated by the network.

PROBLEMS

- 18.1** Prove that the perceptron algorithm, in its pattern-by-pattern mode of operation, converges in a finite number of iteration steps. Assume that $\theta^{(0)} = \mathbf{0}$.

Hint. Note that because classes are assumed to be linearly separable, there is a normalized hyperplane, θ_* , and a $\gamma > 0$, so that

$$\gamma \leq y_n \theta_*^T \mathbf{x}_n, \quad n = 1, 2, \dots, N,$$

where, y_n is the respective label, being +1 for ω_1 and -1 for ω_2 . By the term *normalized hyperplane*, we mean that,

$$\theta_*^T = [\hat{\theta}_*, \theta_{0*}]^T, \text{ with } \|\hat{\theta}_*\| = 1.$$

In this case, $y_n \theta_*^T \mathbf{x}_n$ is the distance of \mathbf{x}_n from the hyperplane θ_* ([61]).

- 18.2** The derivative of the sigmoid functions has been computed in [Problem 7.6](#). Compute the derivative of the hyperbolic tangent function and show that it is equal to,

$$f'(z) = ac(1 - f^2(z)).$$

- 18.3** Show that the effect of the momentum term in the gradient descent backpropagation scheme is to effectively increase the learning convergence rate of the algorithm.

Hint. Assume that the gradient is approximately constant over I successive iterations.

- 18.4** Show that if (a) the activation function is the hyperbolic tangent (b) the input variables are normalized to zero mean and unit variance, then to guarantee that all the outputs of the neurons are zero mean and unit variance, the weights must be drawn from a distribution of zero mean and standard deviation equal to

$$\sigma = m^{-1/2},$$

where m is the number of synaptic weights associated with the corresponding neuron.

Hint. For simplicity, consider the bias to be zero, and also that the inputs to each neuron are mutually uncorrelated.

18.5 Consider the sum of error squares cost function

$$J = \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^{k_L} (\hat{y}_{nm} - y_{nm})^2. \quad (18.96)$$

Compute the elements of the Hessian matrix

$$\frac{\partial^2 J}{\partial \theta_{kj}^r \partial \theta_{k'j'}^{r'}}. \quad (18.97)$$

Near the optimum, show that the second order derivatives can be approximated by

$$\frac{\partial^2 J}{\partial \theta_{kj}^r \partial \theta_{k'j'}^{r'}} = \sum_{n=1}^N \sum_{m=1}^{k_L} \frac{\partial \hat{y}_{nm}}{\partial \theta_{kj}^r} \frac{\partial \hat{y}_{nm}}{\partial \theta_{k'j'}^{r'}}. \quad (18.98)$$

In other words, the second order derivatives can be approximated as products of the first order derivatives. The derivatives can be computed by following similar arguments as the gradient descent backpropagation scheme [25].

18.6 It is common when computing the Hessian matrix to assume that it is diagonal. Show that under this assumption, the quantities

$$\frac{\partial^2 E}{\partial (\theta_{kj})^2},$$

where

$$E = \sum_{m=1}^{k_L} (f(z_m^L) - y_m)^2,$$

propagates backward according to the following,

- $\frac{\partial^2 E}{\partial (\theta_{kj}^r)^2} = \frac{\partial^2 E}{\partial (z_j^r)^2} (y_k^{r-1})^2.$
- $\frac{\partial^2 E}{\partial (z_L^r)^2} = f''(z_j^L) e_j + (f'(z_j^L))^2.$
- $\frac{\partial^2 E}{\partial (z_k^{r-1})^2} = (f'(z_j^{r-1}))^2 \sum_k \frac{\partial^2 E}{\partial (z_k^r)^2} (\theta_{kj}^r)^2 + f''(z_j^{r-1}) \sum_{k=1}^{k_r} \theta_{kl}^r \delta_k^r.$

18.7 Show that the cross-entropy loss function depends on the relative output errors.

18.8 Show that if the activation function is the logistic sigmoid and the relative entropy cost function is used, then δ_{nj}^L in Eq. (18.24) becomes,

$$\delta_{nj}^L = a(\hat{y}_{nj} - 1)y_{nj}.$$

18.9 As in the previous problem, use the relative entropy cost function and the softmax activation function. Then show that,

$$\delta_{nj}^L = \hat{y}_{nj} - y_{nj}.$$

18.10 Derive the gradient of the log-likelihood in Eq. (18.61).

- 18.11** Derive the factorization of the conditional probability in Eq. (18.63).
- 18.12** How are Eqs. (18.81) to (18.83) and the contrastive divergence algorithm modified for the case of an RBM with binary visible and Gaussian hidden nodes?

MATLAB Exercises

- 18.13** Consider a two-dimensional class problem that involves two classes ω_1 (+1) and ω_2 (-1). Each one of them is modeled by a mixture of equiprobable Gaussian distributions. Specifically, the means of the Gaussians associated with ω_1 are $[-5, 5]^T$ and $[5, -5]^T$, while the means of the Gaussians associated with ω_2 are $[-5, -5]^T$, $[0, 0]^T$ and $[5, 5]^T$. The covariances of all Gaussians are $\sigma^2 I$, where $\sigma^2 = 1$.
- Generate and plot a data set X_1 (training set) containing 100 points from ω_1 (50 points from each associated Gaussian) and 150 points from ω_2 (again 50 points from each associated Gaussian). In the same way, generate an additional set X_2 (test set).
 - Based on X_1 , train a two-layer neural network with two nodes in the hidden layer having the hyperbolic tangent as activation function and a single output node with linear activation function⁸, using the standard backpropagation algorithm for 9000 iterations and step-size equal to 0.01. Compute the training and the test errors, based on X_1 and X_2 , respectively. Also, plot the test points as well as the decision lines formed by the network. Finally, plot the training error versus the number of iterations.
 - Repeat step (ii) for step-size equal to 0.0001 and comment on the results.
 - Repeat step (ii) for $k = 1, 4, 20$ hidden layer nodes and comment on the results.
- Hint.* Use different seeds in the *rand* MATLAB function for the train and the test sets. To train the neural networks, use the *newff* MATLAB function. To plot the decision region performed by a neural network, first determine the boundaries of the region where the data live (for each dimension determine the minimum and the maximum values of the data points), then apply a rectangular grid on this region and for each point in the grid compute the output of the network. Then draw this point with different colors according to the class it is assigned (use e.g., the “magenta” and the “cyan” colors).
- 18.14** Consider the classification problem of the previous exercise, as well as the same data sets X_1 and X_2 . Consider a two-layer feed-forward neural network as the one in (ii) and train it using the adaptive backpropagation algorithm with initial step-size equal to 0.0001 and $r_i = 1.05$, $r_d = 0.7$, $c = 1.04$, for 6000 iterations. Compute the training and the test errors, based on X_1 and X_2 , respectively and plot the error during training against the number of iterations. Compare the results with those obtained from the previous exercise (ii).
- 18.15** Repeat the previous exercise for the case where the covariance matrix for the Gaussians is $6I$, for 2, 20 and 50 hidden layer nodes, compute the training and the test errors in each case and draw the corresponding decision regions. Draw your conclusions.
- 18.16** Develop a MATLAB program which implements the experiment of the case study in Section 18.11, skipping the RBM pre-training stage. You will first need to download the OCR data set from the companion website of this book. Your program will accept as input the

⁸ The number of input nodes equals to the dimensionality of the feature space, while the number of output nodes is equal to the number of classes minus one.

number of nodes of each layer of the network. Call MATLAB's backpropagation function to initialize the network weights with random numbers and train directly the network as a whole. During the training stage, plot the training error in the beginning of each training epoch. Do you observe any differences regarding the generated error curve compared with the one in [Section 18.11](#)? Provide a justification of your answer.

- 18.17** Download the OCR data set from the companion website of this book. Then, develop a MATLAB function that receives as input the path to the folder containing the test data set and produces a new data set by corrupting each binary image with noise as follows: 5% of the pixels are randomly chosen from each image and their values are altered, that is, a 0 becomes 1 and vice versa. After the corrupted data set has been generated and stored to a new folder, create a function, such as, `deepClassifier.m`, that feeds it to the trained network of [Section 18.11](#) and computes the resulting test error. The weights of the trained network are available at the `OCRTrained1.mat` file. Repeat the error computation by increasing the noise intensity in a stepwise mode, that is, by 1% each time. How is the performance of the deep network affected?
- 18.18** Repeat the experiment in [Exercise 18.17](#), using binary outputs for the hidden nodes in all stages, instead of activation probabilities. This holds for the pre-training stages and the feed-forward classification procedure. Do you observe any performance deterioration?
- 18.19** Develop the MATLAB code and repeat the autoencoder experiment in [18.12](#). Corrupt the data set (both training and test sets) with noise by randomly altering the values of 5% of the pixels of each image and repeat the training procedure. Plot the reconstructed input and MSE curves and comment on the results.

REFERENCES

- [1] D. Ackle, G.E. Hinton, T. Sejnowski, A learning algorithm for Boltzmann machines, *Cognit. Sci.* 9 (1985) 147-169.
- [2] T. Adali, X. Liu, K. Sonmez, Conditional distribution learning with neural networks and its application to channel equalization, *IEEE Trans. Signal Process.* 45 (4) (1997) 1051-1064.
- [3] P. Baldi, K. Hornik, Neural networks and principal component analysis: learning from examples, without local minima, *Neural Netw.* 2 (1989) 53-58.
- [4] E. Barnard, Optimization for training neural networks, *IEEE Trans. Neural Netw.* 3 (2) (1992) 232-240.
- [5] R.A. Barron, Universal approximation bounds for superposition of a sigmoidal function, *IEEE Trans. Inform. Theory* 39 (3) (1993) 930-945.
- [6] R. Battiti, First and second order methods for learning: Between steepest descent and Newton's methods, *Neural Comput.* 4 (1992) 141-166.
- [7] Y. Bengio, O. Delalleau, N. Le Roux, The curse of highly variable functions for local kernel machines, in: Y. Weiss, B. Schölkopf, J. Platt (Eds.), *Advances in Neural Information Processing Systems (NIPS)*, vol. 18, MIT Press, Cambridge, MA, 2006, pp. 107-114.
- [8] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in: B. Schölkopf, J. Platt, T. Hofmann (Eds.), *Advances in Neural Information Processing Systems (NIPS)*, vol. 19, MIT Press, Cambridge, MA, 2007, pp. 153-161.
- [9] Y. Bengio, Learning deep architectures for AI, *Found. Trends Mach. Learn.* 2 (1) (2009) 1-127, DOI: 10.1561/2200000006.

- [10] Y. Bengio, O. Delalleau, Justifying and generalizing contrastive divergence, *Neural Comput.* 21 (6) (2009) 1601-1621.
- [11] Y. Bengio, A. Courville, P. Vincent, Unsupervised feature learning and deep learning: a review and new perspectives, 2014, arXiv:1206.5538v3 [cs.LG] 23 April 2014.
- [12] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.
- [13] J.S. Bridle, Training stochastic model recognition algorithms as networks can lead to maximum information estimation parameters, in: D.S. Touretzky, et al. (Eds.), *Neural Information Processing Systems, NIPS*, vol. 2, Morgan Kaufmann, San Francisco, CA, 1990, pp. 211-217.
- [14] A. Bryson, W. Denham, S. Dreyfus, Optimal programming problems with inequality constraints I: Necessary conditions for extremal solutions, *J. Am. Inst. Aeronaut. Astronaut.* 1 (1963) 25-44.
- [15] M. Carreira-Perpinan, G.E. Hinton, On contrastive divergence learning, in: *Proceedings 10th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2005, pp. 59-66.
- [16] A. Cichoki, R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, John Wiley, New York, 1993.
- [17] G. Cybenko, Approximation by superpositions of a sigmoidal function, *Math. Control Signals Syst.* 2 (1989) 304-314.
- [18] L. Deng, Y. Dong, Deep Learning: Methods and Applications, vol. 7(3-4), 2014.
- [19] D. Erhan, P.A. Manzagol, Y. Bengio, S. Bengio, P. Vincent, The difficulty of training deep architectures and the effect of unsupervised pretraining, in: *Proceedings of The Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS09)*, 2009, pp. 153-160.
- [20] S.E. Fahlman, Faster learning variations on back-propagation: an empirical study, in: *Proceedings Connectionist Models Summer School*, Morgan Kaufmann, San Francisco, CA, 1988, pp. 38-51.
- [21] Y. Freund, D. Haussler, Unsupervised learning of distributions of binary vectors using two layer networks, Technical Report, UCSC-CRL-94-25, 1994.
- [22] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biol. Cybern.* 36 (1980) 193-202.
- [23] K. Funashashi, On the approximation realization of continuous mappings by neural networks, *Neural Netw.* 2 (3) (1989) 183-192.
- [24] M. Hagiwara, Theoretical derivation of momentum term in backpropagation, in: *International Joint Conference on Neural Networks*, Baltimore, vol. I, 1991, pp. 682-686.
- [25] B. Hassibi, D.G. Stork, G.J. Wolff, Optimal brain surgeon and general network pruning, in: *Proceedings IEEE Conference on Neural Networks*, vol. 1, 1993, pp. 293-299.
- [26] S. Haykin, *Neural Networks*, second ed., Prentice Hall, Upper Saddle River, NJ, 1999.
- [27] D.O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, Wiley, New York, 1949.
- [28] J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, MA, 1991.
- [29] G.E. Hinton, T.J. Sejnowski, Optimal perceptual inference, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Washington, DC, June, 1983.
- [30] G.E. Hinton, Learning distributed representations of concepts, in: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, Lawrence Erlbaum, Hillsdale, 1986, pp. 1-12.
- [31] G.E. Hinton, T.J. Sejnowski, Learning and relearning in Boltzmann machines, in: D.E. Rumelhart, J.L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, MIT Press, Cambridge, MA, 1986, pp. 282-317.
- [32] G.E. Hinton, P. Dayan, B.J. Frey, R.M. Neal, The wake-sleep algorithm for unsupervised neural networks, *Science* 268 (1995) 1558-1161.
- [33] G.E. Hinton, Training products of experts by minimizing contrastive divergence, *Neural Comput.* 14 (2002) 1771-1800.

- [34] G.E. Hinton, S. Osindero, Y. Teh, A fast learning algorithm for deep belief nets, *Neural Comput.* 18 (2006) 1527-1554.
- [35] G.E. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks, *Science* 313 (2006) 504-507.
- [36] G. Hinton, A Practical Guide to Training Restricted Boltzmann Machines, Technical Report, UTML TR 2010-003, University of Toronto, 2010, <http://learning.cs.toronto.edu>.
- [37] G.E. Hinton, Learning multiple layers of representation, *Trends Cognit. Sci.* 11 (10) (2010) 428-434.
- [38] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, B. Kingsbury, Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups, *IEEE Signal Process. Mag.* 29 (6) (2012) 82-97.
- [39] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural Netw.* 2 (5) (1989) 359-366.
- [40] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, *Neurocomputing* 70 (2006) 489-501.
- [41] G.B. Huang, L. Chen, C.K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, *IEEE Trans. Neural Netw.* 17 (4) (2006) 879-892.
- [42] D.H. Hubel, T.N. Wiesel, Receptive fields, binocular interaction, and functional architecture in the cats visual cortex, *J. Physiol.* 160 (1962) 106-154.
- [43] A. Hyvarinen, Connections between score matching, contrastive divergence, and pseudolikelihood for continuous-valued variables, *IEEE Trans. Neural Netw.* 18 (5) (2007) 1529-1531.
- [44] G.-B. Huang, D.-H. Wang, Y. Lan, Extreme learning machines: a survey, *Int. J. Mach. Learn. Cybern.* 2 (2011) 107-122.
- [45] Y. Ito, Representation of functions by superpositions of a step or sigmoid function and their application to neural networks theory, *Neural Netw.* 4 (3) (1991) 385-394.
- [46] R.A. Jacobs, Increased rates of convergence through learning rate adaptation, *Neural Netw.* 2 (1988) 359-366.
- [47] E.M. Johanson, F.U. Dowla, D.M. Goodman, Backpropagation learning for multilayer feedforward neural networks using conjugate gradient method, *Int. J. Neural Syst.* 2 (4) (1992) 291-301.
- [48] A.H. Kramer, A. Sangiovanni-Vincentelli, Efficient parallel learning algorithms for neural networks, in: D.S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 1*, NIPS, Morgan Kaufmann, San Francisco, CA, 1989, pp. 40-48.
- [49] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, *Neural Comput.* 1 (4) (1989) 541-551.
- [50] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage, in: D.S. Touretzky (Ed.), *Advances in Neural Information Systems*, vol. 2, Morgan Kaufmann, San Francisco, CA, 1990, pp. 598-605.
- [51] Y. LeCun, L. Bottou, G.B. Orr, K.R. Müller, Efficient BackProp, in: G.B. Orr, K.-R. Müller (Eds.), *Neural Networks: Tricks of the Trade*, Springer, New York, 1998, pp. 9-50.
- [52] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, *Proc. IEEE* 86 (11) (1998) 2278-2324.
- [53] T.S. Lee, D. Mumford, Hierarchical Bayesian inference in the visual cortex, *J. Opt. Soc. Am. A* 20 (7) (2003) 1434-1448.
- [54] T.S. Lee, D.B. Mumford, R. Romero, V.A.F. Lamme, The Role of the Primary Visual Cortex in Higher Level Vision, *Vis. Res.* 38 (1998) 2429-2454.
- [55] N. Le Roux, Y. Bengio “Representational power of restricted boltzmann machines and deep belief networks,” *Neural Computation*, vol. 20(6), pp. 1631–1649, 2008.
- [56] D.J.C. MacKay, A practical Bayesian framework for back-propagation networks, *Neural Comput.* 4 (3) (1992) 448-472.

- [57] D.J.C. MacKay, The evidence framework applied to classification networks, *Neural Comput.* 4 (5) (1992) 720-736.
- [58] T.K. Marks, J.R. Movellan, Diffusion networks, product of experts, and factor analysis, in: *Proceedings International Conference on Independent Component Analysis*, 2001, pp. 481-485.
- [59] W. McCulloch, W. Pitts, A logical calculus of ideas immanent in nervous activity, *Bull. Math. Biophys.* 5 (1943) 115-133.
- [60] D.B. Mumford, On the computational architecture of the neocortex. II. The role of cortico-cortical loops, *Biol. Cybern.* 66 (1992) 241-251.
- [61] A.B. Navikoff, On convergence proofs on perceptrons, in: *Symposium on the Mathematical Theory of Automata*, vol. 12, Polytechnic Institute of Brooklyn, Brooklyn, 1962, pp. 615-622.
- [62] R. Neal, Connectionist learning of belief networks, *Artif. Intell.* 56 (1992) 71-113.
- [63] A. van den Oord, S. Dieleman, B. Schrauwen, Deep content-based music recommendation, in: *Proceedings Neural Information Processing Systems*, 2013.
- [64] P. Orponen, Computational complexity of neural networks: A survey, *Nordic J. Comput.* 1 (1) (1994) 94-110.
- [65] S.J. Perantonis, P.J.G. Lisboa, Translation, rotation, and scale invariant pattern recognition by high-order neural networks and moment classifiers, *IEEE Trans. Neural Netw.* 3 (2) (1992) 241-251.
- [66] A. Pikrakis, S. Theodoridis, Speech-music discrimination: A deep learning perspective, in: *Proceedings of the 22nd European Signal Processing Conference (EUSIPCO)*, 1-5 September 2014, Lisbon, Portugal, 2014.
- [67] R. Rajesh, J.S. Prakash, Extreme learning machines—A review and state-of-the-art, *Int. J. Wisdom Based Comput.* 1 (1) (2011) 35-49.
- [68] S. Ramon y Cajal, *Histologie du Systéms Nerveux de l' Homme et des Vertebes*, vols. I, II, Maloine, Paris, 1911.
- [69] L.P. Ricotti, S. Ragazzini, G. Martinelli, Learning the word stress in a suboptimal second order backpropagation neural network, in: *Proceedings IEEE International Conference on Neural Networks*, San Diego, vol. 1, 1988, pp. 355-361.
- [70] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the prop algorithm, in: *Proceedings of the IEEE Conference on Neural Networks*, San Francisco, 1993.
- [71] S. Rifai, P. Vincent, X. Muller, X. Glorot, Y. Bengio, Contractive auto-encoders: explicit invariance during feature extraction, in: *Proceedings of the 28th International Conference on Machine Learning (ICML)*, Bellevue, WA, USA, 2011.
- [72] J. Rissanen, G.G. Langdon, Arithmetic coding, *IBM J. Res. Dev.* 23 (1979) 149-162.
- [73] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychol. Rev.* 65 (1958) 386-408.
- [74] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan, Washington, DC, 1962.
- [75] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by backpropagating errors, *Nature* 323 (1986) 533-536.
- [76] R. Russel, Pruning algorithms: a survey, *IEEE Trans. Neural Netw.* 4 (5) (1993) 740-747.
- [77] T. Serre, G. Kreiman, M. Koun, C. Cadieu, U. Knoblich, T. Poggio, A quantitative theory of immediate visual recognition, in: *Progress in Brain Research, Computational Neuroscience: Theoretical Insights into Brain Function*, vol. 165, 2007, pp. 33-56.
- [78] J. Sietsma, R.J.F. Dow, Creating artificial neural networks that generalize, *Neural Netw.* 4 (1991) 67-79.
- [79] E.M. Silva, L.B. Almeida, Acceleration techniques for the backpropagation algorithm, in: L.B. Almeida, et al. (Eds.) *Proceedings on the EURASIP Workshop on Neural Networks*, Portugal, 1990, pp. 110-119.
- [80] P.Y. Simard, D. Steinkraus, J. Platt, Best practice for convolutional neural networks applied to visual document analysis, in: *Proceedings International Conference on Document Analysis and Recognition*, ICDAR, 2003, pp. 958-962.

- [81] K. Simonyan, A. Vedaldi, A. Zisserman, Deep Fisher networks for large-scale image classification, in: Proceedings Neural Information processing Systems, 2013.
- [82] P. Smolensky, Information processing in dynamical systems: foundations of harmony theory, in: Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, 1986, pp. 194-281.
- [83] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov “Dropout: A simple way to prevent neural networks from overfitting” Journal of Machine Learning Research, Vol. 15, pp. 1929–1958, 2014.
- [84] I. Sutskever, T. Tieleman, On the convergence properties of contrastive divergence, in: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 9, Chia Laguna Resort, Sardinia, Italy, 2010.
- [85] K. Swersky, B. Chen, B. Marlin, N. Nando de Freitas, A tutorial on stochastic approximation algorithms for training restricted Boltzmann machines and deep belief nets, in: Proceedings of the Information Theory and Applications Workshop (ITA), San Diego, 31 January 2010-5 February 2010, 2010, pp. 1-10.
- [86] C. Szegedy, A. Toshev, D. Erhan, Deep neural networks for object detection, in: Proceedings Neural Information Processing Systems, 2013.
- [87] G.W. Taylor, G.E. Hinton, S.T. Roweis, Modeling human motion using binary latent variables, in: Advances in Neural Information Processing Systems, 2006, pp. 1345-1352.
- [88] S. Theodoridis, K. Koutroumbas, Pattern Recognition, fourth ed., Academic Press, Boston, 2009.
- [89] T. Tieleman, Training restricted Boltzmann machines using approximations to the likelihood gradient, in: Proceedings of the 25th International Conference on Machine Learning, ACM, New York, NY, USA, 2008, pp. 1064-1071.
- [90] P.E. Utgoff, D.J. Stracuzzi, Many-layered learning, Neural Comput. 14 (2002) 2497-2539.
- [91] V.N. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.
- [92] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2010) 3371-3408.
- [93] P. Vincent, A connection between score matching and denoising autoencoders, Neural Comput. 23 (7) (2011) 1661-1674.
- [94] N. Wang, D.-Y. Yeung, Learning a deep compact image representation for visual tracking, in: Proceedings Neural Information processing Systems (NIPS), 2013.
- [95] R.L. Watrous, Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization, in: Proceedings on the IEEE International Conference on Neural Networks, vol. 2, 1988, pp. 619-627.
- [96] A.S. Weigend, D.E. Rumelhart, B.A. Huberman, Backpropagation, weight elimination and time series prediction, in: D. Touretzky, J. Elman, T. Sejnowski, G. Hinton (Eds.), Proceedings, Connectionist Models Summer School, 1990, pp. 105-116.
- [97] P.J. Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, PhD Thesis, Harvard University, Cambridge, MA, 1974.
- [98] B. Widrow, M.E. Hoff Jr., Adaptive switching networks, IRE WESCON Convention Record, 1960, pp. 96-104.
- [99] W. Wiegerinck, A. Komoda, T. Heskes, Stochastic dynamics on learning with momentum in neural networks, J. Phys. A 25 (1994) 4425-4437.
- [100] A. Yuille, The convergence of contrastive divergences, in: L.K. Saul, W. Weiss, L. Bottou (Eds.), Advances in Neural Information Processing Systems (NIPS), vol. 17, 2004, pp. 1593-1601.
- [101] L. Younes, Parametric inference for imperfectly observed Gibbsian fields, Probab. Theory Relat. Fields 82 (4) (1989) 625-645.
- [102] J. Zourada, Introduction to Artificial Neural Networks, West Publishing Company, St. Paul, MN, 1992.

DIMENSIONALITY REDUCTION AND LATENT VARIABLES MODELING

19

CHAPTER OUTLINE

19.1	Introduction	948
19.2	Intrinsic Dimensionality	949
19.3	Principle Component Analysis.....	949
	<i>PCA, SVD, and Low-Rank Matrix Factorization</i>	951
	<i>Minimum Error Interpretation</i>	953
	<i>PCA and Information Retrieval</i>	953
	<i>Orthogonalizing Properties of PCA and Feature Generation</i>	953
	<i>Latent Variables.....</i>	954
19.4	Canonical Correlation Analysis.....	960
19.4.1	Relatives of CCA	963
	<i>Partial least-squares</i>	964
19.5	Independent Component Analysis	965
19.5.1	ICA and Gaussianity	966
19.5.2	ICA and Higher Order Cumulants	967
	<i>ICA Ambiguities.....</i>	968
19.5.3	Non-Gaussianity and Independent Components.....	968
19.5.4	ICA Based on Mutual Information	969
19.5.5	Alternative Paths to ICA	972
	<i>The Cocktail Party Problem</i>	973
19.6	Dictionary Learning: The <i>k</i>-SVD Algorithm	976
	<i>Why the Name <i>k</i>-SVD.....</i>	978
19.7	Nonnegative Matrix Factorization	981
19.8	Learning Low-Dimensional Models: A Probabilistic Perspective	982
19.8.1	Factor Analysis	982
19.8.2	Probabilistic PCA.....	984
	<i>Mixture of Factors Analyzers: A Bayesian View to Compressed Sensing.....</i>	987
19.9	Nonlinear Dimensionality Reduction	990
19.9.1	Kernel PCA	990
19.9.2	Graph-Based Methods.....	992
	<i>Laplacian Eigenmaps</i>	992
	<i>Local Linear Embedding (LLE)</i>	996
	<i>Isometric Mapping (ISOMAP)</i>	997

19.10 Low-Rank Matrix Factorization: A Sparse Modeling Path	1001
19.10.1 Matrix Completion	1001
19.10.2 Robust PCA	1005
19.10.3 Applications of Matrix Completion and ROBUST PCA	1006
<i>Matrix Completion</i>	1006
<i>Robust PCA/PCP</i>	1007
19.11 A Case Study: fMRI Data Analysis	1008
Problems	1012
<i>MATLAB Exercises</i>	1012
References	1013

19.1 INTRODUCTION

In many practical applications, although the data reside in a high-dimensional space, the true dimensionality, known as *intrinsic dimensionality*, can be of a much lower value. We have met such cases in the context of sparse modeling in Chapter 9. There, although the data lay in a high-dimensional space, a number of the components were known to be zero. The task was to learn the locations of the zeros; this is equivalent with learning the specific subspace, which is determined by the locations of the nonzero components. In this chapter, the goal is to treat the task in a more general setting and assume that the data can live in any possible subspace (not only the ones formed by the removal of coordinate axes) or manifold. For example, in a three-dimensional space, the data may cluster around a straight line, or around the circumference of a circle or the graph of a parabola, arbitrarily placed in \mathbb{R}^3 . In all previous cases, the intrinsic dimensionality of the data is equal to one, as any of these curves can equivalently described in terms of a single parameter. Figure 19.1 illustrates the three cases. Learning the lower dimensional structure associated with a given set of data is gaining in importance in the context of *big data* processing and analysis. Some typical examples are the disciplines of computer vision, robotics, medical imaging, and computational neuroscience.

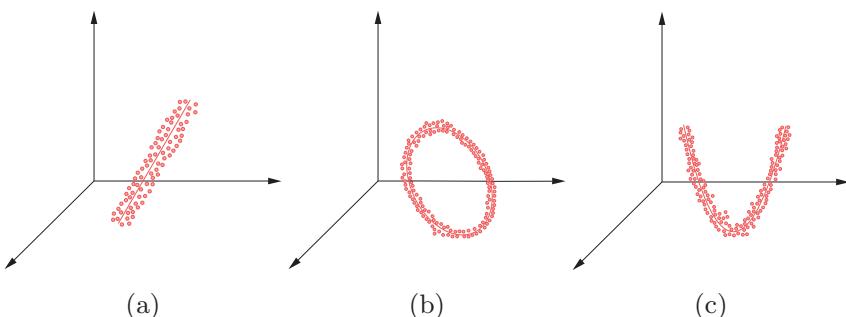


FIGURE 19.1

The data reside close to: (a) a straight line, (b) the circumference of a circle, and (c) the graph of a parabola in the three-dimensional space. In all three cases, the intrinsic dimensionality of the data is equal to one. In (a) the data are clustered around a (translated/affine) linear subspace and in (b) and (c) around one-dimensional manifolds.

The goal of this chapter is to introduce the reader to the main directions, which are followed in this topic, starting from more classical techniques such as the principle component analysis (PCA) and the factor analysis, both in their standard as well as in their more recent probabilistic formulations. Canonical correlation analysis (CCA), independent component analysis (ICA), nonnegative matrix factorization (NMF), and dictionary learning techniques are also discussed; in the latter case, data are represented via an expansion in terms of overcomplete dictionaries, and sparsity-related arguments are mobilized to detect the most relevant atoms in the dictionary. Finally, nonlinear techniques for learning (nonlinear) manifolds are presented such as the kernel PCA, the local linear embedding (LLE), and the isometric mapping (ISOMAP) techniques. At the end of the chapter, a case study in the context of fMRI data analysis is presented.

19.2 INTRINSIC DIMENSIONALITY

A data set, $\mathcal{X} \subset \mathbb{R}^l$, is said to have *intrinsic dimensionality* $m \leq l$, if \mathcal{X} can be (approximately) described in terms of m free parameters. Take as an example the case where the vectors in \mathcal{X} are generated as functions in terms of m random variables, that is, $\mathbf{x} = \mathbf{g}(u_1, \dots, u_m)$, $u_i \in \mathbb{R}$, $i = 1, \dots, m$. The corresponding geometric interpretation is that the respective observation vectors will lie along a manifold, whose form depends on the vector valued function $\mathbf{g} : \mathbb{R}^m \mapsto \mathbb{R}^l$. Let us consider the case where

$$\mathbf{x} = [r \cos \theta, r \sin \theta]^T,$$

where r is a constant and the random variable $\theta \in [0, 2\pi]$. The data lie along the circumference of a circle of radius r and a single free parameter suffices to describe the data. If now a small amount of noise is added, then the data will be clustered close to the circumference, as for example in [Figure 19.1b](#), and the intrinsic dimensionality is equal to one. From a statistical point of view, it means that the components of the random vectors are highly correlated. Sometimes, we say that, the “effective” dimensionality is lower than the apparent one of the “ambient” space, in which the lower dimensional manifold lies.

In a more general setting, the data may lie in groups of manifolds or even in groups of clusters or they may follow a special spatial or temporal structure. For example, in the wavelet domain most of the coefficients of an image are close to zero and can be neglected, yet the larger (nonzero) ones have a particular structure that is characteristic of natural images. Such a structured sparsity has been exploited in the JPEG2000 coding scheme. Structured sparsity representations are often met in many big data applications and are currently a hot topic of research; see, for example, [\[41\]](#). In this chapter, we will only focus on identifying manifold structures, linear (subspaces/affine subspaces) in the beginning, and nonlinear ones later on.

Learning the manifold in which a data set resides can be used to provide a compact low-dimensional encoding of a high-dimensional data set, which can subsequently be exploited for performing processing and learning tasks in a much more efficient way. Also, dimensionality reduction can be used for data visualization.

19.3 PRINCIPLE COMPONENT ANALYSIS

Principle component analysis (PCA) or *Karhunen-Loève transform* is among the oldest and most widely used methods for dimensionality reduction, [\[98\]](#). The assumption underlying PCA, as well as any

dimensionality reduction technique, is that the observed data are generated by a system or process that is driven by a (relatively) small number of *latent* (not directly observed) variables. The goal is to learn this latent structure.

Given a set of observation vectors, $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, of a random vector \mathbf{x} , which will be assumed to be of zero-mean (otherwise the mean/sample mean is subtracted), PCA determines a *subspace* of dimension $m \leq l$, such that after projection on this subspace, the statistical variation of the data is optimally retained. This subspace is defined in terms of m *mutually orthogonal axes*, known as *principle axes* or *principle directions*, which are computed so that the variance of the data, after projection on the subspace, is maximized, [88].

We will derive the principle axes in a step-wise fashion. First, assume that $m = 1$ and the goal is to find a single direction in \mathbb{R}^l so that the variance of the corresponding projections of the data points is maximized. Let \mathbf{u}_1 denote the principle axis. The variance of the projections (and having assumed centered data) is given by

$$\begin{aligned} J(\mathbf{u}_1) &= \frac{1}{N} \sum_{n=1}^N (\mathbf{u}_1^T \mathbf{x}_n)^2 = \frac{1}{N} \sum_{n=1}^N (\mathbf{u}_1^T \mathbf{x}_n)(\mathbf{x}_n^T \mathbf{u}_1) \\ &= \mathbf{u}_1^T \hat{\Sigma} \mathbf{u}_1, \end{aligned}$$

where

$$\hat{\Sigma} := \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^T, \quad (19.1)$$

is the sample covariance matrix of the data. For large values of N or if the statistics can be computed, the covariance (instead of the sample covariance) matrix can be used. The task now becomes that of maximizing the variance. However, because we are only interested in directions, the principle axis will be represented by the respective unit norm vector. Thus, the optimization task is cast as

$$\boxed{\mathbf{u}_1 = \arg \max_{\mathbf{u}} \mathbf{u}^T \hat{\Sigma} \mathbf{u},} \quad (19.2)$$

$$\text{s.t. } \mathbf{u}^T \mathbf{u} = 1. \quad (19.3)$$

This is a constrained optimization problem and the corresponding Lagrangian is given by

$$L(\mathbf{u}, \lambda) = \mathbf{u}^T \hat{\Sigma} \mathbf{u} - \lambda(\mathbf{u}^T \mathbf{u} - 1). \quad (19.4)$$

Taking the gradient and setting it equal to zero we get

$$\hat{\Sigma} \mathbf{u} = \lambda \mathbf{u}. \quad (19.5)$$

In other words, the principle direction is an eigenvector of the sample covariance matrix. Plugging Eq. (19.5) into Eq. (19.2) and taking into account (19.3), we obtain that

$$\mathbf{u}^T \hat{\Sigma} \mathbf{u} = \lambda. \quad (19.6)$$

Hence, the variance is maximized if \mathbf{u}_1 is the eigenvector that corresponds to the maximum eigenvalue, λ_1 . Recall that, because the (sample) covariance matrix is symmetric and positive semidefinite all the

eigenvalues are real and nonnegative. Assuming $\hat{\Sigma}$ to be invertible (hence, necessarily, $N > l$), the eigenvalues are all positive, that is, $\lambda_1 > \lambda_2 > \dots > \lambda_l > 0$, and we also assume they are distinct, in order to simplify the discussion.

The second principle component is selected so that: (a) is orthogonal to \mathbf{u}_1 and (b) maximizes the variance after projecting the data onto this direction. Following similar arguments as before, a similar optimization task results with an extra constraint, $\mathbf{u}^T \mathbf{u}_1 = 0$. It can easily be shown ([Problem 19.1](#)) that the second principle axis is the eigenvector corresponding to the second largest eigenvalue, λ_2 . The process continues until m principle axes have been obtained; they are the eigenvectors corresponding to the m largest eigenvalues.

PCA, SVD, and low-Rank matrix factorization

The SVD decomposition of a matrix was discussed in [Section 6.4](#). Given a matrix $X \in \mathbb{R}^{l \times N}$, we can write

$$X = UDV^T. \quad (19.7)$$

For a rank r matrix X , U is the $l \times r$ matrix having as columns the eigenvectors corresponding to the r nonzero eigenvalues of XX^T , and V is the $N \times r$ matrix with columns the respective eigenvectors of $X^T X$. D is a square $r \times r$ diagonal matrix comprising the singular values¹ $\sigma_i := \sqrt{\lambda_i}$, $i = 1, 2, \dots, r$. If we construct X to have as columns the data vectors \mathbf{x}_n , $n = 1, 2, \dots, N$, then XX^T is a scaled version of the corresponding sample covariance matrix, $\hat{\Sigma}$; hence, the respective eigenvectors coincide and the corresponding eigenvalues are equal to within a scaling factor (N). Without harming generality, we can assume XX^T to be full rank ($r = l < N$), and Eq. (19.7) becomes

$$X = \underbrace{[\mathbf{u}_1, \dots, \mathbf{u}_l]}_{l \times l} \begin{bmatrix} \sqrt{\lambda_1} \mathbf{v}_1^T \\ \vdots \\ \sqrt{\lambda_l} \mathbf{v}_l^T \end{bmatrix}. \quad (19.8)$$

Thus, the columns of X can be written in terms of the following expansion²

$$\mathbf{x}_n = \sum_{i=1}^l z_{ni} \mathbf{u}_i = \sum_{i=1}^m z_{ni} \mathbf{u}_i + \sum_{i=m+1}^l z_{ni} \mathbf{u}_i, \quad (19.9)$$

where $\mathbf{z}_n^T := [z_{n1}, \dots, z_{nl}]$ is the n th column of the $l \times N$ factor on the right-hand side in Eq. (19.8) and the sum has been split into two terms where m can be any value $1 \leq m \leq l$. Note that, due to the orthonormality of the \mathbf{u}_i 's,

$$z_{ni} = \mathbf{u}_i^T \mathbf{x}_n, \quad i = 1, 2, \dots, l, \quad n = 1, 2, \dots, N.$$

¹ Because in some places we are going to involve the variance σ^2 , we will carry on working with the square root of the eigenvalues, to avoid possible confusion.

² Note that what we have defined in previous chapters as the data matrix is the transpose of X . This is because, for dimensionality reduction tasks, it is more common to work with the current notational convention. If the transpose of X is used, the expansion of the data vectors is in terms of the columns of V and the analysis carries on in a similar way.

From [Section 6.4](#), we know that the best, in the Frobenius sense, m -rank matrix approximation of X is given by

$$\hat{X} = \underbrace{[\mathbf{u}_1, \dots, \mathbf{u}_m]}_{l \times m} \underbrace{\begin{bmatrix} \sqrt{\lambda_1} \mathbf{v}_1^T \\ \vdots \\ \sqrt{\lambda_m} \mathbf{v}_m^T \end{bmatrix}}_{m \times N}, \quad (19.10)$$

$$= \sum_{i=1}^m \sqrt{\lambda_i} \mathbf{u}_i \mathbf{v}_i^T. \quad (19.11)$$

Recalling the previous definition of z_{ni} , the n th column vector of \hat{X} can now be written as

$$\hat{x}_n = \sum_{i=1}^m z_{ni} \mathbf{u}_i. \quad (19.12)$$

Comparing Eqs. (19.9) and (19.12) and taking into account the orthonormality of \mathbf{u}_i , $i = 1, 2, \dots, l$, we readily see that \hat{x}_n is the projection of the original observation vectors, \mathbf{x}_n , $n = 1, 2, \dots, N$, onto the subspace $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_m\}$ generated by the m principle axes of XX^T ($\hat{\Sigma}$) ([Figure 19.2](#)).

The previous arguments establish a bridge between PCA and SVD. In other words, the principle axes can be obtained via the SVD decomposition of X . Moreover, the columns of the best m -rank matrix approximation, \hat{X} , of X are the projections of the observation vectors \mathbf{x}_n on the (optimally) reduced in dimension subspace, spanned by the principle axes.

Looking at Eq. (19.10), PCA can also be seen as a *low-rank matrix factorization* method. Matrix factorization will be a recurrent theme in this chapter. Given a matrix, X , there is not a unique way to factorize it, in terms of two matrices. PCA provides an m -rank matrix factorization of X , by imposing orthogonality on the structure of the involved factors. Later on, we are going to discuss other approaches.

Finally, it is important to emphasize that the bridge between PCA and SVD establishes a connection between the low-rank factorization of a matrix, X , and the intrinsic dimensionality of the subspace in which its column vectors reside, since this is the subspace where maximum variance of the data is guaranteed.

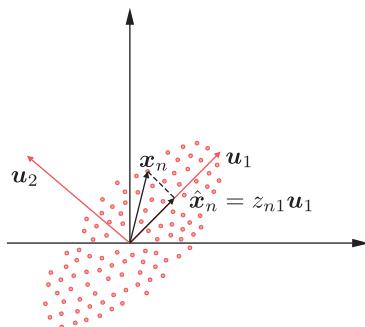


FIGURE 19.2

The projection of \mathbf{x}_n on the principle axis \mathbf{u}_1 is given by $\hat{\mathbf{x}}_n = z_{n1} \mathbf{u}_1$, where $z_{n1} = \mathbf{u}_1^T \mathbf{x}_n$.

Minimum error interpretation

Having established the bridge between PCA and SVD, another interpretation of the PCA method becomes readily available. Because \hat{X} is the best m -rank matrix approximation of X in the Frobenius sense, we have that the quantity

$$\|\hat{X} - X\|_F^2 := \sum_i \sum_j |\hat{X}(i,j) - X(i,j)|^2 = \sum_{n=1}^N \|\hat{x}_n - x_n\|^2$$

is minimum; that is, obtaining any other m -dimensional approximation (say \tilde{x}_n) of x_n , by choosing to project onto another m -dimensional subspace, would result in higher squared error norm approximation, compared to that resulting from PCA. This is also a strong result that establishes a notable merit of the PCA method as a dimensionality reduction technique. This interpretation goes back to Pearson, [137].

PCA and information retrieval

The previous minimum error interpretation paves the way to build around PCA an efficient searching procedure in identifying similar patterns in large databases. Assume that a number N of prototypes are represented in terms of l features, giving rise to feature vectors, $x_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, which are stored in a database. Given an unknown object, which is represented by a feature vector x , the task is to identify to which one among the prototypes this pattern is most similar. Similarity is measured in terms of the Euclidean distance $\|x - x_n\|^2$. If N and l are large, searching for the minimum Euclidean distance can be computationally very expensive. The idea is to keep in the database the components $z_n^{(m)} := [z_{n1}, \dots, z_{nm}]^T$ (see Eq. (19.12)) that describe the projections of the N prototypes in $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_m\}$, instead of the original l dimensional feature vectors. Assuming that m is large enough to capture most of the variability of the original data (i.e., the intrinsic dimensionality of the data is m to a good approximation), then $z_n^{(m)}$ is a good feature vector description because we know that in this case $\hat{x}_n \approx x_n$. Given now an unknown pattern, x , we first project it onto $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_m\}$ resulting in

$$\hat{x} = \sum_{i=1}^m (\mathbf{u}_i^T x) \mathbf{u}_i := \sum_{i=1}^m z_i \mathbf{u}_i. \quad (19.13)$$

Then we have

$$\begin{aligned} \|x_n - x\|^2 &\approx \|\hat{x}_n - \hat{x}\|^2 = \left\| \sum_{i=1}^m z_{ni} \mathbf{u}_i - \sum_{i=1}^m z_i \mathbf{u}_i \right\|^2 \\ &= \|z_n^{(m)} - z\|^2, \end{aligned}$$

where $z := [z_1, \dots, z_m]^T$. In other words, Euclidean distances are computed in the lower dimensional subspace, which leads to substantial computational gains; see, for example, [21, 58, 150] and the references therein. This method is also known as *latent semantics indexing*.

Orthogonalizing properties of PCA and feature generation

We will now shed light on PCA from a different angle. We have just discussed, in the context of the information retrieval application, that PCA can also be seen as a feature generation method that generates a set of new feature vectors, z , whose components describe a pattern in terms of the principle axes. Let us now assume (to make life easier) that N is large enough and the sample covariance matrix

is a good approximation of the (full rank) covariance matrix $\Sigma = \mathbb{E}[\mathbf{x}\mathbf{x}^T]$. We know that any vector $\mathbf{x} \in \mathbb{R}^l$ can be described in terms of $\mathbf{u}_1, \dots, \mathbf{u}_l$, that is,

$$\mathbf{x} = \sum_{i=1}^l z_i \mathbf{u}_i = \sum_{i=1}^l (\mathbf{u}_i^T \mathbf{x}) \mathbf{u}_i.$$

Our focus now turns to the covariance matrix of the random vectors, \mathbf{z} , as \mathbf{x} changes randomly. Taking into account that

$$z_i = \mathbf{u}_i^T \mathbf{x}, \quad (19.14)$$

and the definition of U in Eqs. (19.7)–(19.8), we can write as $\mathbf{z} = U^T \mathbf{x}$, hence,

$$\mathbb{E}[\mathbf{z}\mathbf{z}^T] = \mathbb{E}\left[U^T \mathbf{x} \mathbf{x}^T U\right] = U^T \Sigma U.$$

However, we know from linear algebra (Appendix A.2) that U is the matrix that diagonalizes Σ , hence,

$$\mathbb{E}[\mathbf{z}\mathbf{z}^T] = \text{diag}\{\lambda_1, \dots, \lambda_l\}. \quad (19.15)$$

In other words, the new features are *uncorrelated*, that is,

$$\boxed{\mathbb{E}[z_i z_j] = 0, \quad i \neq j, \quad i, j = 1, 2, \dots, l.} \quad (19.16)$$

Furthermore, note that, the variances of z_i are equal to the eigenvalues λ_i , $i = 1, 2, \dots, l$, respectively. Hence, by selecting as features the ones that correspond to the dominant eigenvalues, one has maximally retained the total variance associated with the original features, x_i ; indeed, the corresponding total variance is given by the trace of the covariance matrix, which in turn is equal to the sum of the eigenvalues, as we know from linear algebra. In other words, the new set of features, z_i , $i = 1, 2, \dots, m$, represent the patterns in a more compact way, as they are *mutually uncorrelated* and most of the variance is retained. It is common in practice, when the goal is that of feature generation, for each one of the z_i 's to be normalized to unit variance.

Later on, we will see that a more recent method, known as independent component analysis (ICA), imposes the constraint that after a linear transformation (a projection is a linear transformation, after all) the obtained latent variables (components) are statistically independent, which is a much stronger condition than being uncorrelated.

Latent variables

The random components, z_i , $i = 1, 2, \dots, m$, are known as *principle components*. Sometimes, their observed values, z_i , are known as *principle scores*. As a matter of fact, the principle components comprise the *latent* variables, which we mentioned at the beginning of this section.

According to the general (linear) latent variables modeling approach, we assume that our l variables comprising, \mathbf{x} , are modeled as

$$\mathbf{x} \approx A\mathbf{z}, \quad (19.17)$$

where A is an $l \times m$ matrix and $\mathbf{z} \in \mathbb{R}^m$ is the corresponding set of latent variables. Adopting the PCA model, we have shown that

$$A = U = [\mathbf{u}_1, \dots, \mathbf{u}_m],$$

and the model implies that each one of the l components of \mathbf{x} is (approximately) generated in terms of these mutually uncorrelated m latent random variables, that is,

$$\mathbf{x}_i \approx u_{i1}\mathbf{z}_1 + \dots + u_{im}\mathbf{z}_m. \quad (19.18)$$

Alternatively, in the linear latent variables modeling, we can assume that the latent variables can also be recovered by a linear model from the original random variables, as for example,

$$\mathbf{z} = W\mathbf{x}. \quad (19.19)$$

In the case of the PCA approach, we have already seen that

$$W = U^T.$$

Equations (19.17) and (19.19) constitute the backbone of this chapter, and different methods provide different solutions for computing A or W .

Let us now collect all the principle score vectors, \mathbf{z}_n , $n = 1, 2, \dots, N$, as the columns of the $m \times N$ score matrix Z , that is,

$$Z := [\mathbf{z}_1, \dots, \mathbf{z}_N]. \quad (19.20)$$

Then (19.10) can be rewritten in terms of the score matrix

$$X \approx UZ. \quad (19.21)$$

Moreover, taking into account the definition of the principle components in Eq. (19.14), we can also write

$$Z = U^TX. \quad (19.22)$$

Remarks 19.1.

- A major issue in practice is to select the m dominant eigenvalues. One way is to rank them in descending order, and determine m so that the gap between λ_m and λ_{m+1} is “large.” The interested reader can obtain more on this issue in [51, 97].
- The treatment so far involved centered quantities. In case we want to approximate the original observation vectors by taking into consideration the respective mean value of the data set, Eq. (19.13) is rephrased as

$$\hat{\mathbf{x}} = \bar{\mathbf{x}} + \sum_{i=1}^m \mathbf{u}_i^T (\mathbf{x} - \bar{\mathbf{x}}) \mathbf{u}_i, \quad (19.23)$$

where $\bar{\mathbf{x}}$ is the sample mean (mean if it is known)

$$\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n,$$

and \mathbf{x} denotes the original (not centered) vector.

- PCA builds upon *global* information spread over *all* the data observations in the set \mathcal{X} . Indeed, the main source of information is the sample covariance matrix $(\mathbf{X}\mathbf{X}^T)$. Thus, PCA is effective if the covariance matrix provides a sufficiently rich description of the data at hand. For example, this is the case for Gaussian-like distributions. In [40], modifications of the standard approach are

suggested in order to deal with data having a clustered nature. Soon, we are going to discuss alternative to PCA techniques in order to overcome this drawback.

- Computing the SVD of large matrices can be computationally costly and a number of efficient techniques have been proposed (see, e.g., [1, 77, 183]). In a number of cases in practice, it turns out that $l > N$. Of course, in this case, the sample covariance is not invertible and some of the eigenvalues are zero. In such scenarios, it is preferable to work with $X^T X$ ($N \times N$) instead of XX^T ($l \times l$) matrix. To this end, the relationships given in [Section 6.4](#), in order to obtain \mathbf{u}_i from \mathbf{v}_i , can be employed.
- The treatment of PCA bears a similarity with the Fisher's linear discriminant method (FLD) ([Chapter 7](#)). They both rely on the eigenstructure of matrices that, in one way or another, encode (co)variance information. However, note that PCA is an *unsupervised* method in contrast to FLD, which is a *supervised* one. As a consequence, PCA performs dimensionality reduction so as to preserve data variability (variance) while FLD class separability. [Figure 19.3](#) demonstrates the difference in the resulting (hyper)planes.
- *Multidimensional Scaling* (MDS) is another linear technique used to project in a lower dimensional space, while respecting certain constraints. Given the set $\mathcal{X} \subset \mathbb{R}^l$, the goal is to project onto a lower dimensional space, so that inner products are optimally preserved; that is, the cost

$$E = \sum_i \sum_j (\mathbf{x}_i^T \mathbf{x}_j - \mathbf{z}_i^T \mathbf{z}_j)^2$$

is minimized, where \mathbf{z}_i is the image of \mathbf{x}_i and the sum runs over all the training points in \mathcal{X} . The problem is similar to the PCA and it can be shown that the solution is given by the eigendecomposition of the Gram matrix,³ $\mathcal{K} := X^T X$. Another side of the same coin is to require the Euclidean distances, instead of the inner products, to be optimally preserved. A Gram matrix, consistent with the squared Euclidean distances, can then be formed, leading to the same solution

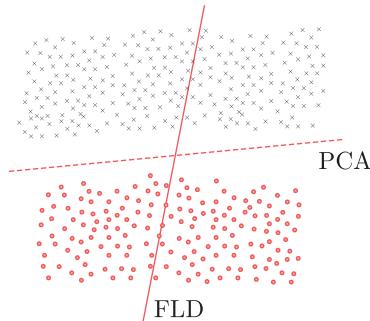


FIGURE 19.3

The case of a two-class task in the two-dimensional space. PCA computes the direction along which the variance is maximally retained after the projections of the data on it. In contrast, FLD computes the line so that the class separability is maximized.

³ In order to avoid confusion, recall that here X has been defined as the transpose of what we called a data matrix in previous chapters.

as before. It turns out that the solutions obtained by PCA and MDS are equivalent. This can readily be understood as $X^T X$ and XX^T share the same (nonzero) eigenvalues. The corresponding eigenvectors are different, yet they are related, as we have seen while introducing the SVD in [Section 6.4](#).

More on these issues can be found in [27, 56]. As we will soon see in [Section 19.9](#), the main idea behind MDS of preserving the distances is used, in one way or another, in a number of more recently developed nonlinear dimensionality reduction techniques.

- In a variant of the basic PCA, known as *supervised PCA* [15, 184], the output variables in regression or in classification (depending on the problem at hand) are used together with the input ones, in order to determine the principle directions.

Example 19.1. This example demonstrates the power of PCA as a method to represent data in a lower dimension space. Each pattern in a database, described in terms of a feature vector, $\mathbf{x}_n \in \mathbb{R}^l$, will be represented by a corresponding vector of a reduced dimensionality, $\mathbf{z}_n^{(m)} \in \mathbb{R}^m$, $n = 1, 2, \dots, N$.

In this example, each feature vector comprises the pixels of a 168×168 face image. These face images are members of the software-based aligned version, [180], of the *labeled faces in the wild* (LFW) database [95]. In particular, among the over 13,000 face images of this database, $N = 1924$ have been selected with criteria such as the quality of the image and the face angle (portraits were of preference). Moreover, the images are zoomed in order to omit most of the background. Examples of the face images used are depicted in [Figure 19.4](#) and the full collection of all the 1924 images can be found in the companion site of this book.

The images are first vectorized (in \mathbb{R}^l , $l = 168 \times 168 = 28,224$) and in the sequel are concatenated in the columns of the $28,224 \times 1924$ matrix X . Moreover, the mean value across each one of the rows is computed and then subtracted from the corresponding element of each column.

In this case, where $l > N$, it is convenient to compute the eigenvectors of $X^T X$, denoted by \mathbf{v}_i , $i = 1, \dots, N$, and then the principle axes directions, that is, the eigenvectors of XX^T are computed by $\mathbf{u}_i \propto X\mathbf{v}_i$ ([Chapter 6, Eq. 6.16](#)). These eigenvectors can be rearranged in a matrix form to give 168×168 images known as *eigenimages*, which in the particular case of face images are referred to as *eigenfaces*. [Figure 19.5](#) shows examples of eigenfaces resulted by the PCA of matrix X and specifically those corresponding, from top left to bottom right, to the 1st, 2nd, 6th, 7th, 8th, 10th, 11th, and 17th larger eigenvalues.

Next, the quality of reconstruction of an original image, in terms of its lower dimensional representation, is examined according to [Eq. \(19.13\)](#) for different values of m . As an example, the images depicting Marilyn Monroe and Andy Warhol, shown in [Figure 19.4](#), are chosen. The results



FIGURE 19.4

Indicative examples of the face images used.

**FIGURE 19.5**

Examples of eigenfaces.

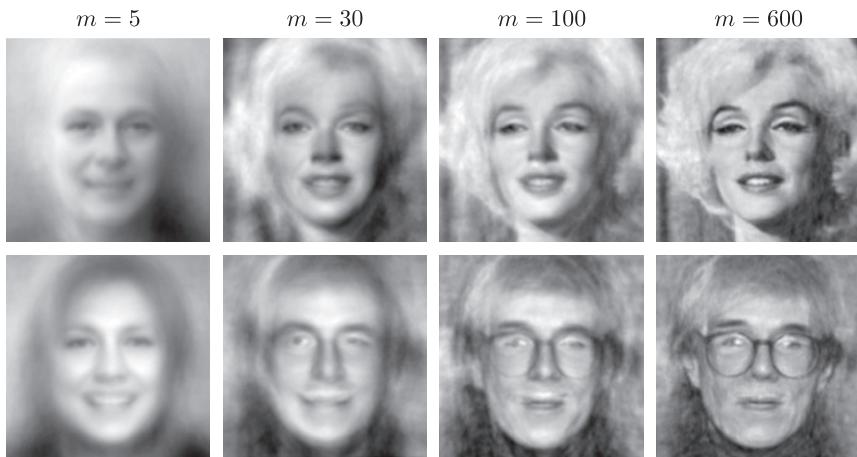
**FIGURE 19.6**

Image compression and reconstruction based on the first m eigenvectors.

are illustrated in Figure 19.6. It is observed that, for $m = 100$ or even better for $m = 600$, the resulting approximation is very close to the original images. Note that exact reconstruction will be achieved when the full set of the 1924 eigenfaces are used.

To put our previous findings in an information retrieval context, assume than one has available an image and wants to know what person is depicted in it. Assuming that the image of this person is in the database, the procedure would be (a) to vectorize the image, (b) to project it onto the subspace spanned by the, say, $m = 100$ eigenfaces, and (c) to search in this lower dimensional space to identify

the vectorized image in the database that is closer in the Euclidean norm sense. Usually, it is preferable to identify the, say, five or ten most similar images and rank them according to the Euclidean distance (or any other distance) similarity. Then, through the database, he/she can have the name and all the associated information that is kept in the database.

In information retrieval, each one of the images in the database, could be stored in terms of the corresponding vector of the principle scores.

Example 19.2. In this example, the use of PCA for image compression is demonstrated. In the previous example, PCA was performed across the different images of a database. Here, the focus will be on a single image.

The pixel values of the image are stored in an $l \times N$ matrix X and the columns of this matrix are considered to be the observation vectors $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$. Note that X needs to be zero-mean along the rows so the mean vector, $\bar{\mathbf{x}}$, is computed and subtracted from each column. Then the eigenvectors corresponding to the m , $1 \leq m < l$ largest eigenvalues are obtained either via the sample covariance matrix or directly through SVD. Exploiting the matrix factorization formulation of PCA in Eq. (19.22) a compressed representation of X , comprising m instead of l rows, is given by

$$Z^{(m)} = \underbrace{[\mathbf{u}_1, \dots, \mathbf{u}_m]}_{m \times l}^T X, \quad (19.24)$$

where the dimensionality m has been explicitly brought into the notation. Thus, only $Z^{(m)}$ and $\mathbf{u}_1, \dots, \mathbf{u}_m$, are needed in order to get an estimate of the, meansubtracted, X via Eq. (19.21). Finally, in order to reconstruct the image, the mean vector $\bar{\mathbf{x}}$ need to be added back to each column, see Eq. (19.23).

The effectiveness of the PCA-based image compression will be demonstrated with the aid of the top-left image depicted in Figure 19.7. This image is square having $l = N = 400$. For any m chosen, the compression ratio is easily computed considering that instead of 400×400 values, of the original image, after compression the storage of $2 \times m \times 400$ values, for the matrix, $Z^{(m)}$ and the eigenvectors, $\mathbf{u}_1, \dots, \mathbf{u}_m$, plus 400 values for the mean vector $\bar{\mathbf{x}}$ are needed. This amounts to a compression ratio of $400 : (2m + 1)$. The reconstructed images together with the corresponding MSE, between the original and the reconstructed image, for different compression rates, are shown in Figure 19.7.

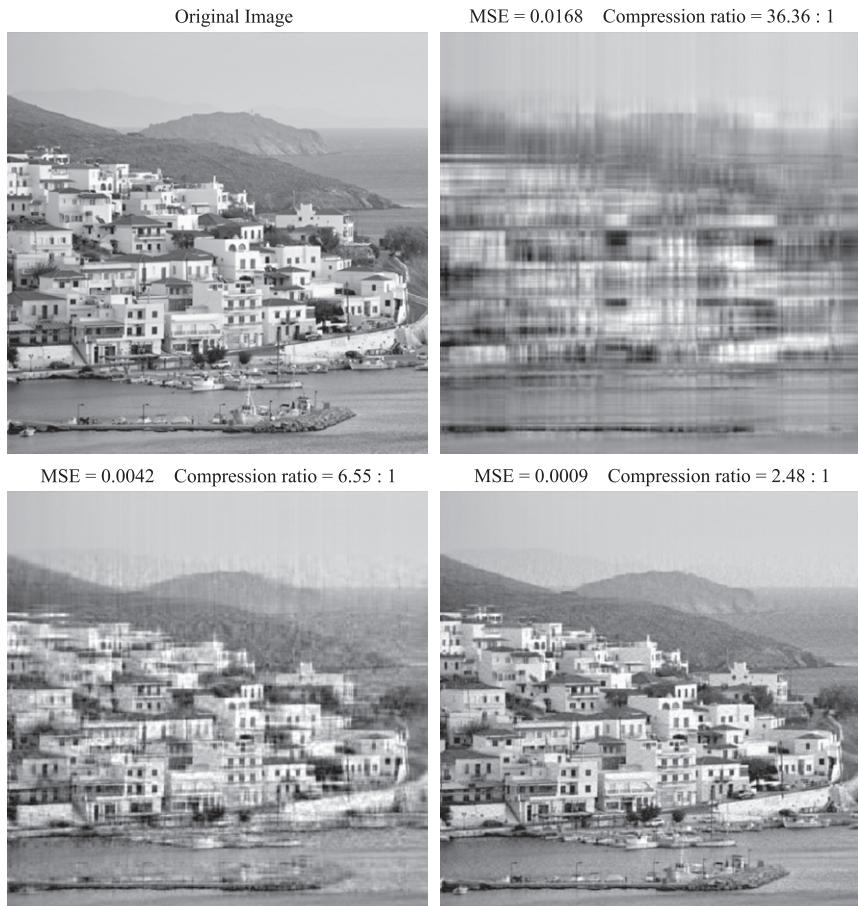
Remarks 19.2.

- *Subspace Tracking:* Online subspace tracking is another old area with a revived interest recently.

A well-known algorithm of relative low complexity, for tracking the signal subspace, is the so-called projection approximation subspace tracking (PAST) proposed in [186]. In PAST, the recursive least-squares (RLS) technique is employed for subspace estimation. Alternative algorithms in this line of philosophy have been presented in, for example, [64, 108, 160, 169].

More recently, the work in [47, 80, 129] tackles the problem of subspace tracking with missing/unobserved data. The methodology presented in [80] is based on gradient descent iterations on the Grassmannian manifold. Furthermore, the algorithms of [47, 129] attempt to estimate the unknown subspace by minimizing properly constructed loss functions.

Finally, [48, 49, 85, 124, 152] attack the subspace tracking problem in environments where observations are contaminated by outlier noise.

**FIGURE 19.7**

PCA-based image compression. The image is from the Greek island Andros.

19.4 CANONICAL CORRELATION ANALYSIS

PCA is a dimensionality reduction technique focusing on a *single* data set. However, in a number of cases, one has to deal with multiple data sets, which although they may originate from different sources, they are closely related. For example, many problems in medical imaging fall under this umbrella. A typical case occurs in the study of brain activity where one can use different modalities, for example, electroencephalogram (EEG), functional magnetic resonance imaging (fMRI), or structural MRI. Each one of these modalities can grasp a different type of information and it is beneficial to exploit all of them in a complementary fashion. Thus, the respective experimental data can appropriately be fused

in order to get a better description concerning the brain activity that gives birth to the data. Another scenario where multiple data sets are of interest, is when a single modality is used but different data are available measured on different subjects; thus, jointly analyzing the results can be beneficial for the finally reached conclusions (see, e.g., [52]).

Canonical correlation analysis (CCA) is an old technique developed in [89] in order to process two data sets jointly. Our starting point is the fact that when two sets of random variables (two random vectors) are involved, the value of their correlation *does depend* on the coordinate system in which the random vectors are represented. The goal behind CCA is to seek a pair of linear transformations, one for each set of variables, such that after the transformation, the resulting transformed variables are *maximally correlated*.

Let us assume that we are given two sets of random variables comprising the components of two random vectors, $\mathbf{x} \in \mathbb{R}^p$ and $\mathbf{y} \in \mathbb{R}^q$, and let the corresponding sets of observations be $\mathbf{x}_n, \mathbf{y}_n$, $n = 1, 2, \dots, N$, respectively. Following a step-wise procedure, as we did for PCA, we will first compute a single pair of directions, namely $\mathbf{u}_{x,1}, \mathbf{u}_{y,1}$, so that the correlation between the projections onto these directions is maximized. Let $\mathbf{z}_{x,1} := \mathbf{u}_{x,1}^T \mathbf{x}$ and $\mathbf{z}_{y,1} := \mathbf{u}_{y,1}^T \mathbf{y}$ be the (zero-mean) random variables after the linear transformation (projection). Note that these variables are the counterparts of what we called principle components in PCA. The corresponding correlation coefficient (normalized covariance) is defined as

$$\rho := \frac{\mathbb{E}[\mathbf{z}_{x,1} \mathbf{z}_{y,1}]}{\sqrt{\mathbb{E}[\mathbf{z}_{x,1}^2] \mathbb{E}[\mathbf{z}_{y,1}^2]}} = \frac{\mathbb{E}[(\mathbf{u}_{x,1}^T \mathbf{x})(\mathbf{y}^T \mathbf{u}_{y,1})]}{\sqrt{\mathbb{E}[(\mathbf{u}_{x,1}^T \mathbf{x})^2] \mathbb{E}[(\mathbf{u}_{y,1}^T \mathbf{y})^2]}}$$

or

$$\rho := \frac{\mathbf{u}_{x,1}^T \Sigma_{xy} \mathbf{u}_{y,1}}{\sqrt{(\mathbf{u}_{x,1}^T \Sigma_{xx} \mathbf{u}_{x,1})(\mathbf{u}_{y,1}^T \Sigma_{yy} \mathbf{u}_{y,1})}}, \quad (19.25)$$

where

$$\mathbb{E}\left[\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} [\mathbf{x}^T, \mathbf{y}^T]\right] := \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{bmatrix}. \quad (19.26)$$

Note that, by the respective definition, we have $\Sigma_{xy} = \Sigma_{yx}^T$. When expectations are not available, covariances are replaced by the corresponding sample covariance values. This is the most common case in practice, so we will adhere to it and use the notation with the “hat.” Furthermore, it can easily be checked out that the correlation coefficient is invariant to scaling (changing, e.g., $\mathbf{x} \rightarrow b\mathbf{x}$). Thus, maximizing it with respect to the directions $\mathbf{u}_{x,1}$ and $\mathbf{u}_{y,1}$, can equivalently be cast as the following constrained optimization task,

$$\max_{\mathbf{u}_x, \mathbf{u}_y} \mathbf{u}_x^T \hat{\Sigma}_{xy} \mathbf{u}_y, \quad (19.27)$$

$$\text{s.t. } \mathbf{u}_x^T \hat{\Sigma}_{xx} \mathbf{u}_x = 1, \quad (19.28)$$

$$\mathbf{u}_y^T \hat{\Sigma}_{yy} \mathbf{u}_y = 1. \quad (19.29)$$

Compare Eqs. (19.27)–(19.29) with the optimization task defining PCA in Eqs. (19.2)–(19.3). For CCA, two directions have to be computed and the constraints involve the weighted Σ -norm instead of the

Euclidean one. Moreover, in PCA the variance is maximized, while CCA cares for the correlation between the projections of the two involved vectors onto the new axes.

Employing Lagrange multipliers, the corresponding Lagrangian of Eqs. (19.27)–(19.29) is given by

$$L(\mathbf{u}_x, \mathbf{u}_y, \lambda_x, \lambda_y) = \mathbf{u}_x^T \hat{\Sigma}_{xy} \mathbf{u}_y - \frac{\lambda_x}{2} (\mathbf{u}_x^T \hat{\Sigma}_{xx} \mathbf{u}_x - 1) - \frac{\lambda_y}{2} (\mathbf{u}_y^T \hat{\Sigma}_{yy} \mathbf{u}_y - 1).$$

Taking the gradients with respect to \mathbf{u}_x and \mathbf{u}_y and equating to zero, we obtain (Problem 19.2)

$$\lambda_x = \lambda_y := \lambda,$$

and

$$\hat{\Sigma}_{xy} \mathbf{u}_y = \lambda \hat{\Sigma}_{xx} \mathbf{u}_x, \quad (19.30)$$

$$\hat{\Sigma}_{yx} \mathbf{u}_x = \lambda \hat{\Sigma}_{yy} \mathbf{u}_y. \quad (19.31)$$

Solving the latter of the two with respect to \mathbf{u}_y and substituting to the first one, we finally get

$$\hat{\Sigma}_{xy} \hat{\Sigma}_{yy}^{-1} \hat{\Sigma}_{yx} \mathbf{u}_x = \lambda^2 \hat{\Sigma}_{xx} \mathbf{u}_x, \quad (19.32)$$

and

$$\mathbf{u}_y = \frac{1}{\lambda} \hat{\Sigma}_{yy}^{-1} \hat{\Sigma}_{yx} \mathbf{u}_x, \quad (19.33)$$

assuming, of course, invertibility of $\hat{\Sigma}_{yy}$. Furthermore, assuming invertibility of $\hat{\Sigma}_{xx}$, too, we end up with the following eigenvalue-eigenvector problem:

$$\left(\hat{\Sigma}_{xx}^{-1} \hat{\Sigma}_{xy} \hat{\Sigma}_{yy}^{-1} \hat{\Sigma}_{yx} \right) \mathbf{u}_x = \lambda^2 \mathbf{u}_x. \quad (19.34)$$

Thus, the axis $\mathbf{u}_{x,1}$ is obtained as an eigenvector of the product of matrices in the parentheses in Eq. (19.34). Taking into account Eq. (19.30) and the constraints, it turns out that the corresponding optimal value of the correlation, ρ , is equal to

$$\rho = \mathbf{u}_{x,1}^T \hat{\Sigma}_{xy} \mathbf{u}_{y,1} = \lambda \mathbf{u}_{x,1}^T \hat{\Sigma}_{xx} \mathbf{u}_{x,1} = \lambda.$$

Hence, selecting $\mathbf{u}_{x,1}$ to be the eigenvector corresponding to the maximum eigenvalue, λ^2 , results in maximum correlation.

The eigenvectors $\mathbf{u}_{x,1}, \mathbf{u}_{y,1}$ are known as the *normalized canonical correlation basis vectors*, the eigenvalue λ^2 as the squared *canonical correlation*, and the projections $\mathbf{z}_{x,1}, \mathbf{z}_{y,1}$ as the *canonical variates*.

The previous idea can now be taken further and compute a pair of subspaces, $\text{span}\{\mathbf{u}_{x,1}, \dots, \mathbf{u}_{x,m}\}$, $\text{span}\{\mathbf{u}_{y,1}, \dots, \mathbf{u}_{y,m}\}$, where $m \leq \min(p, q)$. One way to achieve this goal is in a step-wise fashion, as it was done for the PCA. Assuming that k pairs of basis vectors have already been computed, the $k+1$ is obtained by solving the following constrained optimization task,

$$\max_{\mathbf{u}_x, \mathbf{u}_y} \mathbf{u}_x^T \hat{\Sigma}_{xy} \mathbf{u}_y, \quad (19.35)$$

$$\text{s.t. } \mathbf{u}_x^T \hat{\Sigma}_{xx} \mathbf{u}_x = 1, \quad \mathbf{u}_y^T \hat{\Sigma}_{yy} \mathbf{u}_y = 1, \quad (19.36)$$

$$\mathbf{u}_x^T \hat{\Sigma}_{xx} \mathbf{u}_{x,i} = 0, \quad \mathbf{u}_y^T \hat{\Sigma}_{yy} \mathbf{u}_{y,i} = 0, \quad i = 1, 2, \dots, k, \quad (19.37)$$

$$\mathbf{u}_x^T \hat{\Sigma}_{xy} \mathbf{u}_{y,i} = 0, \quad \mathbf{u}_y^T \hat{\Sigma}_{yx} \mathbf{u}_{x,i} = 0, \quad i = 1, 2, \dots, k. \quad (19.38)$$

In other words, every new pair of vectors is computed so as to be normalized (19.36) and at the same time, each one to be orthogonal (in the generalized sense) to those obtained in the previous iteration steps (Eqs. (19.37) and (19.38)). Note that this guarantees that the derived canonical variates are uncorrelated to all previously derived ones. This reminds us of the uncorrelatedness property of the principle components in PCA. The only nonzero correlation in CCA, which is maximized at every iteration step, is the one between $z_{x,k} = \mathbf{u}_{x,k}^T \mathbf{x}$ and $z_{y,k} = \mathbf{u}_{y,k}^T \mathbf{y}$, $k = 1, 2, \dots, m$.

More on CCA can be found in [6, 24]. Extensions of CCA in reproducing kernel Hilbert spaces have also been developed and used; see, for example, [9, 82, 110] and the references therein. In [82], the kernel CCA is used for content-based image retrieval. The aim is to allow retrieval of images from a text query but without reference to any labeling associated with the image. The task is treated as a cross-modal problem. A probabilistic Bayesian formulation of CCA has been given in [14, 106]. A regularized CCA version, using sparsity-based arguments, has been derived in [83]. In [60], a variant of CCA is proposed, named *correlated component analysis*; instead of two directions (subspaces), a common direction is derived for both data sets. The idea behind this method is that the two data sets may not be much different, so a single direction is enough. In this way, the task has fewer free parameters to estimate. Moreover, the constraint on orthogonality is dropped, which in some cases may not be physically justifiable. A Bayesian extension of the method is provided in [138].

Example 19.3. Let $\mathbf{x} \in \mathbb{R}^2$ be a normally distributed random vector, $\mathcal{N}(\mathbf{0}, I)$. The pair of random variables, (y_1, y_2) , are related to (x_1, x_2) as

$$\mathbf{y} = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix} \mathbf{x}.$$

Note the strong correlation that exists between the involved variables, because

$$y_1 + y_2 = x_1 + x_2.$$

However, the cross-covariance matrix Σ_{yx} ,

$$\Sigma_{yx} = AI = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix},$$

indicates a rather low correlation. After performing CCA, the resulting directions are

$$\mathbf{u}_{x,1} = \mathbf{u}_{y,1} = -\frac{1}{\sqrt{2}}[1, 1]^T,$$

which actually is the direction where the linear equality of the involved variables lies. The maximum correlation coefficient value is equal to 1, indicating strong correlation indeed.

19.4.1 RELATIVES OF CCA

CCA is not the only multivariate technique to process and deal with different data sets jointly. Various techniques have been developed, using different optimizing criteria/constraints, each one serving different needs and goals.

The aim of this subsection is to briefly discuss some of these methods under a common framework. Recall that the eigenvalue-eigenvector problem for computing the pair of canonical basis vectors results

from the pair of equations in Eqs. (19.30)–(19.31). These can be combined into a single one [24], namely

$$Cu = \lambda Bu, \quad (19.39)$$

where

$$\mathbf{u} := [\mathbf{u}_x^T, \mathbf{u}_y^T]^T,$$

and

$$C := \begin{bmatrix} O & \hat{\Sigma}_{xy} \\ \hat{\Sigma}_{yx} & O \end{bmatrix}, \quad B := \begin{bmatrix} \hat{\Sigma}_{xx} & O \\ O & \hat{\Sigma}_{yy} \end{bmatrix}.$$

Changing the structure of the two matrices, C and B , different methods result. For example, if we set $C = \hat{\Sigma}_{xx}$ and $B = I$, we get the eigenvalue-eigenvector task of PCA.

In [178], algorithmic procedures for the solution of the related equations, in a numerically robust way, are discussed.

Partial least-squares

Partial least-squares (PLS) method was first introduced in [175] and it has been used extensively in a number of applications, such as chemometrics, bioinformatics, food research, medicine, pharmacology, social sciences, and physiology, to name but a few. The corresponding eigenanalysis problem results if we set in Eq. (19.39)

$$B = \begin{bmatrix} I & O \\ O & I \end{bmatrix},$$

and keep C the same as for CCA. This eigenvalue-eigenvector problem arises (try it) if instead of maximizing the correlation coefficient ρ in Eq. (19.25), one maximizes the covariance, that is,

$$\text{cov}(z_{x,1}, z_{y,1}) = \mathbb{E}[z_{x,1}z_{y,1}]. \quad (19.40)$$

This means that while trying to reduce the dimensionality, our concern not only focuses on the correlation but *at the same* we want to identify directions that *also* care for maximum variance for both sets of variables. The optimizing task for identifying the first pair of axes, $\mathbf{u}_{x,1}, \mathbf{u}_{y,1}$, now becomes

$$\text{maximize } \mathbf{u}_x^T \hat{\Sigma}_{xy} \mathbf{u}_y, \quad (19.41)$$

$$\text{s.t. } \mathbf{u}_x^T \mathbf{u}_x = 1, \quad (19.42)$$

$$\mathbf{u}_y^T \mathbf{u}_y = 1. \quad (19.43)$$

PLS has been used both for classification as well as for regression tasks. For example in Chapter 6, we used PCA for regression in order to reduce the dimensionality of the space and the LS solution was expressed in this lower dimensional space. However, the principle axes were determined only on the basis of the input data so as to retain maximum variance. In contrast, PLS can be employed by considering the output observations as the second set of variables, and one can select the axes so as to maximize the variances as well as the correlation between the two data sets. The latter can be understood from the fact that by maximizing the covariance (PLS) is equivalent to maximizing the product of the correlation coefficient (used for CCA) times the two variance terms.

The literature on PLS is extensive and the method has been studied both algorithmically and from its performance point of view. The interested reader can obtain more on PLS from [143]. In all the

techniques we have discussed so far, a major focus is on computing the eigenvalues-eigenvectors. To this end, although one can use general packages and algorithms, a number of more efficient alternatives have been derived. A common approach is to solve the task in a two-step iterative procedure. In the first step, the largest eigenvalue (eigenvector) is computed, for which there exist efficient algorithms, such as the power method (e.g., [77]). Then, a procedure known as *deflation* is adopted; this consists of removing from the covariance matrices the variance that has been explained with the features extracted from the first step; see, for example, [127]. Kernelized versions of PLS have also been proposed, for example, [9, 142].

Remarks 19.3.

- Another dimensionality reduction method results if we set in Eq. (19.39)

$$B = \begin{bmatrix} \hat{\Sigma}_{xx} & O \\ O & I \end{bmatrix}.$$

The resulting method is known as *multivariate linear regression* (MLR). This is the task of finding a set of basis vectors and corresponding regressors such that the mean-square error in a regression problem is minimized, [24].

- CCA is *invariant* with respect to affine transformations. This is an important advantage with respect to the ordinary correlation analysis, for example, [6].
- Extensions of CCA and PLS to more than two data sets have also been proposed; see, for example, [52, 103, 174].

19.5 INDEPENDENT COMPONENT ANALYSIS

The latent variable interpretation of PCA was summarized in Eqs. (19.17)–(19.19), where each one of the observed random variables, x_i , is (approximately) written as a linear combination of the latent variables (principle components in this case), z_i , which are in turn obtained via Eq. (19.19), imposing the uncorrelatedness constraint.

The kick-off point for ICA is to assume that the following latent model is true, that is,

$$\mathbf{x} = A\mathbf{s}, \quad (19.44)$$

where the (unknown) latent variables of \mathbf{s} are assumed to be mutually statistically *independent* and we refer to them as the *independent components* (ICs). The task then comprises obtaining estimates of both the matrix A as well as the independent components. We will focus on the case where A is an $l \times l$ square matrix. Extensions to fat and tall matrices, corresponding to scenarios where the number of latent variables, m , is smaller or larger than the number of the observed random variables, l , have also been considered and developed (see, e.g., [93]).

Matrix A is known as the *mixing matrix* and its elements, a_{ij} , as the *mixing coefficients*. The resulting estimates of the latent variables will be denoted as z_i , $i = 1, 2, \dots, l$, and we will also refer to them as independent components. The observed random variables, x_i , $i = 1, 2, \dots, l$, are sometimes called the *mixture variables* or simply mixtures.

To obtain the estimates of the latent variables, we adopt the model

$$\hat{\mathbf{s}} := \mathbf{z} = W\mathbf{x}, \quad (19.45)$$

where W is also known as the *unmixing* or *separating* matrix. Note that

$$\mathbf{z} = W\mathbf{As},$$

and we have to estimate the unknown parameters, so that \mathbf{z} is as close to \mathbf{s} , that is, to be independent. For square matrices, $A = W^{-1}$, assuming invertibility.

19.5.1 ICA AND GAUSSIANITY

Although in general in statistics adopting the Gaussian assumption for a pdf seems to be rather a “blessing,” in the case of ICA this is not true any more. This can easily be understood if we look at the consequences of adopting the Gaussian assumption. If the independent components follow Gaussian distributions, their joint pdf is given by

$$p(\mathbf{s}) = \frac{1}{(2\pi)^{l/2}} \exp\left(-\frac{\|\mathbf{s}\|^2}{2}\right), \quad (19.46)$$

where for simplicity, we have assumed that all the variables are normalized to unit variance. Let the mixing matrix, A , be an orthogonal one, that is, $A^{-1} = A^T$. Then, the joint pdf of the mixtures is readily obtained as (see, Eq. (2.45))

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{l/2}} \exp\left(-\frac{\|A^T \mathbf{x}\|^2}{2}\right) |\det(A^T)|. \quad (19.47)$$

However due to the orthogonality of A , we have that $\|A^T \mathbf{x}\|^2 = \|\mathbf{x}\|^2$ and $|\det(A^T)| = 1$, which makes $p(\mathbf{s})$ indistinguishable from $p(\mathbf{x})$. That is, no conclusion about A can be drawn by observing \mathbf{x} , as all related information has been lost. Seen from another point of view, the mixtures x_i are mutually uncorrelated, as $\Sigma_x = I$, and ICA can provide no further information. This is a direct consequence of the fact that uncorrelatedness for jointly Gaussian variables is equivalent to independence (see Section 2.3.2). In other words, if the latent variables are Gaussians, ICA cannot take us any further than PCA, because the latter provides uncorrelated components. That is, the mixing matrix, A , is *not identifiable* for Gaussian independent components. In a more general setting, in a case where some of the components are Gaussians and some are not, ICA can identify the non-Gaussian ones. Thus, for a matrix A to be identifiable, *at most one* of the independent components can be Gaussian.

From a mathematical point of view, the ICA task is ill-posed for Gaussian variables. Indeed, assume that a set of independent Gaussian components, \mathbf{z} , have been obtained; then, any linear transformation on \mathbf{z} by a unitary matrix will also be a solution (as shown previously). Note that this problem is bypassed in PCA, because the latter imposes a specific structure on the transformation matrix.

In order to deal with independence one has to involve, in one way or another, higher order statistical information. Second-order statistical information suffices for imposing uncorrelatedness, as is the case with PCA, but it is not enough for ICA. To this end, a large number of techniques and algorithms have been developed over the years and reviewing all these techniques is far beyond the limits imposed on a book section. The goal here is to provide the reader with the essence behind these techniques and emphasize the need to bring higher order statistics into the game. The interested reader can delve deeper in this field from [51, 54, 75, 93, 113].

19.5.2 ICA AND HIGHER ORDER CUMULANTS

Imposing the constraint on the components of \mathbf{z} to be independent is equivalent to demanding all higher order *cross-cumulants* (Appendix B.3) to be zero. One possibility to achieve this is to restrict ourselves up to the fourth-order cumulants [53]. As it is stated in Appendix B.3, the first three cumulants for zero-mean variables are equal to the corresponding moments, that is,

$$\kappa_1(z_i) = \mathbb{E}[z_i] = 0,$$

$$\kappa_2(z_i, z_j) = \mathbb{E}[z_i z_j],$$

$$\kappa_3(z_i, z_j, z_k) = \mathbb{E}[z_i z_j z_k],$$

and the fourth-order cumulants are given by

$$\begin{aligned} \kappa_4(z_i, z_j, z_k, z_r) &= \mathbb{E}[z_i z_j z_k z_r] - \mathbb{E}[z_i z_j] \mathbb{E}[z_k z_r] \\ &\quad - \mathbb{E}[z_i z_k] \mathbb{E}[z_j z_r] - \mathbb{E}[z_i z_r] \mathbb{E}[z_j z_k]. \end{aligned}$$

An assumption that is employed is that the involved pdfs are symmetric, which renders odd order cumulants to zero. Thus, we are left only with the second- and fourth-order cumulants. Under the previous assumptions, our goal is to estimate the unmixing matrix, W , so that (a) the second-order and (b) the fourth-order cumulants to become zero. This is achieved in two steps.

Step 1: Compute

$$\hat{\mathbf{z}} = U^T \mathbf{x}, \quad (19.48)$$

where U is the unitary $l \times l$ matrix associated with PCA. This transformation guarantees that the components of $\hat{\mathbf{z}}$ are uncorrelated, that is,

$$\mathbb{E}[\hat{z}_i \hat{z}_j] = 0, \quad i \neq j, \quad i, j = 1, 2, \dots, l.$$

Step 2: Compute a orthogonal matrix, \hat{U} , such that the fourth-order cross-cumulants of the components of the transformed random vector,

$$\mathbf{z} = \hat{U}^T \hat{\mathbf{z}}, \quad (19.49)$$

are zero. In order to achieve this, the following maximization task is solved:

$$\max_{\hat{U} \hat{U}^T = I} \sum_{i=1}^l \kappa_4^2(z_i). \quad (19.50)$$

Step 2 is justified as follows. It can be shown [53] that, the sum of the squares of the fourth-order cumulants is invariant under a linear transformation by an orthogonal matrix. Therefore, as the sum of the squares of the fourth-order cumulants is fixed for \mathbf{z} , maximizing the sum of the squares of the autocumulants of \mathbf{z} will force the corresponding cross-cumulants to zero. Observe that this is basically a diagonalization problem of the fourth-order cumulant multidimensional array. In practice, this can be achieved by generalizing the method of Givens rotations, used for matrix diagonalization, [53]. Note that the sum that is maximized is a function of (a) the elements of the unknown matrix \hat{U} , (b) the elements of the known (for this step) matrix U , and (c) the cumulants of the random components of the mixtures \mathbf{x} , which have to be estimated prior to the application of the method. In practice, it usually turns

out that setting the cross-cumulants to zero is only approximately achieved. This is because the model in Eq. (19.44) may not be exact, for example, due to the existence of noise. Also, the cumulants of the mixtures are only approximately known, because they are estimated by the available observations.

Once U and \hat{U} have been computed, the unmixing matrix is readily available and we can write

$$\mathbf{z} = W\mathbf{x} = (U\hat{U})^T\mathbf{x},$$

and the mixing matrix is given as $A = W^{-1}$.

A number of algorithms have been developed around the idea of higher order cumulants, which are also known as *tensorial methods*. Tensors are generalizations of matrices and cumulant tensors are generalizations of the covariance matrix. Moreover, note that as the eigenanalysis of the covariance matrix leads to uncorrelated (principle) components, the eigenanalysis of the cumulant tensor leads to independent components. The interested reader can obtain a more detailed account of such techniques from [38, 53, 112].

ICA ambiguities

Any ICA method can (approximately) recover the independent components within the following two indeterminacies.

- Independent components (ICs) are recovered to within a constant factor. Indeed, if A and \mathbf{z} are the recovered quantities by an ICA algorithm, then $(1/a)A$ and $a\mathbf{z}$ is also a solution, as is readily seen from Eq. (19.44). Thus, usually the recovered latent variables (ICs) are normalized to unit variance.
- We cannot determine the order of the ICs. Indeed, if A and \mathbf{z} have been recovered and P is a permutation matrix, then AP^{-1} and $P\mathbf{z}$ is also a solution, because the components of $P\mathbf{z}$ are the same as those of \mathbf{z} in a different order (with the same statistical properties).

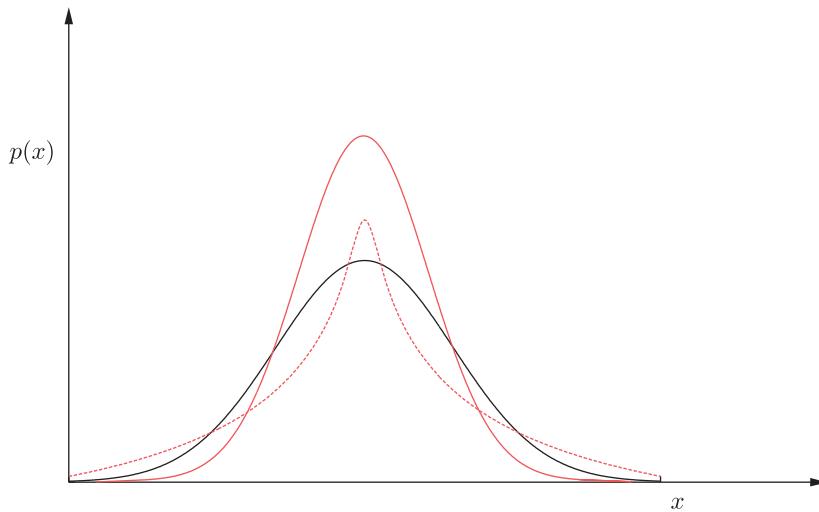
19.5.3 NON-GAUSSIANITY AND INDEPENDENT COMPONENTS

The fourth-order (auto)cumulant, of a random variable, z ,

$$\kappa_4(z) = \mathbb{E}[z^4] - 3(\mathbb{E}[z^2])^2,$$

is known as the *kurtosis* of the variable and it is a measure of *non-Gaussianity*. Variables following the Gaussian distribution have zero kurtosis. Sub-Gaussian variables (variables whose pdf falls at a slower rate than the Gaussian, for the same variance) have negative kurtosis. Super-Gaussian variables (corresponding to pdfs that fall at a faster rate than the Gaussian) have positive kurtosis. Thus, if we keep the variance fixed (e.g., for variables normalized to unit variance), maximizing the sum of squared kurtosis, it results in maximizing the nonGaussianity of the recovered ICs. Usually, the absolute value of the kurtosis of the recovered ICs is used as a measure of ranking them. This is important if ICA is used as a feature generation technique. Figure 19.8 shows some typical examples of a sub-Gaussian and a super-Gaussian together with the corresponding Gaussian distribution. Also, another typical example of a sub-Gaussian distribution is the uniform one.

Recall from Chapter 12 (Section 12.4.1) that the Gaussian distribution is the one that maximizes the entropy under the variance and mean constraints. In other words, it is the most random one, under these

**FIGURE 19.8**

A Gaussian (full gray line) a super-Gaussian (dotted red line) and a sub-Gaussian (full red line).

constraints, and from this point of view the least informative with respect to the underlying structure of the data. In contrast, distributions that have the least resemblance to the Gaussian are more interesting as they are able to better unveil the structure associated with the data. This observation is at the heart of *projection pursuit*, which is closely related to the ICA family of techniques. The essence of these techniques is to search for directions in the feature space where the data projections are described in terms of non-Gaussian distributions [90, 99].

19.5.4 ICA BASED ON MUTUAL INFORMATION

The approach based on zeroing the second- and fourth-order cross-cumulants is not the only one. An alternative path is to estimate W by minimizing the *mutual information* among the latent variables. The notion of mutual information was introduced in Section 2.5. Elaborating a bit on Eq. (2.158) and performing the integrations on the right-hand side (for the case of more than two variables), it is readily shown that

$$I(\mathbf{z}) = -H(\mathbf{z}) + \sum_{i=1}^l H(z_i), \quad (19.51)$$

where $H(z_i)$ is the associated entropy of z_i , defined in Eq. (2.157). In Section 2.5 it has been shown that, $I(\mathbf{z})$ is equal to the Kullback-Leibler (KL) divergence between the joint pdf $p(\mathbf{z})$ and the product of the respective marginal probability densities, $\prod_{i=1}^l p_i(z_i)$. The KL divergence (and, hence, the associated mutual information $I(\mathbf{z})$) is a nonnegative quantity and it becomes zero if the components z_i are statistically independent. This is because only in this case the joint pdf becomes equal to the product of the corresponding marginal pdfs, leading the KL divergence to zero. Hence, the idea now becomes

to compute W so as to force $I(\mathbf{z})$ to be minimum, as this will make the components of \mathbf{z} as independent as possible. Plugging Eq. (19.45) into Eq. (19.51) and taking into account the formula that relates the two pdfs associated with \mathbf{x} and \mathbf{z} (Eq. (2.45)), we end up with

$$I(\mathbf{z}) = -H(\mathbf{x}) - \ln |\det(W)| - \sum_{i=1}^l \int p_i(z_i) \ln p_i(z_i) dz_i. \quad (19.52)$$

The elements of the unknown matrix, W , are also hidden in the marginal pdfs of the latent variables, z_i . However, it is not easy to express this dependence explicitly. One possibility is to expand each one of the marginal densities around the Gaussian pdf, denoted here as $g(z)$, following Edgeworth's expansion (Appendix B), and truncate the series to a reasonable approximation. For example, keeping the first two terms in the Edgeworth expansion we have

$$p_i(z_i) = g(z_i) \left(1 + \frac{1}{3!} \kappa_3(z_i) H_3(z_i) + \frac{1}{4!} \kappa_4(z_i) H_4(z_i) \right), \quad (19.53)$$

where $H_k(z_i)$ is the Hermite polynomial of order k (Appendix B). To obtain an approximate expression for $I(\mathbf{z})$, in terms of cumulants of z_i and W , we can (a) insert in Eq. (19.52) the pdf approximation in Eq. (19.53), (b) adopt the approximation $\ln(1+y) \simeq y - y^2$, and (c) perform the integrations. This is no doubt a rather painful task! For the case of Eq. (19.53) and constraining W to be orthogonal the following is obtained (e.g., [93]):

$$I(\mathbf{z}) \approx C - \sum_{i=1}^l \left(\frac{1}{12} \kappa_3^2(z_i) + \frac{1}{48} \kappa_4^2(z_i) + \frac{7}{48} \kappa_4^4(z_i) - \frac{1}{8} \kappa_3^2(z_i) \kappa_4(z_i) \right), \quad (19.54)$$

where C is a quantity independent of W . Under the assumption that the pdfs are symmetric (thus, third-order cumulants are zero), it can be shown that minimizing the approximate expression of the mutual information in Eq. (19.54) is equivalent to maximizing the sum of the squares of the fourth-order cumulants. Note that the orthogonal W constraint is not necessary, and if it is not adopted other approximate expressions for $I(\mathbf{z})$ result, for example, [84].

Minimization of $I(\mathbf{z})$ in Eq. (19.54) can be carried out by a gradient descent technique (Chapter 5), where the involved expectations (associated with the cumulants) are replaced by the respective instantaneous values. Although we will not treat the derivation of algorithmic schemes in detail, in order to get a flavor of the involved tricks, let us go back to Eq. (19.52), before we apply the approximations. Because $H(\mathbf{x})$ does not depend on W , minimizing $I(\mathbf{z})$ is equivalent with the maximization of

$$J(W) = \ln |\det(W)| + \mathbb{E} \left[\sum_{i=1}^l \ln p_i(z_i) \right]. \quad (19.55)$$

Taking the gradient of the cost function with respect to W results in

$$\frac{\partial J(W)}{\partial W} = W^{-T} - \mathbb{E}[\boldsymbol{\phi}(\mathbf{z}) \mathbf{x}^T], \quad (19.56)$$

where

$$\boldsymbol{\phi}(\mathbf{z}) := \left[-\frac{p'_1(z_1)}{p_1(z_1)}, \dots, -\frac{p'_l(z_l)}{p_l(z_l)} \right]^T, \quad (19.57)$$

and

$$p'_i(z_i) := \frac{dp_i(z_i)}{dz_i}, \quad (19.58)$$

and we used the formula

$$\frac{\partial \det(W)}{\partial W} = W^{-T} \det(W).$$

Obviously, the derivatives of the marginal probability densities depend on the type of approximation adopted in each case. The general gradient ascent scheme at the i th iteration step can now be written as

$$W^{(i)} = W^{(i-1)} + \mu_i \left((W^{(i-1)})^{-T} - \mathbb{E} [\phi(\mathbf{z}) \mathbf{x}^T] \right),$$

or

$$W^{(i)} = W^{(i-1)} + \mu_i \left(I - \mathbb{E} [\phi(\mathbf{z}) \mathbf{z}^T] \right) (W^{(i-1)})^{-T}. \quad (19.59)$$

In practice, the expectation operator is neglected and random variables are replaced by respective observations, in the spirit of the stochastic approximation rationale ([Chapter 5](#)).

The update equation in Eq. (19.59) involves the inversion of the transpose of the current estimate of W . Besides the computational complexity issues, there is no guarantee of the invertibility in the process of adaptation. The use of the so called *natural gradient* [63], instead of the gradient in Eq. (19.56), results in

$$W^{(i)} = W^{(i-1)} + \mu_i \left(I - \mathbb{E} [\phi(\mathbf{z}) \mathbf{z}^T] \right) W^{(i-1)}, \quad (19.60)$$

which does not involve matrix inversion and at the same time improves convergence. A more detailed treatment of this issue is beyond the scope of this book. Just to give an incentive to the mathematically inclined reader for indulging more deeply this field, it suffices to say that our familiar gradient, that is, Eq. (19.56), points to the steepest ascent direction if the space is Euclidean. However, in our case the parameter space consists of all the nonsingular $l \times l$ matrices, which is a multiplicative group. The space is Riemannian and it turns out that the natural gradient, pointing to the steepest ascent direction, results if we multiply the gradient in Eq. (19.56) by $W^T W$, which is the corresponding Riemannian metric tensor [63].

Remarks 19.4.

- From the gradient in Eq. (19.56), it is easy to see that at a stationary point the following is true:

$$\frac{\partial J(W)}{\partial W} W^T = \mathbb{E}[I - \phi(\mathbf{z}) \mathbf{z}^T] = 0. \quad (19.61)$$

In other words, what we achieve with ICA is a *nonlinear generalization* of PCA. Recall that for the latter, the uncorrelatedness condition can be written as

$$\mathbb{E}[I - \mathbf{z} \mathbf{z}^T] = 0. \quad (19.62)$$

The presence of the *nonlinear* function, ϕ , takes us beyond simple uncorrelatedness, and brings the cumulants into the scene. As a matter of fact, Eq. (19.61) was the one that inspired the early pioneering work on ICA, as a direct nonlinear generalization of PCA, [86, 100].

- The origins of ICA are traced back to the seminal paper [86]. For a number of years, it remained an activity pretty much within the French signal processing and statistics communities. Two papers were catalytic for its widespread use and popularity, namely [17] in the mid-nineties and the development of the FastICA,⁴ [92], which allowed for efficient implementations; see [101] for a related review.
- In machine learning, the use of ICA as a feature generation technique is justified by the following argument. In [16], it is suggested that the outcome of the early processing performed by the visual cortical feature detectors might be the result of a *redundancy reduction* process. Thus, searching for independent features, conditioned on the input data, is in line with such a claim; see, for example, [70, 107] and the references therein.
- Although we have focused on the noiseless case, extensions of ICA to noisy tasks have also been proposed (see, e.g., [93]). For an extension of ICA in the complex-valued case, see [2]. Nonlinear extensions have also been considered, including kernelized ICA versions, for example, [13].
- In [3], the treatment of ICA also involves random processes and a wider class of signals, including Gaussians, can be identified.
- In [7], the multiset ICA framework of *independent vector analysis* (IVA) is discussed. It is shown that it generalizes the multiset CCA, if higher-order, besides second-order statistics, are taken into account.

19.5.5 ALTERNATIVE PATHS TO ICA

Besides the previously discussed two paths to ICA, a number of alternatives have been suggested, shedding light on different aspects of the problem. Some notable directions are

- *Infomax Principle*: This method assumes that the latent variables are the outputs of a nonlinear system (neural network, Chapter 18) of the form

$$z_i = \phi_i(\mathbf{w}_i^T \mathbf{x}) + \eta, \quad i = 1, 2, \dots, l,$$

where ϕ_i are nonlinear functions and η additive Gaussian noise. The weight vectors, \mathbf{w}_i , are computed so as to maximize the entropy of the outputs; the reasoning is based on some information theoretic arguments concerning the information flow in the network, [17].

- *Maximum Likelihood*: Starting from Eq. (19.44), the pdf of the observed variables is expressed in terms of the pdfs of the independent components

$$p(\mathbf{x}) = |\det(W)| \prod_{i=1}^l p_i(\mathbf{w}_i^T \mathbf{x}_i),$$

where we used

$$W := A^{-1}.$$

Assuming that we have N observations, $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, and taking the logarithm of the joint $p(\mathbf{x}_1, \dots, \mathbf{x}_N)$, one can maximize the log-likelihood with respect to the W . It is straightforward to derive the log-likelihood function and to observe that it is very similar to $J(W)$ given in Eq. (19.55). The p_i 's are chosen so as to belong to families of non-Gaussians, for example, [93]. A connection between the infomax approach and the maximum likelihood one has been established in [36, 37].

⁴ <http://research.ics.aalto.fi/ica/fastica/index.shtml>.

- *Negentropy*: According to this method, the starting point is to maximize the non-Gaussianity, which is now measured in terms of the *negentropy*, defined as

$$J(\mathbf{z}) := H(\mathbf{z}_{\text{gauss}}) - H(\mathbf{z}),$$

where $\mathbf{z}_{\text{gauss}}$ corresponds to Gaussian distributed variables of the same covariance matrix, which we know corresponds to the maximum entropy, H . Thus, maximizing the negentropy, which is a nonnegative function, is equivalent to making the latent variables as less Gaussian as possible. Usually, approximations of the negentropy are employed, which are expressed in terms of higher order cumulants, or by matching the nonlinearity to source distribution, [93, 132].

- If the unmixing matrix is constrained to be orthogonal, the negentropy and the maximum likelihood approaches become equivalent, [2].

The cocktail party problem

A classical application that demonstrates the power of the ICA is the so-called *cocktail party problem*. In a party, there are various people speaking; in our case, we are going to consider music as well. Let us say that there are people (a female and a male) and there is also monophonic music, making three sources of sound in total. Then, three microphones (as many as the sources) are placed in different places in the room and the mixed speech signals are recorded. We denote the inputs to the three microphones as $x_1(t)$, $x_2(t)$, $x_3(t)$, respectively. In the simplest of the models, the three recorded signals can be considered as linear combinations of the individual source signals. Delays are not considered. The goal is to use ICA and recover the original speech and music from the recorded mixed signals.

To this end and in order to bring the task in the formulation we have previously adopted, we consider the values of the three signals at different time instants as different observations of the corresponding random variables, x_1 , x_2 , x_3 , which are put together to form the random vector \mathbf{x} . We further adopt the very reasonable assumption that the original source signals, denoted as $s_1(t)$, $s_2(t)$, $s_3(t)$, are independent and (similarly as before) the values at different time instants correspond to the values of three latent variables, denoted together as a random vector \mathbf{s} .

We are ready now to apply ICA to compute the unmixing matrix W , from which we can obtain the estimates of the ICs corresponding to the observations received by the three microphones,

$$\mathbf{z}(t) = [z_1(t), z_2(t), z_3(t)]^T = W[x_1(t), x_2(t), x_3(t)]^T.$$

Figure 19.9a shows the three different signals, which are linearly combined (by a set of mixing coefficients defining a mixing matrix A) to form the three “microphone signals.” Figure 19.9b shows the resulting signals, which are then used as described before for the ICA analysis. Figure 19.9c shows the recovered original signals, as the corresponding ICs. The FastICA algorithm was employed.⁵ Figure 19.10 is the result when PCA is used and the original signals are obtained via the (three) principle components.

One can observe that ICA manages to separate the signals with very good accuracy, whereas PCA fails. The reader can also listen to the signals by downloading the corresponding “.wav” files from the site of this book.

Note that the cocktail party problem is representative of a large class of tasks where a number of recorder signals result as linear combinations of other independent signals; the goal is the recovery of the latter. A notable application of this kind is found in electroencephalogram (EEG). The EEG data

⁵ <http://research.ics.aalto.fi/ica/fastica/>.

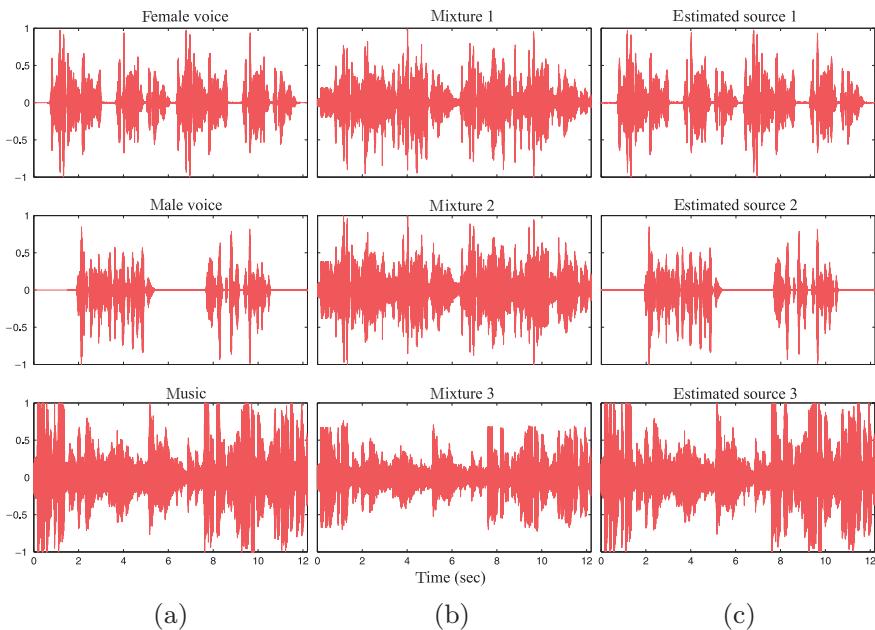


FIGURE 19.9

ICA source separation in the cocktail party setting.

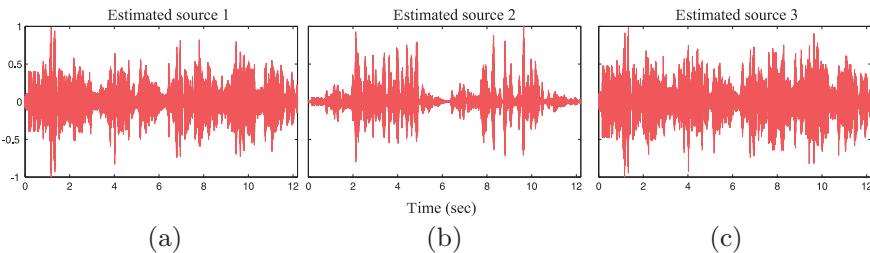


FIGURE 19.10

PCA source separation in the cocktail party setting.

consists of electrical potentials recorded at different locations on the scalp (or more recently in the ear, [105]), which are generated by the combination of different underlying components of brain and muscle activity. The task is to use ICA to recover the components, which in turn can unveil useful information about the brain activity, for example, [148].

The cocktail party problem is a typical example of a more general class of tasks known as *blind source separation* (BSS). The goal in these tasks is to estimate the “causes” (sources, original signals) based only on information residing in the observations, without any other extra information, and this is the reason that the word “blind” is used. Viewed in another way, BSS is an example of unsupervised learning. ICA is, probably, the most widely used technique for such problems.

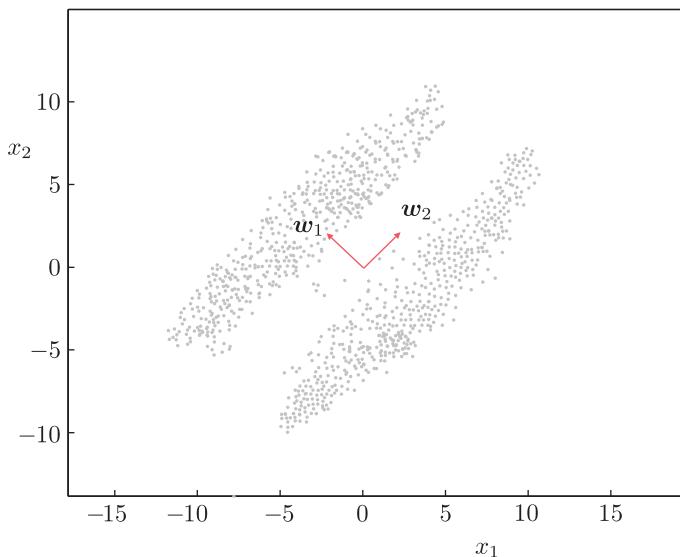


FIGURE 19.11

The setup for the ICA simulation example. The two vectors point to the projection directions resulting from the analysis. The optimal direction for projection, resulting from the ICA analysis, is that of w_2 .

Example 19.4. The goal of this example is to demonstrate the power of ICA as a feature generation technique, where the most informative of the generated features are to be kept.

The example is a realization of the case shown in Figure 19.11. A number of 1024 samples of a two-dimensional normal distribution was generated.

The mean and covariance matrix of the normal pdf were

$$\boldsymbol{\mu} = [-2.6042, 2.5]^T, \quad \Sigma = \begin{bmatrix} 10.5246 & 9.6313 \\ 9.6313 & 11.3203 \end{bmatrix}$$

Similarly, 1024 samples from a second normal pdf were generated with the same covariance matrix and mean $-\boldsymbol{\mu}$. For the ICA, the method based on the second- and fourth-order cumulants, presented in this section, was used. The resulting transformation matrix W is

$$W = \begin{bmatrix} -0.7088 & 0.7054 \\ 0.7054 & 0.7088 \end{bmatrix} := \begin{bmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \end{bmatrix}$$

The vectors \mathbf{w}_2 and \mathbf{w}_1 point in the principle and minor axis directions, respectively, obtained from the PCA analysis. According to PCA, the most informative direction is along the principle axis \mathbf{w}_2 , which is the one with maximum variance. However, the most interesting direction for projection, according to the ICA analysis, is that of \mathbf{w}_1 . Indeed, the kurtosis of the obtained ICs z_1, z_2 , along these directions are

$$\kappa_4(z_1) = -1.7,$$

$$\kappa_4(z_2) = 0.1,$$

respectively. Thus, projection in the principle (PCA) axis direction results in a variable with a pdf close to a Gaussian. The projection on the minor axis direction results in a variable with a pdf that deviates from the Gaussian (it is bimodal) and it is the more interesting one from the classification point of view. This can be easily verified by looking at the figure; projecting on the direction w_2 leads to class overlapping.

19.6 DICTIONARY LEARNING: THE k -SVD ALGORITHM

The concept of *overcomplete dictionaries* and their importance in modeling real-world signals has been introduced in [Chapter 9](#). We return to this topic, this time in a more general setting. There, the dictionary was assumed known with preselected atoms. In this section, the blind version of this task is considered; that is, the atoms of the dictionary are unknown and have to be estimated from the observed data. Recall that, this was the case with ICA; however, instead of the independence concept, used for ICA, sparsity arguments will be mobilized here. Giving the freedom to the dictionary to adapt to the needs of the specific, each time, input can lead to enhanced performance compared to dictionaries with preselected atoms.

Our starting point is that the observed l random variables are expressed in terms of $m > l$ latent ones according to the linear model

$$\mathbf{x} = A\mathbf{z}, \quad \mathbf{x} \in \mathbb{R}^l, \quad \mathbf{z} \in \mathbb{R}^m, \quad (19.63)$$

and A is an unknown $l \times m$ matrix. Usually, $m \gg l$. Even if A were known and fixed, it does not need special mathematical skills to see that this task has not a single solution and one has to embed constraints into the problem. To this end, we are going to adopt sparsity-promoting constraints, as we have already discussed in various parts in this book.

Let \mathbf{x}_n , $n = 1, 2, \dots, N$, be the observations that will constitute the only available information. The task is to obtain the atoms (columns of A) of the dictionary as well as the latent variables that are assumed to be sparse; that is, we are going to establish a sparse representation of our input observations (vectors). No doubt, there are different paths to achieve the goal. We are going to focus on one of the most widely known and used methods, known as k -SVD, proposed in [4].

Let $X := [\mathbf{x}_1, \dots, \mathbf{x}_N]$, $A := \{\mathbf{a}_1, \dots, \mathbf{a}_m\}$, and $Z := [\mathbf{z}_1, \dots, \mathbf{z}_N]$, where \mathbf{z}_n is the latent vector corresponding to the input \mathbf{x}_n , $n = 1, 2, \dots, N$. The dictionary learning (DL) task is cast as the following optimization problem

$$\text{minimize with respect to } A, Z \quad \|X - AZ\|_F^2, \quad (19.64)$$

$$\text{subject to} \quad \|\mathbf{z}_n\|_0 \leq T_0, \quad n = 1, 2, \dots, N, \quad (19.65)$$

where T_0 is a threshold value. This is a nonconvex optimization task, and it is performed iteratively; each iteration comprises two stages. In the first one, A is assumed to be fixed and optimization is carried out with respect to \mathbf{z}_n , $n = 1, 2, \dots, N$. In the second stage, the latent vectors are assumed fixed and optimization is carried out with respect to the columns of A .

In k -SVD, a slightly different rationale is adopted. While optimizing with respect to the columns of A , one at a time, an update of some of the elements of Z is also performed. This is a crucial difference of

the k -SVD, compared to the more standard optimization techniques, which appears to lead to improved performance in practice.

Stage 1: Assume A to be known and fixed to the value obtained from the previous iteration. Then, the associated optimization task becomes

$$\begin{aligned} \min_Z \quad & ||X - AZ||_F^2, \\ \text{s.t.} \quad & ||z_n||_0 \leq T_0, \quad n = 1, 2, \dots, N, \end{aligned}$$

which, due to the definition of the Frobenius norm, is equivalent to solving N distinct optimization tasks,

$$\min_{z_n} \quad ||x_n - Az_n||^2, \quad (19.66)$$

$$\text{s.t.} \quad ||z_n||_0 \leq T_0, \quad n = 1, 2, \dots, N. \quad (19.67)$$

A similar objective is met if the following optimization tasks are considered instead,

$$\begin{aligned} \min_{z_n} \quad & ||z_n||_0, \\ \text{s.t.} \quad & ||x_n - Az_n||^2 < \epsilon, \quad n = 1, 2, \dots, N, \end{aligned}$$

where ϵ is a constant acting as an upper bound of the error.

The task in Eqs. (19.66)–(19.67) can be solved by any one of the ℓ_0 minimization solvers, which have been considered in [Chapter 10](#), for example, the OMP. This stage is known as *sparse coding*.

Stage 2: This stage is known as the *codebook update*. Having obtained z_n , $n = 1, 2, \dots, N$, (for fixed A) from stage 1, the goal now is to optimize with respect to the columns of A . This is achieved on a *column-by-column* basis. Assume that we currently consider the update of a_k ; this is carried out so as to minimize the (squared) Frobenius norm, $||X - AZ||_F^2$. To this end, we can write the product AZ as a sum of rank-one matrices, that is,

$$AZ = [a_1, \dots, a_m][z_1^r, \dots, z_m^r]^T = \sum_{i=1}^m a_i z_i^{rT}, \quad (19.68)$$

where z_i^{rT} , $i = 1, 2, \dots, m$, are the *rows* of Z . Note that in the above sum, the vectors for indices, $i = 1, 2, \dots, k-1$, are fixed to their recently updated values during this second stage of the current iteration step, while and vectors corresponding to $i = k+1, \dots, m$, are fixed to the values that are available from the previous iteration step. This strategy allows for the use of the most recent updated information. We will now minimize with respect to the rank-one outer product matrix, $a_k z_k^{rT}$. Observe that this product, besides the k th column of A , also involves the k th row of Z ; both of them will be updated. The rank-one matrix is estimated so as to minimize

$$||E_k - a_k z_k^{rT}||_F^2, \quad (19.69)$$

where,

$$E_k := X - \sum_{i=1, i \neq k}^m a_i z_i^{rT}.$$

In other words, we seek to find the best, in the Frobenius sense, rank-one approximation of E_k . Recall from [Chapter 6 \(Section 6.4\)](#) that the solution is given via the SVD of E_k . However, if we do that, there

is no guarantee that whatever sparse structure has been embedded in \mathbf{z}_k^r , from the update in stage 1, will be retained. According to the k -SVD, this is bypassed by focusing on the active set, that is, involving only the nonzero of its coefficients. Thus, we first search for the locations of the nonzero coefficients in \mathbf{z}_k^r and let

$$\omega_k := \left\{ j_k, 1 \leq j_k \leq N : z_k^r(j_k) \neq 0 \right\}.$$

Then, we form the reduced vector $\tilde{\mathbf{z}}_k^r \in \mathbb{R}^{|\omega_k|}$, where $|\omega_k|$ denotes the cardinality of ω_k , which contains only the nonzero elements of \mathbf{z}_k^r . A little thought reveals that when writing $X = AZ$, the column of current interest, \mathbf{a}_k , contributes (as part of the corresponding linear combination) only to the columns \mathbf{x}_{j_k} , $j_k \in \omega_k$, of X . We then collect the corresponding columns of E_k to construct a reduced order matrix, \tilde{E}_k , which comprises the columns that are associated with the locations of the nonzero elements of \mathbf{z}_k^r , and select $\mathbf{a}_k \tilde{\mathbf{z}}_k^{rT}$ so that to minimize

$$\|\tilde{E}_k - \mathbf{a}_k \tilde{\mathbf{z}}_k^{rT}\|_F^2. \quad (19.70)$$

Performing SVD, $\tilde{E}_k = UDV^T$, \mathbf{a}_k is set equal to \mathbf{u}_1 corresponding to the largest of the singular values and $\tilde{\mathbf{z}}_k^r = D(1, 1)\mathbf{v}_1$. Thus, the atoms of the dictionary are obtained in normalized form (recall from the theory of SVD, that $\|\mathbf{u}_1\| = 1$). In the sequel, the updated values obtained for $\tilde{\mathbf{z}}_k^r$ are placed in the corresponding locations in \mathbf{z}_k^r . The latter now has at least as many zeros as it had before, as some of the elements in \mathbf{v}_1 may be zeros. Simple arguments (Problem 19.3) show that at each iteration the error decreases and the algorithm converges to a local minimum. The success of the algorithm depends on the ability of the greedy algorithm to provide a sparse solution during the first stage. As we know from Chapter 10, greedy algorithms work well for sparsity levels, T_0 , small enough compared to l .

In summary, each iteration step of the k -SVD algorithm comprises the following computation steps.

- Initialize $A^{(0)}$ with columns normalized to unit ℓ_2 norm.
- Set $i = 1$.
- *Stage 1:* Solve the optimization task in Eqs. (19.66)–(19.67) to obtain the sparse coding representation vectors, \mathbf{z}_n , $n = 1, 2, \dots, N$; use any algorithm developed for this task.
- *Stage 2:* For any column, $k = 1, 2, \dots, m$, in $A^{(i-1)}$, update it according to the following:
 - Identify the locations of the nonzero elements in the k th row of the computed, from stage 1, matrix Z .
 - Select the columns in X , which correspond to the locations of the nonzero elements of the k th row of Z and form a reduced order error matrix, \tilde{E}_k .
 - Perform SVD on \tilde{E}_k : $\tilde{E}_k = UDV^T$.
 - Update the k th column of $A^{(i)}$ to be the eigenvector corresponding to the largest singular value, $\mathbf{a}_k^{(i)} = \mathbf{u}_1$.
 - Update Z , by embedding in the nonzero locations of its k th row, the values $D(1, 1)\mathbf{v}_1^T$.
- Stop if a convergence criterion is met.
- If not, $i = i + 1$, and continue.

Why the name k -SVD

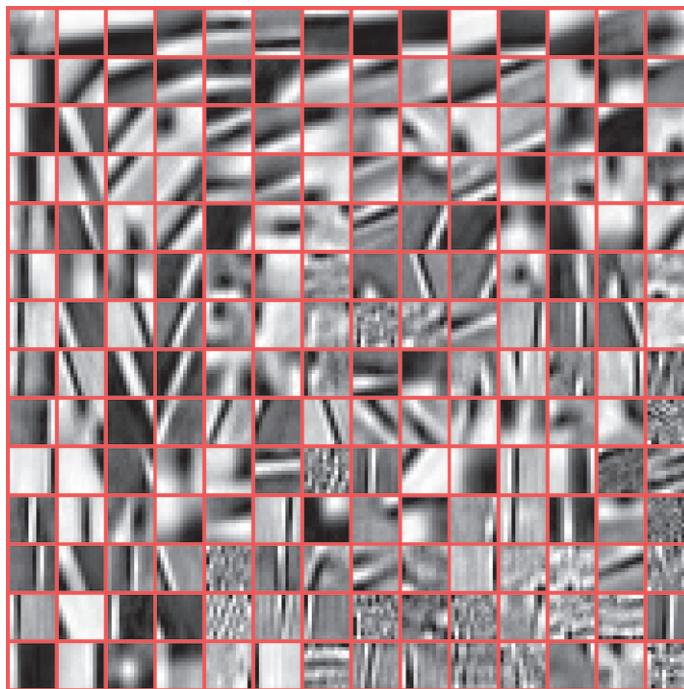
The SVD part of the name is pretty obvious. However, the reader may wonder about the presence of “ k ” in front. As stated in [4], the algorithm can be considered a generalization of the k -means algorithm,

introduced in [Chapter 12](#) (Algorithm 12.1). There, we can consider the mean values, which represent each cluster, as the code words (atoms) of a dictionary. During the first stage of the k -means learning, given the representatives of each cluster, a sparse coding scheme is performed; that is, each input vector is assigned to a single cluster. Thus, we can think of the k -means clustering as a sparse coding scheme, that associates a latent vector with each one of the observations. Note that each of the latent vectors has only one nonzero element, pointing to the cluster where the respective input vector is assigned, according to the smallest Euclidean distance from all cluster representatives. This is a major difference with the k -SVD dictionary learning, during which each observation vector can be associated with more than one atoms; hence, the sparsity level of the corresponding latent vector can be larger than one. Furthermore, based on the assignment of the input vectors to the clusters, in the second stage of the k -means algorithm, an update of the cluster representatives is performed, and for each representative only the input vectors assigned to it are used. This is also similar in spirit to what happens in the second stage of the k -SVD. The difference is that each input observation may be associated with more than one atom. As it is pointed out in [4], if one sets $T_0 = 1$, the k -means algorithm can result from the k -SVD.

Remarks 19.5.

- Alternative to k -SVD paths to dictionary learning have also been suggested. For example, in [68] a DL technique referred to as method of optimal directions (MOD) was proposed, which differs from k -SVD in the dictionary update step. In particular, the full dictionary is updated via direct minimization of the Frobenius norm. Moreover, in [185] a majorization approach is followed, which allows the incorporation of more general sparsity constraints. On the other hand, in [119, 134] probabilistic arguments are employed, using a Laplacian prior to enforce sparsity. We know from [Chapter 13 \(Section 13.5\)](#) that in this case, the involved integrations are not analytically tractable and the different methods differ in the different approximations used to bypass this obstacle. In the former, the maximum value of the integrand is used and in the latter a Gaussian approximation of the posterior is adopted in order to handle the integration. In [76], variational bound techniques are mobilized, see [Section 13.9](#).
- The method proposed in [118] bears some similarities to the k -SVD, because it also revolves around the SVD, but the dictionary is constrained to be a union of orthonormal bases. This can lead to some computational advantages; on the other hand, k -SVD puts no constraints on the atoms of the dictionary, which gives more freedom in modeling the input. Another difference lies in the column-by-column update introduced in k -SVD.
- A more detailed comparative study of k -SVD with other methods is given in [4].
- Dictionary learning is essentially a matrix factorization problem where a certain type of constraint is imposed on the right matrix factor. This approach can be considered to be just a manifestation of a wider class of constrained matrix factorization methods that allow several types of constraints to hold. Such techniques include the regularized PCA where functional and/or sparsity constraints are imposed to the left and to the right factors, [12, 179, 189], as well as the structured sparse matrix factorization in [28] together with its online counterpart [131].

Example 19.5. The goal of this example is to show the performance of the DL technique in the context of the image denoising task. In the case study of [Section 9.10](#), image denoising, based on a predetermined and fixed DCT dictionary, was considered. Here, the k -SVD will be employed in order to learn the dictionary using information of the image *itself*. The two (256×256) images, without and

**FIGURE 19.12**

Dictionary resulting from k -SVD.

with noise corresponding to $\text{PSNR} = 22$, are shown in Figures 19.13a,b, respectively. The noisy image is divided in overlapped patches of size 12×12 (144), resulting in $(256 - 12 + 1)^2 = 60,025$ patches in total; these will constitute the training data set used for the learning of the dictionary. Specifically, the patches are sequentially extracted from the noisy image, then vectorized in lexicographic order, and used as columns, one after the other to define the $(144 \times 60,025)$ matrix X . Then, k -SVD is mobilized in order to train an overcomplete dictionary of size 144×196 . The resulting atoms, reshaped in order to form 12×12 pixel patches, are shown in Figure 19.12. Compare the atoms of this dictionary with atoms of the fixed DCT dictionary of Figure 9.14.

Next, we follow the same procedure as in Section 9.10, by replacing the DCT dictionary with the one obtained by the k -SVD method. The resulting denoised image is shown in Figure 19.13c. Note that, although the dictionary was trained based on *the noisy data*, it led to about 2dB PSNR improvement over the fixed-dictionary case. As a matter of fact, because the number of patches is large and each one of them is carrying a different noise realization, the noise, during the dictionary learning stage, is averaged out leading to nearly noise-free dictionary atoms. More advanced use of dictionary learning techniques to further improve performance in tasks such as denoising and inpainting can be found in [66, 67, 130].

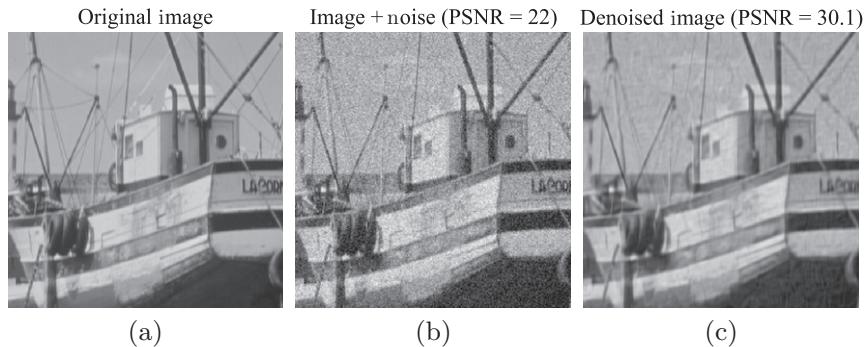
**FIGURE 19.13**

Image de-noising based on dictionary learning.

19.7 NONNEGATIVE MATRIX FACTORIZATION

The strong connection between dimensionality reduction and low-rank matrix factorization has already been stressed while discussing PCA. ICA can also be considered as a low-rank matrix factorization, if a smaller number, compared to the l observed random variables, of independent components is retained (e.g., selecting the $m < l$ least Gaussian ones).

An alternative to the previously discussed low-rank matrix factorization schemes was suggested in [135, 136], which guarantees the *nonnegativity* of the elements of the resulting matrix factors. Such a constraint is enforced in certain applications because negative elements contradict physical reality. For example, in image analysis, the intensity values of the pixels cannot be negative. Also, probability values cannot be negative. The resulting factorization is known as *nonnegative matrix factorization* (NMF) and it has been used successfully in a number of applications including document clustering [181], molecular pattern discovery [26], image analysis [115], clustering [161], music transcription and music instrument classification [19, 156], and face verification [187].

Given an $l \times N$ matrix X , the task of NMF consists of finding an approximate factorization of X , that is,

$$X \approx AZ, \quad (19.71)$$

where A and Z are $l \times m$ and $m \times N$ matrices, respectively, $m \leq \min(N, l)$ and all the matrix elements are nonnegative, that is, $A(i, k) \geq 0$, $Z(k, j) \geq 0$, $i = 1, 2, \dots, l$, $k = 1, 2, \dots, m$, $j = 1, 2, \dots, N$. Clearly, if matrices A and Z are of low rank, their product is also a low rank, at most m , approximation of X . The significance of the above is that every column vector in X is represented by the expansion

$$x_i \approx \sum_{k=1}^m Z(k, i) \mathbf{a}_k, \quad i = 1, 2, \dots, N,$$

where \mathbf{a}_k , $k = 1, 2, \dots, m$, are the column vectors of A and constitute the basis of the expansion. The number of vectors in the basis is less than the dimensionality of the vector itself. Hence, NMF can also be seen as a method for *dimensionality reduction*.

To get a good approximation in Eq. (19.71) one can adopt different costs. The most common cost is the Frobenius norm of the error matrix. In such a setting, the NMF task is cast as follows:

$$\min_{A,Z} \|X - AZ\|_F^2 := \sum_{i=1}^l \sum_{j=1}^N (X(i,j) - [AZ](i,j))^2 \quad (19.72)$$

$$\text{s.t. } A(i,k) \geq 0, Z(k,j) \geq 0, \quad (19.73)$$

where $[AZ](i,j)$ is the (i,j) element of matrix AZ , and i, j, k run over all possible values. Besides the Frobenius norm, other costs have also been suggested (see, e.g., [158]).

Once the problem has been formulated, the major issue rests at the solution of the optimization task. To this end, a number of algorithms have been proposed, for example, Newton type or gradient descent type. Such algorithmic issues, as well as a number of related theoretic ones, are beyond the scope of this book and the interested reader may consult, for example, [50, 62, 168]. More recently, regularized versions, including sparsity-promoting regularizers, have been proposed; see, for example, [51] for a more recent review on the topic.

19.8 LEARNING LOW-DIMENSIONAL MODELS: A PROBABILISTIC PERSPECTIVE

In this section, the emphasis is to look at the dimensionality reduction task from a Bayesian perspective. Our focus will be more on presenting the main ideas and less on algorithmic procedures; the latter depend on the specific model and can be dug out from the palette of algorithms that have already been presented in Chapters 12 and 13. Our path to low-dimensional modeling traces its origin to the so-called *factor analysis*.

19.8.1 FACTOR ANALYSIS

Factor analysis was originally proposed in the work of Charles Spearman [159]. Charles Spearman (1863–1945) was an English psychologist who has made important contributions to statistics. Spearman was interested in human intelligence and developed the method in 1904, for analyzing multiple measures of cognitive performance. He argued that there exists a general intelligence factor (the so-called g-factor) that can be extracted by applying the factor analysis method on intelligence test data. However, this notion has been strongly disputed, as intelligence comprises a multiplicity of components (see, e.g., [78]).

Let $\mathbf{x} \in \mathbb{R}^l$. The factor analysis model assumes that there are $m < l$ underlying (latent) zero-mean variables or *factors* $\mathbf{z} \in \mathbb{R}^m$ so that

$$\mathbf{x}_i - \mu_i = \sum_{j=1}^m a_{ij} z_j + \epsilon_i, \quad i = 1, 2, \dots, l, \quad (19.74)$$

or

$$\mathbf{x} - \boldsymbol{\mu} = A\mathbf{z} + \boldsymbol{\epsilon}, \quad (19.75)$$

where $\boldsymbol{\mu}$ is the mean of \mathbf{x} and $A \in \mathbb{R}^{l \times m}$ is formed by the weights a_{ij} known as *factor loadings*. The variables z_j , $j = 1, 2, \dots, m$, are sometimes called *common factors*, because they contribute to

all observed variables, x_i , and ϵ_i are the *unique* or *specific factors*. As we have already done so far and without loss of generality, we will assume our data is centered, that is, $\mu = \mathbf{0}$. In factor analysis, we assume ϵ_i to be of zero-mean and mutually uncorrelated, that is, $\Sigma_\epsilon = \mathbb{E}[\epsilon\epsilon^T] := \text{diag}\{\sigma_1^2, \sigma_2^2, \dots, \sigma_l^2\}$. We also assume that \mathbf{z} and ϵ are independent. The $m (< l)$ columns of A form a lower dimensional subspace, and ϵ is that part of \mathbf{x} not contained in this subspace. The first question that is now raised is whether the model in Eq. (19.75) is any different from our familiar regression task. The answer is in the affirmative. Note that here the matrix A is not known. All that we are given is the set of observations, \mathbf{x}_n , $n = 1, 2, \dots, N$, and we have to obtain the subspace described by A . It is basically the same linear model that we have considered so far in this chapter, with the difference that now we have introduced the noise term. Once A is known, \mathbf{z}_n can be obtained for each \mathbf{x}_n .

From Eq. (19.75), it is readily seen that

$$\Sigma_x = \mathbb{E}[\mathbf{x}\mathbf{x}^T] = A\mathbb{E}[\mathbf{z}\mathbf{z}^T]A^T + \Sigma_\epsilon.$$

We will further assume that $\mathbb{E}[\mathbf{z}\mathbf{z}^T] = I$; hence, we can write

$$\Sigma_x = AA^T + \Sigma_\epsilon. \quad (19.76)$$

Hence, A results as a factor of $(\Sigma_x - \Sigma_\epsilon)$. However, such a factorization, if it exists, is not unique. This can be easily checked out if we consider $\bar{A} = AU$, where U is an orthonormal matrix. Then, $\bar{A}\bar{A}^T = AA^T$. This has brought a lot of controversy around the factor analysis method when it comes to interpreting individual factors; see, for example, [43] for a discussion. To remedy this drawback, a number of authors have suggested methods and criteria that deal with the rotation (orthogonal or oblique) in order to gain improved interpretation of the factors [147]. However, from our perspective, where our goal is to express our problem in a lower dimensional space, this is not a problem. Any orthonormal matrix imposes a rotation within the subspace spanned by the columns of A ; but we do not care about the exact choice of the coordinates, that is, the common factors.

There are different methods to obtain A (see, e.g., [59]). A popular one is to assume $p(\mathbf{x})$ to be Gaussian and employ the ML method to optimize with respect to the unknown parameters that define Σ_x in Eq. (19.76). Once A becomes available, one way to estimate the factors is to further assume that these can be expressed as linear combinations of the observations, that is,

$$\mathbf{z} = W\mathbf{x}.$$

Post multiplying by \mathbf{x} , taking expectations, recalling Eq. (19.75) and that $\mathbb{E}[\mathbf{z}\mathbf{z}^T] = I$, we get

$$\mathbb{E}[\mathbf{z}\mathbf{x}^T] = \mathbb{E}[\mathbf{z}\mathbf{z}^T A^T] + \mathbb{E}[\mathbf{z}\epsilon^T] = A^T. \quad (19.77)$$

Also,

$$\mathbb{E}[\mathbf{z}\mathbf{x}^T] = W\mathbb{E}[\mathbf{x}\mathbf{x}^T] = W\Sigma_x. \quad (19.78)$$

Hence,

$$W = A^T \Sigma_x^{-1}.$$

Thus, given a value \mathbf{x} , the values of the corresponding latent variables are obtained by

$$\mathbf{z} = A^T \Sigma_x^{-1} \mathbf{x}. \quad (19.79)$$

19.8.2 PROBABILISTIC PCA

New light on this old problem was shed via the Bayesian rationale in the late nineties, [144, 165, 166]; the task was treated for the special case $\Sigma_\epsilon = \sigma^2 I$ and it was named *probabilistic PCA* (PPCA). The latent variables, \mathbf{z} , are dressed with a Gaussian prior,

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\mathbf{0}, I),$$

which is in agreement with the earlier assumption $\mathbb{E}[\mathbf{z}\mathbf{z}^T] = I$, and the conditional pdf is chosen as

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\mathbf{x}|A\mathbf{z}, \sigma^2 I),$$

where, for simplicity, we assume $\boldsymbol{\mu} = 0$ (otherwise the mean would be $A\mathbf{z} + \boldsymbol{\mu}$). We are by now pretty familiar with writing down

$$p(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_{z|x}, \Sigma_{z|x}), \quad (19.80)$$

and

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\mathbf{0}, \Sigma_x), \quad (19.81)$$

where (see Eqs. (12.10), (12.15) and (12.17), Chapter 12)

$$\Sigma_{z|x} = \left(I + \frac{1}{\sigma^2} A^T A \right)^{-1}, \quad (19.82)$$

$$\boldsymbol{\mu}_{z|x} = \frac{1}{\sigma^2} \Sigma_{z|x} A^T \mathbf{x}, \quad (19.83)$$

$$\Sigma_x = \sigma^2 I + A A^T. \quad (19.84)$$

Note that using the Bayesian framework, the computation of the latent variables corresponding to a given set of observations, \mathbf{x} , can naturally be obtained via the posterior $p(\mathbf{z}|\mathbf{x})$ in Eq. (19.80). For example, one can pick the respective mean value

$$\mathbf{z} = \frac{1}{\sigma^2} \Sigma_{z|x} A^T \mathbf{x}. \quad (19.85)$$

Using the matrix inversion lemma (Problem 19.4), it turns out that Eqs. (19.79) and (19.85) are exactly the same; however, now, it comes as a natural consequence of our Bayesian assumptions.

One way to compute A is to apply the ML method on $\prod_{n=1}^N p(\mathbf{x}_n)$ and maximize with regard to A , σ^2 (and $\boldsymbol{\mu}$, if $\boldsymbol{\mu} \neq 0$). It turns out that the maximum likelihood solution for A is given by ([165]),

$$A_{\text{ML}} = U_m \text{diag}\{\lambda_1 - \sigma^2, \dots, \lambda_m - \sigma^2\} R,$$

where U_m is the $l \times m$ matrix with columns the eigenvectors corresponding to the m largest eigenvalues, λ_i , $i = 1, 2, \dots, m$, of the sample covariance matrix of \mathbf{x} , and R is an arbitrary orthogonal matrix ($R R^T = I$). Setting $R = I$, the columns of A are the (scaled) principle directions as computed by the classical PCA, discussed in Section 19.3. In any case, the columns of A span the principle subspace of the standard PCA. Note that as $\sigma^2 \rightarrow 0$, PPCA tends to PCA (Problem 19.5). Also, it turns out that

$$\sigma_{\text{ML}}^2 = \frac{1}{l-m} \sum_{i=m+1}^l \lambda_i. \quad (19.86)$$

The previously established connection with PCA does not come as a surprise. It has been well known for a long time (e.g., [5]) that if in the factor analysis model, one assumes $\Sigma_\epsilon = \sigma^2 I$, then at stationary points of the likelihood function the columns of A are scaled eigenvectors of the sample covariance matrix. Furthermore, σ^2 is the average of the discarded eigenvalues, as suggested in Eq. (19.86).

Another way to estimate A and σ^2 is via the EM algorithm [144, 165]. This is possible because we have $p(z|x)$ in an analytic form. Given the set $(\mathbf{x}_n, \mathbf{z}_n)$, $n = 1, 2, \dots, N$, of the observed and latent variables, the complete log-likelihood function is given by

$$\begin{aligned}\ln p(\mathcal{X}, \mathcal{Z}; A, \sigma^2) &= \sum_{n=1}^N \left(\ln p(\mathbf{x}_n | \mathbf{z}_n; A, \sigma^2) + \ln p(\mathbf{z}_n) \right) \\ &= - \sum_{n=1}^N \left(\frac{l}{2} \ln(2\pi) - \frac{l}{2} \ln \beta + \frac{\beta}{2} \|\mathbf{x}_n - A\mathbf{z}_n\|^2 \right. \\ &\quad \left. + \frac{m}{2} \ln(2\pi) + \frac{1}{2} \mathbf{z}_n^T \mathbf{z}_n \right),\end{aligned}$$

which is of the same form as the one given in Eq. (12.72). We have used $\beta = \frac{1}{\sigma^2}$. Thus, following similar steps as for Eq. (12.72) and rephrasing Eqs. (12.73)–(12.77) to our current notation, the E-step becomes

- E-step:

$$\begin{aligned}\mathcal{Q}(A, \beta; A^{(j)}, \beta^{(j)}) &= - \sum_{n=1}^N \left(-\frac{l}{2} \ln \beta + \frac{1}{2} \|\boldsymbol{\mu}_{z|x}^{(j)}(n)\|^2 + \frac{1}{2} \text{trace} \left\{ \Sigma_{z|x}^{(j)} \right\} \right. \\ &\quad \left. + \frac{l}{2} \|\mathbf{x}_n - A\boldsymbol{\mu}_{z|x}^{(j)}(n)\|^2 + \frac{\beta}{2} \text{trace} \left\{ A \Sigma_{z|x}^{(j)} A^T \right\} \right) + C\end{aligned}$$

where C is a constant and

$$\boldsymbol{\mu}_{z|x}^{(j)}(n) = \beta^{(j)} \Sigma_{z|x}^{(j)} A^{(j)T} \mathbf{x}_n, \quad \Sigma_{z|x}^{(j)} = \left(I + \beta^{(j)} A^{(j)T} A^{(j)} \right)^{-1}.$$

- M-step: Taking the derivatives with regard to β and A and equating to zero (Problem 19.6), we obtain

$$A^{(j+1)} = \left(\sum_{n=1}^N \mathbf{x}_n \boldsymbol{\mu}_{z|x}^{(j)T}(n) \right) \left(N \Sigma_{z|x}^{(j)} + \sum_{n=1}^N \boldsymbol{\mu}_{z|x}^{(j)}(n) \boldsymbol{\mu}_{z|x}^{(j)T}(n) \right)^{-1}, \quad (19.87)$$

and

$$\beta^{(j+1)} = \frac{Nl}{\sum_{n=1}^N \left(\|\mathbf{x}_n - A^{(j+1)} \boldsymbol{\mu}_{z|x}^{(j)}(n)\|^2 + \text{trace} \left\{ A^{(j+1)} \Sigma_{z|x}^{(j)} A^{(j+1)T} \right\} \right)}. \quad (19.88)$$

Observe that having adopted the EM algorithm, one does not need to compute the eigenvalues/eigenvectors of Σ_x . Even retrieving only the m principle components, the lowest cost one has to pay is $\mathcal{O}(ml^2)$ operations. Beyond that, $\mathcal{O}(Nl^2)$ are needed to compute Σ_x . For the EM approach, the covariance matrix need not be computed and the most demanding part comprises the matrix vector

products, which amount to $\mathcal{O}(Nml)$. Hence for $m \ll l$, computational savings are expected compared to the classical PCA. Keep in mind, though, that the two methods optimize different criteria. PCA guarantees minimum least-squares error reconstruction, PPCA via the EM optimizes the likelihood. Thus, for applications where the error reconstruction is important, such as in compression, one has to be aware of this fact; see [165] for a related discussion.

The other alternative route to solve PPCA is by considering A and σ^2 as random variables with appropriate priors and apply the variational EM algorithm, see [23]. This has the added advantage that if one uses as the prior

$$p(A|\alpha) = \prod_{k=1}^m \left(\frac{\alpha_k}{2\pi} \right)^{l/2} \exp \left(-\frac{\alpha_k}{2} \mathbf{a}_k^T \mathbf{a}_k \right),$$

with different precisions, $\alpha_k, k = 1, 2, \dots, m$, per column, then using large enough m one can achieve pruning of the unnecessary components; this was discussed in Section 13.5. Hence, such an approach could provide the means of automatic determination of m . The interested reader, besides the references given before, can dig out useful related information from [45].

Example 19.6. Figure 19.14 shows a set of data, which have been generated via a two-dimensional Gaussian, with zero-mean value and covariance matrix equal to

$$\Sigma = \begin{bmatrix} 5.05 & -4.95 \\ -4.95 & 5.05 \end{bmatrix}.$$

The corresponding eigenvalues/eigenvectors are computed as

$$\begin{aligned} \lambda_1 &= 0.05, & \mathbf{a}_1 &= [1, 1]^T, \\ \lambda_2 &= 5.00, & \mathbf{a}_2 &= [-1, 1]^T. \end{aligned}$$

Observe that the data are distributed mainly around a straight line. The EM PPCA algorithm was run on this set of data, for $m = 1$. The resulting matrix A , which now becomes a vector, is

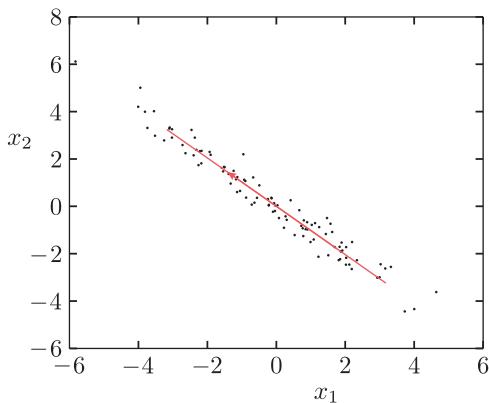


FIGURE 19.14

Data points are distributed around a straight line (one-dimensional subspace) in \mathbb{R}^2 . The subspace is fully recovered by the PPCA, running the EM algorithm.

$$\mathbf{a} = [-1.71, 1.71]^T$$

and $\beta = 0.24$. Note that the obtained vector \mathbf{a} points in the direction of the line (subspace) around which the data are distributed.

Remarks 19.6.

- In PPCA, a special diagonal structure was assumed for Σ_ϵ . An EM algorithm for the more general case has also been derived in the early eighties, [146]. Moreover, if the Gaussian prior imposed on the latent variables is replaced by another one, different algorithms result. For example, if non-Gaussian priors are used, then ICA versions are obtained. As a matter of fact, employing different priors, probabilistic versions of the canonical correlation analysis (CCA) and the partial least-squares (PLS) methods result; related references have already been given in the respective sections. Sparsity-promoting priors have also been used, resulting in what is known as *sparse factor analysis*, for example, [8, 23]. Once the priors have been adopted, one uses standard arguments, more or less, to solve the task, like those discussed in Chapters 12 and 13.
- Besides real-valued variables, extensions to categorical variables have also been considered, for example, [104]. A unifying view of various probabilistic dimensionality reduction techniques is provided in [133].

19.8.3 MIXTURE OF FACTORS ANALYZERS: A BAYESIAN VIEW TO COMPRESSED SENSING

Let us go back to our original model in Eq. (19.75) and rephrase it into a more “trendy” fashion. Matrix A had dimensions $l \times m$ with $m < l$, and $\mathbf{z} \in \mathbb{R}^m$. Let us now make $m > l$. For example, the columns of A may comprise vectors of an overcomplete dictionary. Thus, this section can be considered as the probabilistic counterpart of Section 19.6. The required low dimensionality of the modeling is expressed by imposing sparsity on \mathbf{z} ; we can rewrite the model in terms of the respective observations as, [45],

$$\mathbf{x}_n = A(\mathbf{z}_n \circ \mathbf{b}) + \boldsymbol{\epsilon}_n, \quad n = 1, 2, \dots, N,$$

where N is the number of our training points and the vector $\mathbf{b} \in \mathbb{R}^m$ has elements $b_i \in \{0, 1\}$, $i = 1, 2, \dots, m$. The product $\mathbf{z}_n \circ \mathbf{b}$ is the point-wise vector product, that is,

$$\mathbf{z}_n \circ \mathbf{b} = [z_n(1)b_1, z_n(2)b_2, \dots, z_n(m)b_m]^T. \quad (19.89)$$

If $\|\mathbf{b}\|_0 \ll l$, then \mathbf{x}_n is sparsely represented in terms of the columns of A and its intrinsic dimensionality is equal to $\|\mathbf{b}\|_0$. Adopting the same assumptions as before, that is,

$$p(\boldsymbol{\epsilon}) = \mathcal{N}(\boldsymbol{\epsilon} | \mathbf{0}, \beta^{-1}I_l), \quad p(\mathbf{z}) = \mathcal{N}(\mathbf{z} | \mathbf{0}, \alpha^{-1}I_m),$$

where now we have explicitly brought l and m into the notation in order to remind us of the associated dimensions. Also, for the sake of generality, we have assumed that the elements of \mathbf{z} correspond to precision values different than one. Following our familiar standard arguments (as for Eq. (12.15)), it is readily shown that the observations \mathbf{x}_n , $n = 1, 2, \dots, N$, are drawn from

$$\mathbf{x} \sim \mathcal{N}(\mathbf{x} | \mathbf{0}, \Sigma_x), \quad (19.90)$$

$$\Sigma_x = \alpha^{-1}A\Lambda A^T + \beta^{-1}I_l, \quad (19.91)$$

where

$$\Lambda = \text{diag} \{b_1, \dots, b_m\}, \quad (19.92)$$

which guarantees that in Eq. (19.91) only the columns of A , which correspond to nonzero values of \mathbf{b} , contribute to the formation of Σ_x . We can rewrite the matrix product in the following form:

$$A \Lambda A^T = \sum_{i=1}^m b_i \mathbf{a}_i \mathbf{a}_i^T,$$

and because only $\|\mathbf{b}\|_0 := k \ll l$ nonzero terms contribute to the summation, this corresponds to a rank $k < l$ matrix, provided that the respective columns of A are linear independent. Furthermore, assuming that β^{-1} is small, then Σ_x turns out to have a rank approximately equal to k .

Our goal now becomes the learning of the involved parameters; that is, A , β , α , and Λ . This can be done in a standard Bayesian setting by imposing priors on α , β (typically gamma pdfs) and for the columns of A ,

$$p(\mathbf{a}_i) = \mathcal{N} \left(\mathbf{a}_i | 0, \frac{1}{l} I_l \right), \quad i = 1, 2, \dots, m,$$

which guarantees unit expected norm for each column. The prior for the elements of \mathbf{b} are chosen to follow a Bernoulli distribution (see [45] for more details).

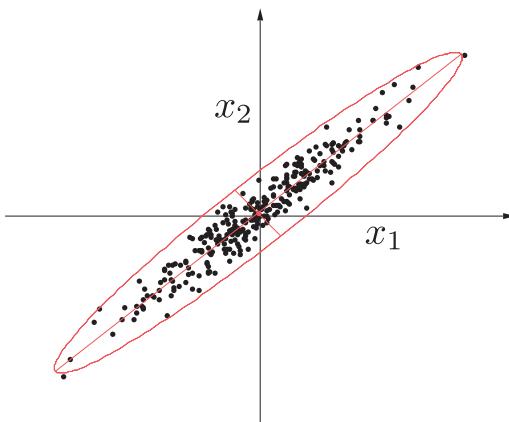
Before generalizing the model, let us see the underlying geometric interpretation of the adopted model. Recall from our statistics basics (see also, Section 2.3.2, Chapter 2) that most of the activity of a set of jointly Gaussian variables takes place within an (hyper)ellipsoid whose principle axes are determined by the eigenstructure of the covariance matrix. Thus, assuming that the values of \mathbf{x} lie close to a subspace/(hyper)plane, the resulting Gaussian model Eqs. (19.90)–(19.91) can sufficiently model it by adjusting the elements of Σ_x (after training) so that the corresponding high probability region forms a sufficiently flat ellipsoid; see Figure 19.15 for an illustration.

Once we have established the geometric interpretation of our factor model, let us leave our imagination free to act. Can this viewpoint be extended for modeling data that originate from a union of subspaces? A reasonable response to this challenge would be to resort to a mixture of factors; one for each subspace. However, there is more to it than that. It has been shown, for example, [25] that a compact manifold can be covered by a finite number of topological disks, whose dimensionality is equal to the dimensionality of the manifold. Associating topological disks with the principle hyperplanes that define sufficiently flat hyperellipsoids, one can model the data activity, which takes place along a manifold, by a sufficient number of factors, one per ellipsoid ([45]).

A *mixture of factor analyzers* (MSA) is defined as

$$p(\mathbf{x}) = \sum_{j=1}^J P_j \mathcal{N} \left(\mathbf{x} | \boldsymbol{\mu}_j, \alpha_j^{-1} A_j \Lambda_j A_j^T + \beta^{-1} I_l \right), \quad (19.93)$$

where $\sum_{j=1}^J P_j = 1$, $\Lambda_j = \text{diag} \{b_{j1}, \dots, b_{jm}\}$, $b_{ji} \in \{0, 1\}$, $i = 1, 2, \dots, m$. The expansion in Eq. (19.93) for fixed J and preselected Λ_j , for the j th factor, has been known for some time; in this context, learning of the unknown parameters is achieved in the Bayesian framework, by imposing appropriate priors and mobilizing techniques such as the variational EM (e.g., [74]), the EM ([165]),

**FIGURE 19.15**

Data points that lie close to a hyperplane can be sufficiently modeled by a Gaussian pdf whose high probability region corresponds to a sufficiently flat (hyper)ellipsoid.

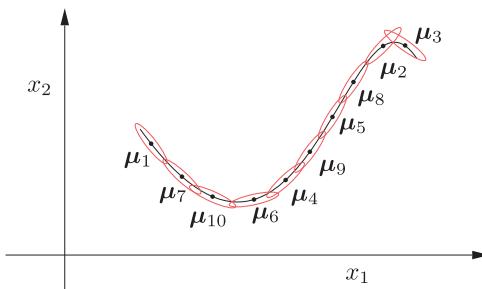
and the maximum likelihood [170]. In a more recent treatment of the problem, the dimensionality of each Λ_j , $j = 1, 2, \dots, J$, as well as the number of factors, J , can be learned by the learning scheme. To this end, *nonparametric priors* are mobilized; see, for example, [39, 45, 87], and [Section 13.12](#). The model parameters are then computed via Gibbs sampling ([Chapter 14](#)) or variational Bayesian techniques. Note that in general, different factors may turn out to have different dimensionality.

For *nonlinear manifold learning*, the geometric interpretation of Eq. (19.93) is illustrated in [Figure 19.16](#). The number J is the number of flat ellipsoids used to cover the manifold, μ_j are the sampled points on the manifold, the columns $A_j \Lambda_j$ (approximately) span the local k -dimensional tangent subspace, the noise variance β^{-1} depends on the manifold curvature, and the weights P_j reflect the respective density of the points across the manifold. The method has also been used for matrix completion (see also [Section 19.10.1](#)) via a low rank matrix approximation of the involved matrices ([39]).

Once the model has been learned, using Bayesian inference on a set of training data \mathbf{x}_n , $n = 1, 2, \dots, N$, it can subsequently be used for compressed sensing; that is, to be able to obtain any \mathbf{x} , which belongs in the ambient space \mathbb{R}^l but “lives” in the learned k -dimensional manifold, modeled by Eq. (19.93), using $K \ll l$ measurements. To this end, in a complete analogy with what has been said in [Chapter 9](#), one has to determine a sensing matrix, which is denoted here as $\Phi \in \mathbb{R}^{K \times l}$, so that to be able to recover \mathbf{x} from the measured (projection) vector

$$\mathbf{y} = \Phi \mathbf{x} + \boldsymbol{\eta},$$

where $\boldsymbol{\eta}$ denotes the vector of the (unobserved) samples of the measurement noise; all that is now needed is to compute the posterior $p(\mathbf{x}|\mathbf{y})$. Assuming the noise samples follow a Gaussian and because $p(\mathbf{x})$ is a sum of Gaussians, it can be readily seen that the posterior is also a sum of Gaussians determined by the parameters in Eq. (19.93) and the covariance matrix of the noise vector. Hence, \mathbf{x} can be recovered

**FIGURE 19.16**

The curve (manifold) is covered by a number of sufficiently flat ellipsoids centered at the respective mean values.

by a substantially smaller number of measurements, K , compared to l . In [45], a theoretical analysis is carried out that relates the dimensionality of the manifold, k , the dimensionality of the ambient space, l , and Gaussian/sub-Gaussian types of sensing matrices Φ ; this is an analogy to the RIP so that a stable embedding is guaranteed (Section 9.9).

One has to point out a major difference between the techniques developed in Chapter 9 and the current section. There, the model that generates the data was assumed to be known; the signal, denoted there by s , was written as

$$s = \Psi\theta,$$

where Ψ was the matrix of the dictionary and θ the sparse vector. That is, the signal was assumed to reside in a subspace, which is spanned by some of the columns of Ψ ; in order to recover the signal vector, one had to search for it in a union of subspaces. In contrast, in the current section, we had to “learn” the manifold in which the signal, denoted here by x , lies.

19.9 NONLINEAR DIMENSIONALITY REDUCTION

All the techniques that have been considered so far build around linear models, which relate the observed and the latent variables. In this section, we turn our attention to their nonlinear relatives. Our aim is to discuss the main directions that are currently popular and we will not delve into many details. The interested reader can get a deeper understanding and related implementation details from the references provided in the text.⁶

19.9.1 KERNEL PCA

As its name suggests, this is a kernelized version of the classical PCA, and it was first introduced in [151]. As we have seen in Chapter 11, the idea behind any kernelized version of a linear method is to map the variables that originally lie in a low dimensional space, \mathbb{R}^l into a high (possibly infinite)

⁶ Much of this section is based on [164].

dimensional reproducing kernel Hilbert space (RKHS). This is achieved by adopting an implicit mapping,

$$\mathbf{x} \in \mathbb{R}^I \longmapsto \phi(\mathbf{x}) \in \mathbb{H}. \quad (19.94)$$

Let \mathbf{x}_n , $n = 1, 2, \dots, N$, be the available training points. The sample covariance matrix of the images, after mapping into \mathbb{H} and assuming centered data, is given by⁷

$$\hat{\Sigma} = \frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^T. \quad (19.95)$$

The goal is to perform the eigendecomposition of $\hat{\Sigma}$, that is,

$$\hat{\Sigma} \mathbf{u} = \lambda \mathbf{u}. \quad (19.96)$$

By the definition of $\hat{\Sigma}$, it can be shown that \mathbf{u} lies in the span $\{\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \dots, \phi(\mathbf{x}_N)\}$. Indeed,

$$\lambda \mathbf{u} = \left(\frac{1}{N} \sum_{n=1}^N \phi(\mathbf{x}_n) \phi(\mathbf{x}_n)^T \right) \mathbf{u} = \frac{1}{N} \sum_{n=1}^N (\phi(\mathbf{x}_n)^T \mathbf{u}) \phi(\mathbf{x}_n),$$

and for $\lambda \neq 0$ we can write

$$\mathbf{u} = \sum_{n=1}^N a_n \phi(\mathbf{x}_n). \quad (19.97)$$

Combining Eqs. (19.96) and (19.97), it turns out (Problem 19.7) that the problem is equivalent to performing an eigendecomposition of the corresponding kernel matrix (Chapter 11)

$$\mathcal{K}\mathbf{a} = N\lambda\mathbf{a}, \quad (19.98)$$

where

$$\mathbf{a} := [a_1, a_2, \dots, a_N]^T. \quad (19.99)$$

As we already know (Section 11.5.1), the elements of the kernel matrix are $\mathcal{K}(i, j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$ with $\kappa(\cdot, \cdot)$ being the adopted kernel function. Thus, the k th eigenvector of $\hat{\Sigma}$, corresponding to the k th (nonzero) eigenvalue of \mathcal{K} in Eq. (19.98), is expressed as

$$\mathbf{u}_k = \sum_{n=1}^N a_{kn} \phi(\mathbf{x}_n), \quad k = 1, 2, \dots, p \quad (19.100)$$

where $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_p$ denote the respective eigenvalues in descending order and λ_p is the smallest nonzero one and $\mathbf{a}_k^T := [a_{k1}, \dots, a_{kN}]$ is the k th eigenvector of the kernel matrix. The latter is assumed to be normalized so that $\langle \mathbf{u}_k, \mathbf{u}_k \rangle = 1$, $k = 1, 2, \dots, p$, where $\langle \cdot, \cdot \rangle$ is the inner product in the Hilbert space \mathbb{H} . This imposes an equivalent normalization on the respective \mathbf{a}_k 's, resulting from

$$1 = \langle \mathbf{u}_k, \mathbf{u}_k \rangle = \left\langle \sum_{i=1}^N a_{ki} \phi(\mathbf{x}_i), \sum_{j=1}^N a_{kj} \phi(\mathbf{x}_j) \right\rangle$$

⁷ If the dimension of \mathbb{H} is infinite, the definition of the covariance matrix needs a special interpretation, but we will not bother with it here.

$$\begin{aligned}
&= \sum_{i=1}^N \sum_{j=1}^N a_{ki} a_{kj} \mathcal{K}(i, j) \\
&= \mathbf{a}_k^T \mathcal{K} \mathbf{a}_k = N \lambda_k \mathbf{a}_k^T \mathbf{a}_k, \quad k = 1, 2, \dots, p.
\end{aligned} \tag{19.101}$$

We are now ready to summarize the basic steps for performing a kernel PCA; that is, to compute the corresponding latent variables (kernel principle components). Given $\mathbf{x}_n \in \mathbb{R}^l$, $n = 1, 2, \dots, N$, and a kernel function $\kappa(\cdot, \cdot)$

- Compute the $N \times N$ kernel matrix, with elements $\mathcal{K}(i, j) = \kappa(\mathbf{x}_i, \mathbf{x}_j)$.
- Compute the m dominant eigenvalues/eigenvectors λ_k , \mathbf{a}_k , $k = 1, 2, \dots, m$, of \mathcal{K} (Eq. (19.98)).
- Perform the required normalization (Eq. (19.101)).
- Given a feature vector $\mathbf{x} \in \mathbb{R}^l$, obtain its low-dimensional representation by computing the m projections onto each one of the dominant eigenvectors,

$$z_k := \langle \phi(\mathbf{x}), \mathbf{u}_k \rangle = \sum_{n=1}^N a_{kn} \kappa(\mathbf{x}, \mathbf{x}_n), \quad k = 1, 2, \dots, m. \tag{19.102}$$

The operations given in Eq. (19.102) correspond to a *nonlinear mapping* in the input space. Note that, in contrast to the linear PCA, the dominant eigenvectors \mathbf{u}_k , $k = 1, 2, \dots, m$, are not computed explicitly. All we know are the respective (nonlinear) projections, z_k along them. However, after all, this is what we are finally interested in.

Remarks 19.7.

- Kernel PCA is equivalent to performing a standard PCA in the RKHS \mathbb{H} . It can be shown that all the properties associated with the dominant eigenvectors, as discussed for the PCA, are still valid for the kernel PCA. That is, (a) the dominant eigenvector directions optimally retain most of the variance; (b) the MSE in approximating a vector (function) in \mathbb{H} in terms of the m dominant eigenvectors is minimal, with respect to any other m directions; and (c) projections onto the eigenvectors are uncorrelated [151].
- Recall from Remarks 19.1 that the eigendecomposition of the Gram matrix was required for the metric multidimensional scaling (MDS) method. Because the kernel matrix is the Gram matrix in RKHS, kernel PCA can be considered as a kernelized version of MDS, where inner products in the input space have been replaced by kernel operations in the Gram matrix.
- Note that the kernel PCA method does not consider an explicit underlying structure of the manifold on which the data reside.
- A variant of the kernel PCA, known as the kernel entropy component analysis (ECA), has been developed in [96], where the dominant directions are selected so as to maximize the Renyi entropy.

19.9.2 GRAPH-BASED METHODS

Laplacian eigenmaps

The starting point of this method is the assumption that the points in the data set, \mathcal{X} , lie on a smooth manifold $\mathcal{M} \supset \mathcal{X}$, whose intrinsic dimension is equal to $m < l$ and it is embedded in \mathbb{R}^l , that is, $\mathcal{M} \subset$

\mathbb{R}^l . The dimension m is given as a parameter by the user. In contrast, this is not required in the kernel PCA, where m is the number of dominant components, which, in practice, is determined so that the gap between λ_m and λ_{m+1} has a “large” value.

The main philosophy behind the method is to compute the low-dimensional representation of the data so that *local neighborhood information* in $\mathcal{X} \subset \mathcal{M}$ is optimally preserved. In this way, one attempts to get a solution that reflects the geometric structure of the manifold. To achieve this, the following steps are in order:

Step 1: Construct a graph $G = (V, E)$, where $V = \{v_n, n = 1, 2, \dots, N\}$ is a set of vertices and $E = \{e_{ij}\}$ is the corresponding set of edges connecting vertices (v_i, v_j) , $i, j = 1, 2, \dots, N$ (see also [Chapter 15](#)). Each node, v_n , of the graph corresponds to a point, \mathbf{x}_n , in the data set, \mathcal{X} . We connect v_i, v_j , that is, insert the edge e_{ij} between the respective nodes, if points $\mathbf{x}_i, \mathbf{x}_j$ are “close” to each other. According to the method, there are two ways of quantifying “closeness.” Vertices v_i, v_j are connected with an edge if:

1. $\|\mathbf{x}_i - \mathbf{x}_j\|^2 < \epsilon$, for some user-defined parameter ϵ , where $\|\cdot\|$ is the Euclidean norm in \mathbb{R}^l , or
2. \mathbf{x}_j is among the k -nearest neighbors of \mathbf{x}_i or \mathbf{x}_i is among the k -nearest neighbors of \mathbf{x}_j , where k is a user-defined parameter and neighbors are chosen according to the Euclidean distance in \mathbb{R}^l . The use of the Euclidean distance is justified by the smoothness of the manifold that allows to approximate, locally, manifold geodesics by Euclidean distances in the space where the manifold is embedded. The latter is a known result from differential geometry.

For those who are unfamiliar with such concepts, think of a sphere embedded in the three-dimensional space. If somebody is constrained to live on the surface of the sphere, the shortest path to go from one point to another is the geodesic between these two points. Obviously this is not a straight line but an arc across the surface of the sphere. However, if these points are close enough, their geodesic distance can be approximated by their Euclidean distance, computed in the three-dimensional space.

Step 2: Each edge, e_{ij} , is associated with a weight, $W(i, j)$. For nodes that are not connected, the respective weights are zero. Each weight, $W(i, j)$, is a measure of the “closeness” of the respective neighbors, $\mathbf{x}_i, \mathbf{x}_j$. A typical choice is

$$W(i, j) = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}\right), & \text{if } v_i, v_j \text{ correspond to neighbors,} \\ 0 & \text{otherwise,} \end{cases}$$

where σ^2 is a user-defined parameter. We form the $N \times N$ weight matrix W having as elements the weights $W(i, j)$. Note that W is symmetric and it is *sparse* because, in practice, many of its elements turn out to be zero.

Step 3: Define the diagonal matrix D with elements $D_{ii} = \sum_j W(i, j)$, $i = 1, 2, \dots, N$, and also the matrix $L := D - W$. The latter is known as the *Laplacian matrix of the graph*, $G(V, E)$. Perform the generalized eigendecomposition

$$Lu = \lambda Du.$$

Let $0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_m$ be the smallest $m+1$ eigenvalues.⁸ Ignore the \mathbf{u}_o eigenvector corresponding to $\lambda_0 = 0$ and choose the next m eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_m$. Then map

$$\mathbf{x}_n \in \mathbb{R}^l \mapsto \mathbf{z}_n \in \mathbb{R}^m, \quad n = 1, 2, \dots, N,$$

where

$$\mathbf{z}_n^T = [u_{1n}, u_{2n}, \dots, u_{mn}], \quad n = 1, 2, \dots, N. \quad (19.103)$$

That is, \mathbf{z}_n comprises the n th components of the m previous eigenvectors. The computational complexity of a general eigendecomposition solver amounts to $O(N^3)$ operations. However, for sparse matrices, such as the Laplacian matrix, L , efficient schemes can be employed to reduce complexity to be subquadratic in N , e.g., the Lanczos algorithm [77].

The proof concerning the statement of step 3, will be given for the case of $m = 1$. For this case, the low-dimensional space is the real axis. Our path evolves along the lines adopted in [18]. The goal is to compute $\mathbf{z}_n \in \mathbb{R}$, $n = 1, 2, \dots, N$, so that connected points (in the graph, i.e., neighbors) stay as close as possible after the mapping onto the one-dimensional subspace. The criterion used to satisfy the closeness after the mapping is

$$E_L = \sum_{i=1}^N \sum_{j=1}^N (z_i - z_j)^2 W(i,j), \quad (19.104)$$

to become minimum. Observe that if $W(i,j)$ has a large value (i.e., $\mathbf{x}_i, \mathbf{x}_j$ are close in \mathbb{R}^l), then if the respective z_i, z_j are far apart in \mathbb{R} it incurs a heavy penalty in the cost function. Also, points that are not neighbors do not affect the minimization as the respective weights are zero. For the more general case, where $1 < m < l$, the cost function becomes

$$E_L = \sum_{i=1}^N \sum_{j=1}^N \|z_i - z_j\|^2 W(i,j).$$

Let us now reformulate Eq. (19.104). After some trivial algebra, we obtain

$$\begin{aligned} E_L &= \sum_i z_i^2 \sum_j W(i,j) + \sum_j z_j^2 \sum_i W(i,j) - 2 \sum_i \sum_j z_i z_j W(i,j) \\ &= \sum_i z_i^2 D_{ii} + \sum_j z_j^2 D_{jj} - 2 \sum_i \sum_j z_i z_j W(i,j) \\ &= 2\mathbf{z}^T L \mathbf{z}, \end{aligned} \quad (19.105)$$

where

$L := D - W : \quad \text{Laplacian Matrix of the Graph,}$

(19.106)

and $\mathbf{z}^T = [z_1, z_2, \dots, z_N]$. The Laplacian matrix, L , is symmetric and positive semidefinite. The latter is readily seen from the definition in Eq. (19.105), where E_L is always a nonnegative scalar. Note that the

⁸ In contrast to the notation used for PCA, the eigenvalues here are marked in ascending order. This is because, in this subsection, we are interested in determining the smallest values and such a choice is notationally more convenient.

larger the value of D_{ii} the more “important” is the sample \mathbf{x}_i . This is because it implies large values for $W(i,j)$, $j = 1, 2, \dots, N$, and plays a dominant role in the minimization process. Obviously, the minimum of E_L is achieved by the trivial solution $z_i = 0$, $i = 1, 2, \dots, N$. To avoid this, as it is common in such cases, we constrain the solution to a prespecified norm. Hence, our problem now becomes

$$\begin{aligned} \min_{\mathbf{z}} \quad & \mathbf{z}^T \mathbf{L} \mathbf{z}, \\ \text{s.t.} \quad & \mathbf{z}^T \mathbf{D} \mathbf{z} = 1. \end{aligned}$$

Although we can work directly on the previous task, we will slightly reshape it in order to use tools that are more familiar to us. Define

$$\mathbf{y} = \mathbf{D}^{1/2} \mathbf{z}, \quad (19.107)$$

and

$$\tilde{\mathbf{L}} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2}, \quad (19.108)$$

which is known as the *normalized graph Laplacian* matrix. It is now readily seen that our optimization problem becomes

$$\min_{\mathbf{y}} \quad \mathbf{y}^T \tilde{\mathbf{L}} \mathbf{y}, \quad (19.109)$$

$$\text{s.t.} \quad \mathbf{y}^T \mathbf{y} = 1. \quad (19.110)$$

Using Lagrange multipliers and equating the gradient of the Lagrangian to zero, it turns out that the solution is given by

$$\tilde{\mathbf{L}} \mathbf{y} = \lambda \mathbf{y}. \quad (19.111)$$

In other words, computing the solution becomes equivalent to solving an eigenvalue-eigenvector problem. Substituting Eq. (19.111) into the cost function in (19.109) and taking into account the constraint (19.110), it turns out that the value of the cost associated with the optimal \mathbf{y} is equal to λ . Hence, the solution is the eigenvector corresponding to the minimum eigenvalue. However, the minimum eigenvalue of $\tilde{\mathbf{L}}$ is zero and the corresponding eigenvector corresponds to a trivial solution. Indeed, observe that

$$\tilde{\mathbf{L}} \mathbf{D}^{1/2} \mathbf{1} = \mathbf{D}^{-1/2} \mathbf{L} \mathbf{D}^{-1/2} \mathbf{D}^{1/2} \mathbf{1} = \mathbf{D}^{-1/2} (\mathbf{D} - \mathbf{W}) \mathbf{1} = \mathbf{0},$$

where $\mathbf{1}$ is the vector having all its elements equal 1. In words, $\mathbf{y} = \mathbf{D}^{1/2} \mathbf{1}$ is an eigenvector corresponding to the zero eigenvalue and it results in the trivial solution, $z_i = 1$, $i = 1, 2, \dots, N$. That is, all the points are mapped onto the same point in the real line. To exclude this undesired solution, recall that $\tilde{\mathbf{L}}$ is a positive semidefinite matrix and, hence, 0 is its smallest eigenvalue. In addition, if the graph is assumed to be connected, that is, there is at least one path (see Chapter 15) that connects any pair of vertices, $\mathbf{D}^{1/2} \mathbf{1}$ is the only eigenvector associated with the zero eigenvalue, λ_0 , [18]. Also, as $\tilde{\mathbf{L}}$ is a symmetric matrix, we know (Appendix A.2) that its eigenvectors are orthogonal to each other. In the sequel, we impose an extra constraint and we now require the solution to be *orthogonal* to $\mathbf{D}^{1/2} \mathbf{1}$. Constraining the solution to be orthogonal to the eigenvector corresponding to the smallest (zero) eigenvalue, drives the solution to the next eigenvector corresponding to the next smallest (nonzero) eigenvalue λ_1 . Note that the eigendecomposition of $\tilde{\mathbf{L}}$ is equivalent to what we called generalized eigendecomposition of \mathbf{L} in step 3 before.

For the more general case of $m > 1$, we have to compute the m eigenvectors associated with $\lambda_1 \leq \dots \leq \lambda_m$. As a matter of fact, for this case, the constraints prevent us from mapping into a subspace of dimension less than the desired m . For example, we do not want to project in a three-dimensional space and the points to lie on a two-dimensional plane or on a one-dimensional line. For more details, the interested reader is referred to the insightful paper [18].

Local linear embedding (LLE)

As was the case with the Laplacian eigenmap method, *local linear embedding* (LLE) assumes that the data points rest on a smooth enough manifold of dimension m , which is embedded in the \mathbb{R}^l space, with $m < l$ [145]. The smoothness assumption allows us to further assume that, provided there is sufficient data and the manifold is “well” sampled, nearby points lie on (or close to) a “locally” *linear* patch of the manifold (see, also, related comments in Section 19.8.3). The algorithm in its simplest form is summarized in the following three steps:

- Step 1: For each point, \mathbf{x}_n , $n = 1, 2, \dots, N$, search for its nearest neighbors.
- Step 2: Compute the weights $W(i,j)$, $i, j = 1, 2, \dots, N$, that best reconstruct each point, \mathbf{x}_n , from its nearest neighbors, so as to minimize the cost

$$\arg \min_W E_W = \sum_{n=1}^N \left\| \mathbf{x}_n - \sum_{j=1}^N W(i,j) \mathbf{x}_{n_j} \right\|^2, \quad (19.112)$$

where \mathbf{x}_{n_j} denotes the j th neighbor of the n th point. The weights are constrained: (a) to be zero for points which are not neighbors and (b) the rows of the weight matrix add to one, that is,

$$\sum_{j=1}^N W(i,j) = 1. \quad (19.113)$$

That is, the sum of the weights, over all neighbors, must be equal to one.

Step 3: Once the weights have been computed from the previous step, use them to obtain the corresponding points $\mathbf{z}_n \in \mathbb{R}^m$, $n = 1, 2, \dots, N$, so that to minimize the cost with respect to the unknown set of points $\mathcal{Z} = \{\mathbf{z}_n, n = 1, 2, \dots, N\}$,

$$\arg \min_{\mathbf{z}_n: n=1, \dots, N} E_{\mathcal{Z}} = \sum_{n=1}^N \left\| \mathbf{z}_n - \sum_{j=1}^N W(n,j) \mathbf{z}_j \right\|^2. \quad (19.114)$$

The above minimization takes place subject to two constraints, to avoid degenerate solutions: (a) the outputs are centered, $\sum_n \mathbf{z}_n = \mathbf{0}$, and (b) the outputs have unit covariance matrix [149]. Nearest points, in step 1, are searched in the same way as it is carried out for the Laplacian eigenmap method. Once again, the use of the Euclidean distance is justified by the smoothness of the manifold, as long as the search is limited “locally” among neighboring points. For the second step, the method exploits the local linearity of a smooth manifold and tries to predict *linearly* each point by its neighbors using the least-squares error criterion. Minimizing the cost subject to the constraint given in Eq. (19.113) results in a solution that satisfies the following three properties:

1. Rotation invariance.
2. Scale invariance.
3. Translation invariance.

The first two can easily be verified by the form of the cost function and the third one is the consequence of the imposed constraints. The implication of this is that the computed weights encode information about the intrinsic characteristics of each neighborhood and they do not depend on the particular point.

The resulting weights, $W(i,j)$, reflect the intrinsic properties of the local geometry underlying the data, and because our goal is to retain the local information after the mapping, these weights are used to reconstruct each point in the \mathbb{R}^m subspace by its neighbors. As is nicely stated in [149], it is as if we take a pair of scissors to cut small linear patches of the manifold and place them in the low-dimensional subspace.

It turns out that solving (19.114) for the unknown points, z_n , $n = 1, 2, \dots, N$, is equivalent to

- Performing an eigendecomposition of the matrix $(I - W)^T(I - W)$.
- Discarding the eigenvector that corresponds to the smallest eigenvalue.
- Taking the eigenvectors that correspond to the next (smaller) eigenvalues. These yield the low-dimensional latent variable scores, z_n , $n = 1, 2, \dots, N$.

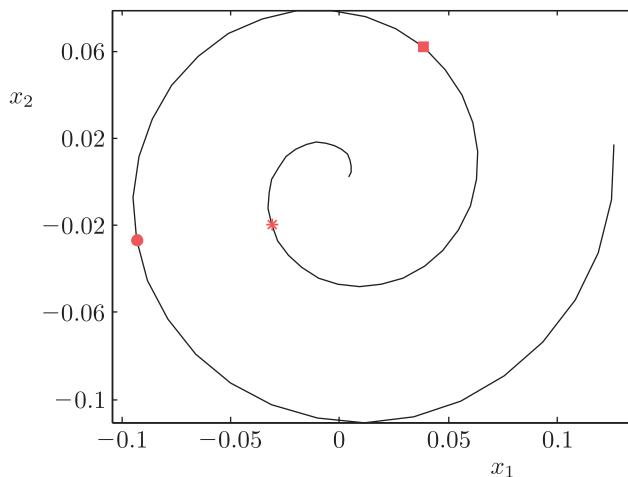
Once again, the involved matrix W is sparse and if this is taken into account the eigenvalue problem scales relatively well to large data sets with complexity subquadratic in N . The complexity for step 2 scales as $O(Nk^3)$ and it is contributed by the solver of the linear set of equations with k unknowns for each point. The method needs two parameters to be provided by the user, the number of nearest neighbors, k (or ϵ) and the dimensionality m . The interested reader can find more on the LLE method in [149].

Isometric mapping (ISOMAP)

In contrast to the two previous methods that unravel the geometry of the manifold on a local basis, the ISOMAP algorithm adopts the view that only the geodesic distances between all pairs of the data points can reflect the true structure of the manifold. Euclidean distances between points in a manifold cannot represent it properly, because points that lie far apart, as measured by their geodesic distance, may be close when measured in terms of their Euclidean distance (see Figure 19.17). ISOMAP is basically a variant of the multidimensional scaling (MDS) algorithm, in which the Euclidean distances are substituted by the respective geodesic distances along the manifold. The essence of the method is to estimate geodesic distances between points that lie faraway. To this end, a two-step procedure is adopted.

Step 1: For each point, x_n , $n = 1, 2, \dots, N$, compute the nearest neighbors and construct a graph $G(V, E)$ whose vertices represent the data points and the edges connect nearest neighbors. (Nearest neighbors are computed with either of the two alternatives used for the Laplacian eigenmap method. The parameters k or ϵ are user-defined parameters.) The edges are assigned weights based on the respective Euclidean distance (for nearest neighbors, this is a good approximation of the respective geodesic distance).

Step 2: Compute the pairwise geodesic distances among all pairs (i,j) , $i, j = 1, 2, \dots, N$, along shortest paths through the graph. The key assumption is that the geodesic between any two points on the manifold can be approximated by the *shortest path* connecting the two points along the graph $G(V, E)$. To this end, efficient algorithms can be used to achieve it with complexity $\mathcal{O}(N^2 \ln N + N^2 k)$ (e.g., Djikstar's algorithm, [55]). This cost can be prohibitive for large values of N .

**FIGURE 19.17**

The point denoted by a “star,” is deceptively closer to the point denoted by a “dot” than to the point denoted by a “box,” if distance is measured in terms of the Euclidean distance. However, if one is constrained to travel along the spiral, the geodesic distance is the one that determines closeness and it is the “box” point that is closer to the “star.”

Having estimated the geodesics between all pairs of point, the MDS method is mobilized. Thus, the problem becomes equivalent to performing the eigendecomposition of the respective Gram matrix and selecting the m most dominant eigenvectors to represent the low-dimensional space. After the mapping, Euclidean distances between points in the low-dimensional subspace match the respective geodesic distances on the manifold in the original high-dimensional space. As it is the case in PCA and MDS, m is estimated by the number of significant eigenvalues. It can be shown that ISOMAP is guaranteed asymptotically ($N \rightarrow \infty$) to recover the true dimensionality of a class of nonlinear manifolds [61, 163].

All three graph-based methods share a common step for computing nearest neighbors in a graph. This is a problem of complexity $\mathcal{O}(N^2)$ but more efficient search techniques can be used by employing a special type of data structures, for example, [22]. A notable difference between the ISOMAP on the one side and the Laplacian eigenmap and LLE methods on the other is that the latter two approaches rely on the eigendecomposition of sparse matrices as opposed to the ISOMAP that relies on the eigendecomposition of the dense Gram matrix. This gives a computational advantage to the Laplacian eigenmap and LLE techniques. Moreover, the calculation of the shortest paths in the ISOMAP is another computationally demanding task. Finally, it is of interest to note that the three graph-based techniques perform the task of dimensionality reduction while trying to unravel, in one way or another, the geometric properties of the manifold on which the data (approximately) lie. In contrast, this is not the case with the kernel PCA, which shows no interest in any manifold learning. However, as the world is very small, in [81] it is pointed out that the graph-based techniques can be seen as special cases of the kernel PCA! This becomes possible if data-dependent kernels, derived from graphs encoding neighborhood information, are used in place of predefined kernel functions.

The goal of this section was to present some of the most basic directions that have been suggested for nonlinear dimensionality reduction. Besides the previous basic schemes, a number of variants have been proposed in the literature (e.g., [20, 65, 153]). In [140] and [111] (*diffusion maps*), the low-dimensional embedding is achieved so as to preserve certain measures that reflect the connectivity of the graph $G(V, E)$. In [29, 94], the idea of preserving the local information in the manifold has been carried out to define linear transforms of the form $\mathbf{z} = \mathbf{A}^T \mathbf{x}$, and the optimization is now carried out with respect to the elements of \mathbf{A} . The task of incremental manifold learning for dimensionality reduction was more recently considered in [114]. In [162, 173], the *maximum variance unfolding* method is introduced. The variance of the outputs is maximized under the constraint that (local) distances and angles are preserved among neighbors in the graph. Like the ISOMAP, it turns out that the top eigenvectors of a Gram matrix have to be computed, albeit avoiding the computationally demanding step of estimating geodesic distances, as it required by the ISOMAP. In [154], a general framework, called *graph embedding*, is presented that offers a unified view for understanding and explaining a number of known (including PCA and nonlinear PCA) dimensionality reduction techniques and it also offers a platform for developing new ones. For a more detailed and insightful treatment of the topic, the interested reader is referred to [27]. A review of nonlinear dimensionality reduction techniques can be found in, for example, [31, 116].

Example 19.7. Let a data set consisting of 30 points be in the two-dimensional space. The points result from sampling the spiral of Archimedes (see Figure 19.18a), described by

$$x_1 = a\theta \cos \theta, \quad x_2 = a\theta \sin \theta.$$

The points of the data set correspond to the values $\theta = 0.5\pi, 0.7\pi, 0.9\pi, \dots, 2.05\pi$ (θ is expressed in radians), and $a = 0.1$. For illustration purposes and in order to keep track of the “neighboring” information, we have used a sequence of six symbols, “x”, “+”, “*”, “□”, “◇”, “○” with black color, followed by the same sequence of symbols in red color, repeatedly.

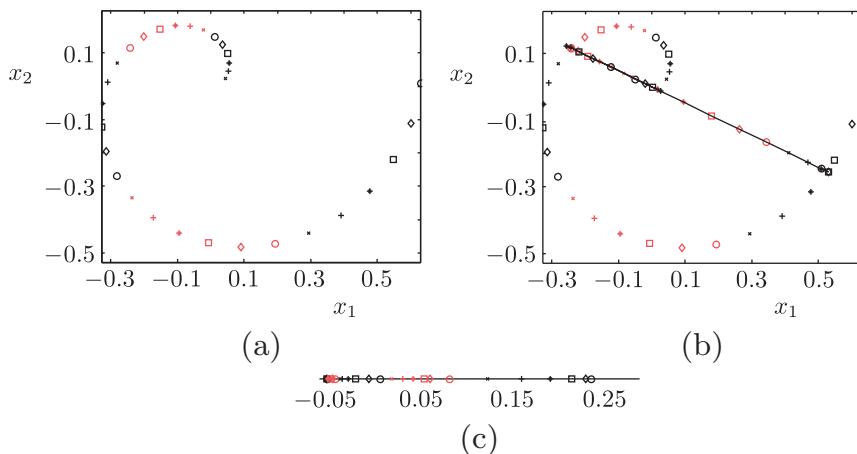
To study the performance of PCA for this case, where data lie on a nonlinear manifold, we first performed the eigendecomposition of the covariance matrix, estimated from the data set. The resulting eigenvalues are

$$\lambda_2 = 0.089 \quad \text{and} \quad \lambda_1 = 0.049.$$

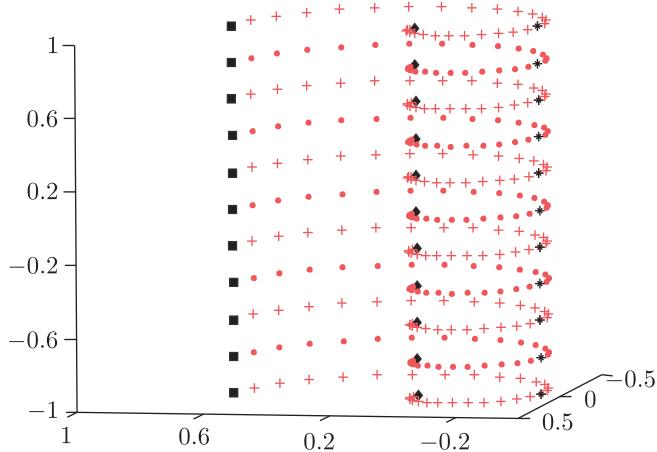
Observe that, the eigenvalues are comparable in size. Thus, if one would trust the “verdict” coming from PCA, the answer concerning the dimensionality of the data would be that it is equal to 2. Moreover, after projecting along the direction of the principle component (the straight line in Figure 19.18b), corresponding to λ_2 , neighboring information is lost because points from different locations are mixed together.

In the sequel, the Laplacian eigenmap technique for dimensionality reduction is employed, with $\epsilon = 0.2$ and $\sigma = \sqrt{0.5}$. The obtained results are shown in Figure 19.18c. Looking from right to left, we see that the Laplacian method nicely “unfolds” the spiral in a one-dimensional straight line. Furthermore, neighboring information is retained in this one-dimensional representation of the data. Black and red areas are succeeding each other in the right order, and also, observing the symbols, one can see that neighbors are mapped to neighbors.

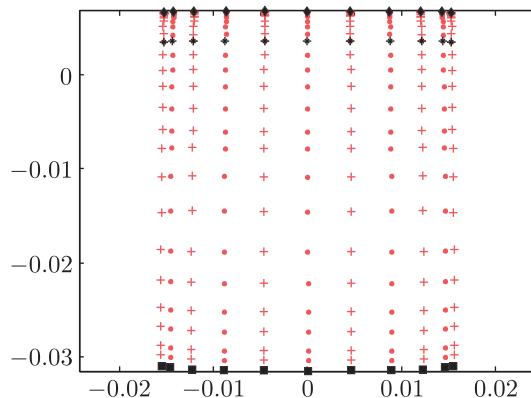
Example 19.8. Figure 19.19 shows samples from a three-dimensional spiral, parameterized as $x_1 = a\theta \cos \theta$, $x_2 = a\theta \sin \theta$, and sampled at $\theta = 0.5\pi, 0.7\pi, 0.9\pi, \dots, 2.05\pi$ (θ is expressed in radians), $a = 0.1$ and $x_3 = -1, -0.8, -0.6, \dots, 1$.

**FIGURE 19.18**

(a) A spiral of Archimedes in the two-dimensional space. (b) The previous spiral together with the projections of the sampled points on the direction of the first principle component, resulting from PCA. It is readily seen that neighboring information is lost after the projection and points corresponding to different parts of the spiral overlap. (c) The one-dimensional map of the spiral using the Laplacian method. In this case, the neighboring information is retained after the nonlinear projection and the spiral nicely unfolds to a one-dimensional line.

**FIGURE 19.19**

Samples from a three-dimensional spiral. One can think of it as a number of two-dimensional spirals one above the other. Different symbols have been used in order to track neighboring information.

**FIGURE 19.20**

Two-dimensional mapping of the spiral of Figure 19.19 using the Laplacian eigenmap method. The three-dimensional structure is unfolded to the two-dimensional space by retaining the neighboring information.

For illustration purposes and in order to keep track of the “identity” of each point, we have used red crosses and dots interchangeably, as we move upward in the x_3 dimension. Also, the first, the middle, and the last points for each level of x_3 are denoted by black “ \diamond ”, black “ \star ,” and black “ \square ”, respectively. Basically, all points at the same level lie on a two-dimensional spiral.

Figure 19.20 shows the two-dimensional mapping of the three-dimensional spiral using the Laplacian method for dimensionality reduction, with parameter values $\epsilon = 0.35$ and $\sigma = \sqrt{0.5}$. Comparing Figures 19.19 and 19.20, we see that all points corresponding to the same level x_3 are mapped across the same line, with the first point being mapped to the first one and so on. That is, as it was the case of the Example 19.7, the Laplacian method unfolds the three-dimensional spiral into a two-dimensional surface, while retaining neighboring information.

19.10 LOW-RANK MATRIX FACTORIZATION: A SPARSE MODELING PATH

The low-rank matrix factorization task has already been discussed from different perspectives. In this section, the task will be considered in a specific context; that of missing entries and/or in the presence of outliers. Such a focus is dictated by a number of more recent applications, especially in the framework of big data problems. To this end, sparsity-promoting arguments will be mobilized to offer a fresh look at this old problem. We are not going to delve into many details and our purpose is to highlight the main directions and methods, which have been considered.

19.10.1 MATRIX COMPLETION

To recapitulate some of the main findings in Chapters 9 and 10, let us consider a signal vector $s \in \mathbb{R}^l$, where only N of its components are observed and the rest are unknown. This is equivalent with sensing s via a sensing matrix having its N rows picked uniformly at random from the standard (canonical) basis

$\Phi = I$, where I is the $l \times l$ identity matrix. The question that was posed there was whether it is possible to recover s exactly based on these N components. From the theory presented in Chapter 9, we know that one can recover all the components of s , provided that s is sparse in some basis or dictionary, Ψ , which exhibits low mutual coherence with $\Phi = I$, and N is large enough, as has been pointed out in Section 9.9.

Inspired by the theoretical advances in compressed sensing, a question similar in flavor and with a prominent impact regarding practical applications was posed in [32]. Given an $l_1 \times l_2$ matrix M , assume that only $N \ll l_1 l_2$ among its entries are known. Concerning notation, we refer to a general matrix M , irrespective of how this matrix was formed. For example, it may correspond to an image array. The question now is whether one is able to recover the exact full matrix. This problem is widely known as *matrix completion* [32]. The answer, although it might come as a surprise, is “yes” with high probability, provided that (a) the matrix is *well structured* and complies with certain assumptions, (b) it has a *low rank*, $r \ll l$, where $l = \min(l_1, l_2)$, and (c) that N is large enough. Intuitively, this is plausible because a low-rank matrix is fully described in terms of a number of parameters (degrees of freedom), which is much smaller than its total number of entries. These parameters are revealed via its singular value decomposition (SVD)

$$M = \sum_{i=1}^r \sigma_i \mathbf{u}_i \mathbf{v}_i^T = U \begin{bmatrix} \sigma_1 & & O \\ & \ddots & \\ O & & \sigma_r \end{bmatrix} V^T, \quad (19.115)$$

where r is the rank of the matrix, $\mathbf{u}_i \in \mathbb{R}^{l_1}$ and $\mathbf{v}_i \in \mathbb{R}^{l_2}$, $i = 1, 2, \dots, r$, are the left and right orthonormal singular vectors, spanning the column and row spaces of M , respectively, σ_i , $i = 1, 2, \dots, r$, are the corresponding singular values, and $U = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_r]$, $V = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r]$.

Let σ_M denote the vector containing all the singular values of M , that is, $\sigma_M = [\sigma_1, \sigma_2, \dots, \sigma_r]^T$, then $\text{rank}(M) := \|\sigma_M\|_0$. Counting the parameters associated with the singular values and vectors in Eq. (19.115), it turns out that the number of degrees of freedom of a rank r matrix is equal to $d_M = r(l_1 + l_2) - r^2$ (Problem 19.8). When r is small, d_M is much smaller than l .

Let us denote with Ω the set of N pairs of indices, (i, j) , $i = 1, 2, \dots, l_1$, $j = 1, 2, \dots, l_2$, of the locations of the known entries of M , which have been sampled uniformly at random. Adopting a similar rationale to the one running across the backbone of sparsity-aware learning, one would attempt to recover M based on the following rank minimization problem,

$$\begin{aligned} \min_{\hat{M} \in \mathbb{R}^{l_1 \times l_2}} \quad & \|\sigma_{\hat{M}}\|_0 \\ \text{s.t.} \quad & \hat{M}(i, j) = M(i, j), \quad (i, j) \in \Omega. \end{aligned} \quad (19.116)$$

It turns out that, assuming that there exists a unique low-rank matrix having as elements the specific known entries, then the task in (19.116) leads to the exact solution [32]. However, compared to the case of sparse vectors, in the matrix completion problem the uniqueness issue gets much more involved. The following issues play a crucial part concerning the uniqueness of the task in (19.116).

1. If the number of known entries is lower than the degrees of freedom, that is, $N < d_M$, then there is no way to recover the missing entries whatsoever, because there is an infinite number of low-rank matrices consistent with the N observed entries.
2. Even if $N \geq d_M$, uniqueness is still not guaranteed. It is required that the N elements with indices in Ω are such that at least one entry per column and one entry per row are observed. Otherwise,

even a rank-1 matrix $M = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T$ cannot be recovered. This becomes clear with a simple example. Assume that M is a rank-1 matrix and that no entry in the first column as well as in the last row is observed. Then, because for this case $M(i,j) = \sigma_1 u_{1i} v_{1j}$, it is clear that no information concerning the first component of \mathbf{v}_1 as well as the last component of \mathbf{u}_1 is available; hence, it is impossible to recover these singular vector components, regardless of which method is used. As a consequence, the matrix cannot be completed. On the other hand, if the elements of Ω are picked at random and N is large enough, one can only hope that Ω is such as to comply with the requirement above; i.e., at least one entry per row and column to be observed, with high probability. It turns out that this problem resembles the famous theorem in probability theory known as the *coupon collector's* problem. According to this, at least $N = C_0 l \ln l$ entries are needed, where C_0 is a constant [126]. This is the information theoretic limit for exact matrix completion [34] of any low-rank matrix.

3. Even if points (1) and (2) above are fulfilled, uniqueness is still not guaranteed. In fact, not every low-rank matrix is liable to exact completion, regardless of the number and the positions of the observed entries. Let us demonstrate this via an example. Let one of the singular vectors be sparse. Assume, without loss of generality, that the third left singular vector, \mathbf{u}_3 , is sparse with sparsity level $k = 1$ and also that its nonzero component is the first one, that is, $u_{31} \neq 0$. The rest of \mathbf{u}_i and all \mathbf{v}_i are assumed to be dense. Let us return to the SVD for awhile in Eq. (19.115). Observe that the matrix M is written as the sum of r , $l_1 \times l_2$ matrices $\sigma_i \mathbf{u}_i \mathbf{v}_i^T$, $i = 1, \dots, r$. Thus, in this specific case where \mathbf{u}_3 is $k = 1$ sparse, the matrix $\sigma_3 \mathbf{u}_3 \mathbf{v}_3^T$ has zeros everywhere except for its first row. In other words, the information that $\sigma_3 \mathbf{u}_3 \mathbf{v}_3^T$ brings to the formation of M is concentrated in its first row only. This argument can also be viewed from another perspective; the entries of M obtained from any row except the first one, do not provide any useful information with respect to the values of the free parameters σ_3 , \mathbf{u}_3 , \mathbf{v}_3 . As a result, in this case, unless one incorporates extra information about the sparse nature of the singular vector, the entries from the first row that are missed are not recoverable, because the number of parameters concerning this row is larger than the available number of data.

Intuitively, when a matrix has dense singular vectors it is better rendered for exact completion as each one among the observed entries carries information associated with all the d_M parameters that fully describe it. To this end, a number of conditions, which evaluate the suitability of the singular vectors, have been established. The simplest one is given next [32]:

$$\|\mathbf{u}_i\|_\infty \leq \sqrt{\frac{\mu_B}{l_1}}, \quad \|\mathbf{v}_i\|_\infty \leq \sqrt{\frac{\mu_B}{l_2}}, \quad i = 1, \dots, r. \quad (19.117)$$

where μ_B is a bound parameter. In fact, μ_B is a measure of the coherence of matrix U (and similarly of V),⁹ (vis-à-vis the standard basis), defined as follows:

$$\mu(U) := \frac{l_1}{r} \max_{1 \leq i \leq l_1} \|P_U \mathbf{e}_i\|^2, \quad (19.118)$$

where P_U defines the orthogonal projection to subspace U and \mathbf{e}_i is the i th vector of the canonical basis. Note that when U results from SVD, then $\|P_U \mathbf{e}_i\|^2 = \|U^T \mathbf{e}_i\|^2$. In essence, coherence is an index quantifying the extent to which the singular vectors are correlated with the standard basis \mathbf{e}_i , $i = 1, 2, \dots, l$. The smaller the μ_B , the less “spiky” the singular vectors are likely to be, and the

⁹ This is a quantity different than the mutual-coherence already discussed in Section 9.6.1.

corresponding matrix is better suited for exact completion. Indeed, assuming for simplicity a square matrix M , that is, $l_1 = l_2 = l$, then if *any one* among the singular vectors is sparse having a single nonzero component only, then, taking into account that $\mathbf{u}_i^T \mathbf{u}_i = \mathbf{v}_i^T \mathbf{v}_i = 1$, this value will have magnitude equal to one and the bound parameter will take its largest value possible, that is, $\mu_B = l$. On the other hand, the smaller value that μ_B can get is 1, something that occurs when the components of *all* the singular vectors assume the same value (in magnitude). Note that in this case, due to the normalization, this common component value has magnitude $\frac{1}{l}$. Tighter bounds to a matrix coherence result from the more elaborate incoherence property [32, 141] and the strong incoherence property [34]. In all cases, the larger the bound parameter the larger the number of known entries becomes, which is required in order to guarantee uniqueness.

In section 19.10.3, the aspects of uniqueness will be discussed in the context of a real-life application.

The task formulated in (19.116) is of limited practical interest because it is an NP-hard task. Thus, borrowing the arguments used in Chapter 9, the ℓ_0 (pseudo)norm is replaced by a *convexly* relaxed counterpart of it, that is,

$$\begin{aligned} \min_{\hat{M} \in \mathbb{R}^{l_1 \times l_2}} \quad & \|\sigma_{\hat{M}}\|_1, \\ \text{s.t.} \quad & \hat{M}(i,j) = M(i,j), \quad (i,j) \in \Omega, \end{aligned} \quad (19.119)$$

where $\|\sigma_{\hat{M}}\|_1$, that is, the sum of the singular values, is referred to as the *nuclear norm* of the matrix \hat{M} , often denoted as $\|\hat{M}\|_*$. The nuclear norm minimization was proposed in [69] as a convex approximation of rank minimization, which can be cast as a semidefinite programming task.

Theorem 19.1. *Let M be an $l_1 \times l_2$ matrix of rank r , which is a constant much smaller than $l = \min(l_1, l_2)$, obeying (19.117). Suppose that we observe N entries of M with locations sampled uniformly at random. Then there is a positive constant C such that if*

$$N \geq C\mu_B^4 l \ln^2 l, \quad (19.120)$$

then M is the unique solution to the task in (19.119) with probability at least $1 - l^{-3}$.

There might be an ambiguity on how small the rank should be in order for the corresponding matrix to be characterized as “low rank.” More rigorously, a matrix is said to be of low rank if $r = \mathcal{O}(1)$, which means that r is a constant with no dependence (not even logarithmic), on l . Matrix completion is also possible for more general rank cases where, instead of the mild coherence property of (19.117), the incoherence and the strong incoherence properties [32, 34, 79, 141] are mobilized in order to get similar theoretical guarantees. The detailed exposition of these alternatives is beyond the scope of this book. In fact, Theorem 19.1 embodies the essence of the matrix completion task: with high probability, nuclear-norm minimization recovers all the entries of a low-rank matrix, M , with no error. More importantly, the number of entries, N , that the convexly relaxed problem needs is only by a logarithmic factor larger than the information theoretic limit, which, as it was mentioned before, equates to $C_0 l \ln l$. Moreover, similar to compressed sensing, robust matrix completion in the presence of noise is also possible as long as the request $\hat{M}(i,j) = M(i,j)$ in Eqs. (19.116) and (19.119) is replaced by $\|\hat{M}(i,j) - M(i,j)\|_2 \leq \epsilon$ [33]. Furthermore, the notion of matrix completion has also been extended to tensors, for example, [72, 155].

19.10.2 ROBUST PCA

The developments on matrix completion theory led, more recently, to the formulation and solution of another problem of high significance. To this end, the notation $\|M\|_1$, that is, the ℓ_1 norm of a matrix, is introduced and defined as the sum of the absolute values of its entries, that is, $\|M\|_1 = \sum_{i=1}^{l_1} \sum_{j=1}^{l_2} |M(i,j)|$. In other words, it acts on the matrix as if this were a long vector.

Assume now that M is expressed as the sum of a low-rank matrix, L , and a sparse matrix, S , that is, $M = L + S$. Consider the following convex minimization problem task, [35, 42, 176, 182], which is usually referred to as *principle component pursuit* (PCP),

$$\min_{\hat{L}, \hat{S}} \quad \| \sigma_M \|_1 + \lambda \| \hat{S} \|_1, \quad (19.121)$$

$$\text{s.t.} \quad \hat{L} + \hat{S} = M, \quad (19.122)$$

\hat{L} , \hat{S} are both $l_1 \times l_2$ matrices. It can be shown that solving the task in (19.121)–(19.121) recovers both L and S according to the following theorem. [35]:

Theorem 19.2. *The PCP recovers both L and S with probability at least $1 - cl_1^{-10}$, where c is a constant, provided that:*

1. *The support set Ω of S is uniformly distributed among all sets of cardinality N .*
2. *The number, k , of nonzero entries of S is relatively small, that is, $k \leq \rho l_1 l_2$, where ρ is a sufficiently small positive constant.*
3. *L obeys the incoherence property.*
4. *The regularization parameter, λ , is constant with value $\lambda = \frac{1}{\sqrt{l_2}}$,*
5. *$\text{rank}(L) \leq C \frac{l_2}{\ln^2 l_1}$, with C being a constant.*

In other words, based on *all* the entries of a matrix M , which is known to be the sum of two unknown matrices L and S , with the first one being of low-rank matrix and the second being sparse, then PCP recovers exactly, with probability almost 1, both L and S , irrespective of how large the magnitude of the entries of S are, provided that *both r and k are sufficiently small*.

The applicability of the previous task is very broad. For example, PCP can be employed in order to find a low-rank approximation of M . In contrast to the standard PCA (SVD) approach, PCP is robust and insensitive in the presence of outliers, as these are naturally modeled, via the presence of S . Note that outliers are sparse by their nature. For this reason, the above task is widely known as *robust PCA via nuclear norm minimization*. (More classical PCA techniques are known to be sensitive to outliers and a number of alternative approaches have in the past been proposed toward its robustification, for example, [91, 102].)

When PCP serves as a robust PCA approach, the matrix of interest is L and S accounts for the outliers. However, PCP estimates both L and S . As will be discussed soon, another class of applications are well accommodated when the focus of interest is turned to the sparse matrix S itself.

Remarks 19.8.

- Just as ℓ_1 -minimization is the tightest convex relaxation of the combinatorial ℓ_0 -minimization problem in sparse modeling, the nuclear-norm minimization is the tightest convex relaxation of the NP-hard rank minimization task. Besides the nuclear norm, other heuristics have also been proposed such as the log-determinant heuristic [69] and the max-norm [71].

- The nuclear norm as a rank minimization approach is the generalization of the trace-related cost, which is often used in the control community for the rank minimization of positive semidefinite matrices [125]. Indeed, when the matrix is symmetric and positive semidefinite, the nuclear norm of M is the sum of the eigenvalues and, thus, it is equal to the trace of M . Such problems arise when, for example, the rank minimization task refers to covariance matrices and positive semidefinite Toeplitz or Hankel matrices (see, e.g., [69]).
- Both matrix completion (19.119) and PCP (19.122) can be formulated as semidefinite programs and are solved based on interior-point methods. However, whenever the size of a matrix becomes large (e.g., 100×100), these methods are deemed to fail in practice due to excessive computational load and memory requirements. As a result, there is an increasing interest, which has propelled intensive research efforts, for the development of efficient methods to solve both optimization tasks, or related approximations, which scale well with large matrices. Many of these methods revolve around the philosophy of the iterative soft and hard thresholding techniques, as discussed in Chapter 9. However, in the current low-rank approximation setting, it is the singular values of the estimated matrix that are thresholded. As a result, in each iteration, the estimated matrix, after thresholding its singular values, tends to be of lower rank. The thresholding of the singular values is either imposed, such as in the case of the singular value thresholding (SVT) algorithm [30], or it results as a solution of regularized versions of (19.119) and (19.122) (see, e.g., [46, 167]). Moreover, algorithms inspired by greedy methods such as CoSaMP, have also been proposed (e.g., [117, 172]).
- Improved versions of PCP that allow for exact recovery even if some of the constraints of Theorem 19.2 are relaxed have also been developed (see, e.g., [73]). Fusions of PCP with matrix completion and compressed sensing are possible, in the sense that only a subset of the entries of M is available and/or linear measurements of the matrix in a compressed sensing fashion can be used instead of matrix entries, for example, [172, 177]. Moreover, stable versions of PCP dealing with noise have also been investigated, for example, [188].

19.10.3 APPLICATIONS OF MATRIX COMPLETION AND ROBUST PCA

The number of applications in which these techniques are involved is ever increasing and their extensive presentation is beyond the scope of this book. Next, some key applications are selectively discussed in order to reveal the potential of these methods and at the same time to assist the reader in better understanding the underlying notions.

Matrix completion

A typical application where the matrix completion problem arises is in the *collaborative filtering* task (e.g., [157]), which is essential for building up successful recommender systems. Let us consider that a group of individuals provide their ratings concerning products that they have enjoyed. Then, a matrix with ratings can be filled, where each row indexes a different individual and the columns index the products. As a popular example, take the case where the products are different movies. Inevitably, the associated matrix will be partially filled because it is not common that all customers have watched all the movies and submitted ratings for all of them. Matrix completion comes to provide an answer, potentially in the affirmative, to the following question: Can we predict the ratings that the users would give to films that they have not seen yet? This is the task of a recommender system in order to encourage

users to watch movies, which are likely to be of their preference. The exact objective of competition for the famous Netflix prize (<http://www.netflixprize.com/>) was the development of such a recommender system.

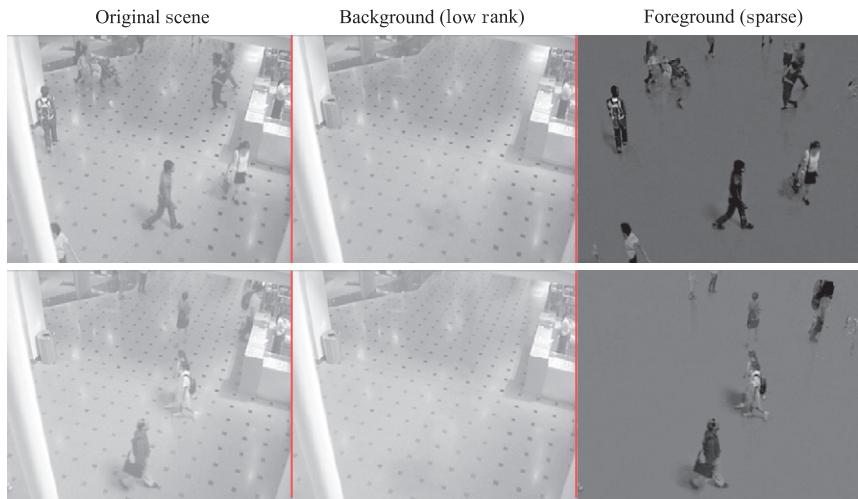
The aforementioned problem provides a good opportunity to build up our intuition about the matrix completion task. First, an individual's preferences or taste in movies are typically governed by a small number of factors, such as gender, the actors that appear in it, the continent of origin, and so on. As a result, a matrix fully filled with ratings is expected to be low rank. Moreover, it is clear that each user needs to have at least one movie rated in order to have any hope of filling out her/his ratings across all movies. The same is true for each movie. This requirement complies with the second assumption in [Section 19.10.1](#), concerning uniqueness; that is, one needs to know at least one entry per row and column. Finally, imagine a single user who rates movies with criteria that are completely different from those used by the rest of the users. One could, for example, provide ratings at random or depending on, let's say, the first letter of the movie title. The ratings of this particular user cannot be described in terms of the singular vectors that model the ratings of the rest of the users. Accordingly, for such a case, the rank of the matrix increases by one and the user's preferences will be described by an extra set of left and right singular vectors. However, the corresponding left singular vector will comprise a single nonzero component, at the place corresponding to the row dedicated to this user, and the right singular vector will comprise her/his ratings normalized to unit norm. Such a scenario complies with the third point concerning the uniqueness in the matrix completion problem, as previously discussed. Unless all the ratings of the specific user are known, the matrix cannot get fully completed.

Other applications of matrix completion includes system identification [120], recovering structure from motion [44], multitask learning [10], and sensor network localization [128].

Robust PCA/PCP

In the collaborative filtering task, robust PCA offers an extra attribute compared to matrix completion, which can be proved very crucial in practice. The users are allowed to even tamper with some of the ratings without affecting the estimation of the low-rank matrix. This seems to be the case whenever the rating process involves many individuals in an environment, which is not strictly controlled, because some of them occasionally are expected to provide ratings in an ad hoc, or even malicious manner.

One of the first applications of PCP was in video surveillance systems (e.g., [35]) and the main idea behind it appeared to be popular and extendable to a number of computer vision applications. Take the example of a camera recording a sequence of frames consisting of a merely static background and a foreground with a few moving objects, for example, vehicles and/or individuals. A common task in surveillance video is to extract from the background the foreground, in order, for example, to detect any activity or to proceed with further processing such as face recognition. Suppose the successive frames are converted to vectors in lexicographic order and then are placed as columns in a matrix M . Due to the background, even though this may slightly vary due, for example, to changes in illumination, successive columns are expected to be highly correlated. As a result, the background contribution to the matrix M can be modeled as an approximately low-rank matrix L . On the other hand, the objects in the foreground appear as “anomalies” and correspond to only a fraction of pixels in each frame; that is, to a limited number of entries in each column of M . Moreover, due to the motion of the foreground objects, the positions of these anomalies are likely to change from one column of M to the next. Therefore, they can be modeled as a sparse matrix S .

**FIGURE 19.21**

Background-Foreground separation via PCP.

Next, the above discussed philosophy is applied to a video acquired from a shopping mall surveillance camera, [121], with the corresponding PCP task being solved with a dedicated accelerated proximal gradient algorithm, [122]. The results are shown in Figure 19.21. In particular, two randomly selected frames are depicted together with the corresponding columns of the matrices L and S reshaped back to pictures.

19.11 A CASE STUDY: fMRI DATA ANALYSIS

In the brain, tasks involving action, perception, cognition, and so forth, are performed via the simultaneous activation of a number of the so-called *functional brain networks* (FBN), which are engaged in proper interactions in order to effectively execute the task. Such networks are usually related to low-level brain functions and they are defined as a number of *segregated* specialized small brain regions, potentially distributed over the whole brain. For each FBN, the involved segregated brain regions define the *spatial map*, which characterizes the specific FBN. Moreover, these brain regions, irrespective of their anatomical proximity or remoteness, exhibit *strong* functional connectivity, which is expressed as strong coherence in the activation timepatterns of these regions. Examples of such functional brain networks are the visual, sensorimotor, auditory, default-mode, dorsal attention, and executive control networks [139].

Functional magnetic resonance imaging (fMRI) [123] is a powerful noninvasive tool for detecting brain activity along time. Most commonly, it is based on *blood oxygenation level-dependent* (BOLD)

contrast, which translates to detecting localized changes in the hemodynamic flow of oxygenated blood in activated brain areas. This is achieved by exploiting the different magnetic properties of oxygen-saturated versus oxygen-desaturated hemoglobin. The detected fMRI signal is recorded in both the spatial (3-D) as well as the temporal (1-D) domain. The spatial domain is segmented with a 3-D grid to elementary cubes of edge size 3–5 mm, which are named *voxels*. Indicatively, a complete volume scan typically consists of $64 \times 64 \times 48$ voxels and it is acquired in one or two seconds, [123]. Relying on adequate postprocessing, which effectively compensates for possible time lags and other artifacts, [123], it is fairly accurate to assume that each acquisition is performed instantly. The, say l , in total voxel values, corresponding to a single scan, are collected in a flattened (row) 1-D vector, $\mathbf{x}_n \in \mathbb{R}^l$. Considering $n = 1, 2, \dots, N$, successive acquisitions, the full amount of data is collected in a data matrix $X \in \mathbb{R}^{N \times l}$. Thus, each column, $i = 1, 2, \dots, l$, of X represents the evolution in time of the values of the corresponding i th voxel. Each row, $n = 1, 2, \dots, N$, corresponds to the activation pattern, at the corresponding time n , over all l voxels.

The recorded voxel values result from the cumulative contribution of several FBNs, where each one of them is activated following certain time patterns, depending on the tasks that the brain is performing. The above can be mathematically modeled according to the following factorization of the data matrix:

$$X = \sum_{j=1}^m \mathbf{a}_j \mathbf{z}_j^T := AZ, \quad (19.123)$$

where $\mathbf{z}_j \in \mathbb{R}^l$ is a sparse vector of latent variables, representing the spatial map of the j th FBN having nonzero values only in positions that correspond to brain regions associated with the specific FBN, and $\mathbf{a}_j \in \mathbb{R}^N$ represents the activation *time course* of the respective FBN. The model assumes that m FBNs have been activated. In order to understand better the previous model, take as an example the extreme case where only one set of brain regions (one FBN) is activated. Then, matrix X is written as

$$X = \mathbf{a}_1 \mathbf{z}_1^T := \begin{bmatrix} a_1(1) \\ a_1(2) \\ \vdots \\ a_1(N) \end{bmatrix} \underbrace{[\dots, *, \dots, *, \dots, *, \dots,]}_{l \text{ (voxels)}},$$

where $*$ denotes a nonzero element (active voxel in the FBN) and the dots zero ones. Observe that according to this model, all nonzero elements in the n th row of X result from the nonzero elements of \mathbf{z}_1 multiplied by the *same* number, $a_1(n)$, $n = 1, 2, \dots, N$. If now two FBNs are active, the model for the data matrix becomes

$$X = \mathbf{a}_1 \mathbf{z}_1^T + \mathbf{a}_2 \mathbf{z}_2^T = [\mathbf{a}_1, \mathbf{a}_2] \begin{bmatrix} \mathbf{z}_1^T \\ \mathbf{z}_2^T \end{bmatrix}.$$

Obviously, for m FBNs, Eq. (19.123) results.

One of the major goals of the fMRI analysis is to detect, study, and characterize the different FBNs and to relate them to particular mental and physical activities. In order to achieve this, the subject (person) subjected to fMRI is presented with carefully designed experimental procedures, so that the activation of the FBNs will be as controlled as possible.

ICA has been successfully employed for fMRI unmixing, that is, for estimating matrices A and Z above. If we consider each column of X to be a realization of a random vector \mathbf{x} , the fMRI data generation mechanism can be modeled to follow the classical ICA latent model, that is, $\mathbf{x} = A\mathbf{s}$, where the components of \mathbf{s} are statistically independent and A is an unknown mixing matrix. The goal of ICA is to recover the unmixing matrix, W and Z . Matrix A is then obtained from W . The use of ICA in the fMRI task could be justified by the following argument. Nonzero elements of Z in the same column contribute to the formation of a single element of X , for each time instant, n , and correspond to different FBNs. Thus, they are assumed to correspond to two statistically independent sources.

As a result of the application of ICA on X , one hopes that each row of the obtained matrix Z could be associated with an FBN; that is, to a spatial activity map. Furthermore, the corresponding column of A could represent the respective time activation pattern.

This approach will be applied next for the case of the following experimental procedure, [57]: A visual pattern was presented to the subject, in which an 8 Hz reversing black and white checkerboard was shown intermittently in the left and right visual ends for 30 s at a time. This is a typical block design paradigm in fMRI, consisting of three different conditions to which the subject is exposed. Checkerboard on the left (red block) checkerboard on the right (black block) and no visual stimulus (white block). The subject was instructed to focus on the cross at the central during the full time of the experiment, Figure 19.22. More details about the scanning procedure and the preprocessing of the data can be found in [57]. The Group ICA of fMRI Toolbox (GIFT)¹⁰ simulation tool was used.

When ICA is performed on the obtained data set,¹¹ the aforementioned matrices Z and A are computed. Ideally, at least some of the rows of Z should constitute spatial maps of the true FBNs, and the corresponding columns of A should represent the activation patterns of the respective FBNs, which correspond to the specific experimentation procedure.

The news is good, as shown in Figure 19.23. In particular, Figures 19.23a and 19.23b show two time courses (columns of A). For each one of them, one section of the associated spatial map (corresponding

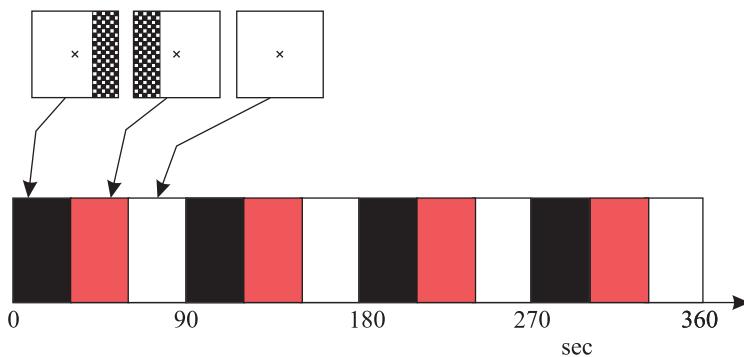
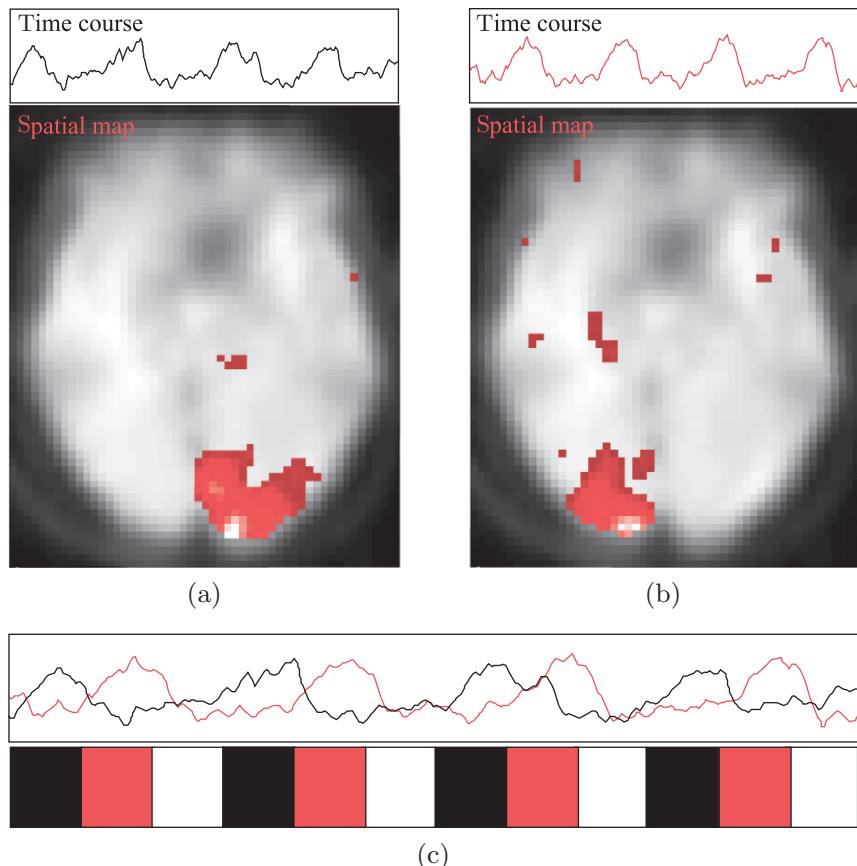


FIGURE 19.22

The fMRI experimental procedure used.

¹⁰ It can be obtained from <http://mialab.mrn.org/software/gift/>.

¹¹ The specific data set is provided as test data with GIFT.

**FIGURE 19.23**

The two time courses follow well the fMRI experimental setup used.

row of Z) is considered, which represents voxels of a slice in the brain. The areas that are activated (red) are those corresponding to the left (a) and to the right (b) visual cortex, which are the regions of the brain responsible for processing visual information. Activation of this part should be expected according to the characteristics of the specific experimentation procedure. More interesting, as it is seen in Figure 19.23c, the two activation patterns, as represented by the two time courses, follow closely the two different conditions; namely the checkerboard to be placed on the left or on the right from the point the subject is focusing on.

Besides ICA, alternative methods, discussed in this chapter, can also be used, which can better exploit the low-rank nature of X . Dictionary learning is a promising candidate leading to notably good results, see, for example, [11, 109, 171].

PROBLEMS

- 19.1** Show that the second principle component in PCA is given as the eigenvector corresponding to the second largest eigenvalue.
- 19.2** Show that the pair of directions, associated with CCA, which maximize the respective correlation coefficient, satisfy the following pair of relations,

$$\Sigma_{xy}\mathbf{u}_y = \lambda \Sigma_{xx}\mathbf{u}_x,$$

$$\Sigma_{yx}\mathbf{u}_x = \lambda \Sigma_{yy}\mathbf{u}_y.$$

- 19.3** Establish the arguments that verify the convergence of the k -SVD.
- 19.4** Prove that Eqs. (19.79) and (19.85) are the same.
- 19.5** Show that the ML PPCA tends to PCA as $\sigma^2 \rightarrow 0$.
- 19.6** Show Eqs. (19.87)–(19.88).
- 19.7** Show Eq. (19.98).
- 19.8** Show that the number of degrees of freedom of a rank r matrix is equal to $r(l_1 + l_2) - r^2$.

MATLAB Exercises

- 19.9** This exercise reproduces the results of Example 19.1. Download the faces from this book's website and read them one by one using the `imread.m` MATLAB function. Store them as columns in a matrix X . Then, compute and subtract the mean in order for the rows to become zero-mean.
- A direct way to compute the eigenvectors (eigenfaces) would be to use the `svd.m` MATLAB function in order to perform an SVD to the matrix X , that is $X = UDV^T$. In this way, the eigenfaces are the columns of U . However, this needs a lot of computational effort because X has too many rows. Alternatively, you can proceed as follows. First compute the product $A = X^T X$ and then the SVD of A (using the `SVD.m` MATLAB function) in order to compute the right singular vectors of X via $A = VD^2V^T$. Then calculate each eigenface according to $\mathbf{u}_i = \frac{1}{\sigma_i} X \mathbf{v}_i$, where σ_i is the i th singular value of the SVD. In the sequel, select one face at random in order to reconstruct it using the first 5, 30, 100, and 600 eigenvectors.
- 19.10** Recompute the eigenfaces as in the MATLAB Exercise 19.9, using all the face images apart from one, that you choose. Then reconstruct that face that did not take part in the computation of the eigenfaces, using the first 300 and 1000 eigenvectors. Is the reconstructed face anywhere close to the true one?
- 19.11** Download the fast ICA MATLAB software package from <http://research.ics.aalto.fi/ica/fastica/> in order to reproduce the results of the cocktail party example described in Subsection 19.5.5. The two voice and the music signals can be downloaded from this book's website and read using the `wavread.m` MATLAB function. Generate a random mixing matrix A , (3×3) and produce with it the 3 mixture signals. Each one of them simulates the signal received in each microphone. Then, apply FastICA in order to estimate the source signals. Use the MATLAB function `wavplay.m` in order to listen to the original signals, to the mixtures, and the recovered ones. Repeat the previous steps performing PCA instead of ICA and compare the results.
- 19.12** This exercise reproduces the dictionary learning-based denoising Example 19.5. The image depicting the boat can be obtained from this book's website. Moreover, the k -SVD either need

to be implemented according to [Section 19.6](#) or an implementation available on the web can be downloaded and used, e.g., <http://www.cs.technion.ac.il/~elad/software/>.

Then the next steps to be followed are: First, extract from the image all the possible sliding patches of size 12×12 using the im2col.m MATLAB function and store them as columns in a matrix X . Using this matrix, train an overcomplete dictionary, Ψ , of size, (144×196) , for 100 k -SVD iterations with $T_0 = 5$. For the first iteration, the initial dictionary atoms are drawn from a zero-mean Gaussian distribution and then normalized to unit norm. As a next step, de-noise each image patch separately. In particular, assuming that y_i is the i th patch reshaped in column vector use the OMP ([Section 10.2.1](#)) in order to estimate a sparse vector $\theta_i \in \mathbb{R}^{196}$ with $\|\theta_i\|_0 = 5$, such that $\|y_i - A\theta_i\|$ is small. Then, $\hat{y}_i = \Psi\theta_i$ is the i th de-noised patch.

Finally, average the values of the overlapped patches in order to form the full de-noised image.

- 19.13** Download one of the videos (they are provided in the form of a sequence of bitmap images) from http://perception.i2r.a-star.edu.sg/bk_model/bk_index.html.

In [Section 19.10.3](#), the “shopping center” bitmap image sequence has been used. Read one by one the bitmap images using the imread.m MATLAB function then convert them from color to grayscale using rgb2gray.m and finally store them as columns in a matrix X .

Download one of the MATLAB implementations of an algorithm performing the robust PCA task from http://perception.csl.illinois.edu/matrix-rank/sample_code.html.

The “Accelerated Proximal Gradient” method and the accompanied proximal_gradient_rPCA.m MATLAB function is a good and easy to use choice. Set $\lambda = 0.01$. Note, however, that depending on the video used, this regularization parameter might need to get fine-tuned.

REFERENCES

- [1] D. Achlioptas, F. McSherry, Fast computation of low rank approximations, in: Proceedings of the ACM STOC Conference, 2001, pp. 611-618.
- [2] T. Adali, H. Li, M. Novey, J.F. Cardoso, Complex ICA using nonlinear functions, IEEE Trans. Signal Process. 56 (9) (2008) 4356-4544.
- [3] T. Adali, M. Anderson, G.S. Fu, Diversity in independent component and vector analyses: Identifiability, algorithms, and applications in medical imaging, IEEE Signal Process. Mag. 31 (3) (2014) 18-33.
- [4] M. Aharon, M. Elad, A. Bruckstein, k -SVD: an algorithm for designing overcomplete dictionaries for sparse representation. IEEE Trans. Signal Process. 54 (11) (2006) 4311-4322.
- [5] T.W. Anderson, Asymptotic theory for principal component analysis, Ann. Math. Stat. 34 (1963) 122-148.
- [6] T.W. Anderson, An Introduction to Multivariate Analysis, second ed., John Wiley, New York, 1984.
- [7] M. Anderson, X.L. Li, T. Adali, Joint blind source separation with multivariate Gaussian model: Algorithms and performance analysis, IEEE Trans. Signal Process. 60 (4) (2012) 2049-2055.
- [8] C. Archambeau, F. Bach, Sparse probabilistic projections, in: D. Koller, D. Schuurmans, Y. Bengio, L. Bottou (Eds.), Neural Information Processing Systems, NIPS, Vancouver, Canada, 2008.
- [9] J. Arenas-García, K.B. Petersen, G. Camps-Valls, L.K. Hansen, Kernel multivariate analysis framework for supervised subspace learning, IEEE Signal Process. Mag. 30 (4) (2013) 16-29.
- [10] A. Argyriou, T. Evgeniou, M. Pontil, Multi-task feature learning, in: Advances in Neural Information Processing Systems, vol. 19, MIT Press, Cambridge, MA, 2007.
- [11] V. Abolghasemi, S. Ferdowsi, S. Sanei, Fast and incoherent dictionary learning algorithms with application to fMRI, Signal Image Video Process. 2013. DOI: 10.1007/s11760-013-0429-2.
- [12] G.I. Allen, Sparse and Functional Principal Components Analysis, 2013, arXiv preprint arXiv:1309.2895.

- [13] F.R. Bach, M.I. Jordan, Kernel independent component analysis, *J. Mach. Learn. Res.* 3 (2002) 1-48.
- [14] F. Bach, M. Jordan, A probabilistic interpretation of canonical correlation analysis, Technical Report 688, University of Berkeley, 2005.
- [15] E. Barshan, A. Ghodsi, Z. Azimifar, M.Z. Jahromi, Supervised principal component analysis: Visualization, classification and regression on subspaces and submanifolds, *Pattern Recognit.* 44 (2011) 1357-1371.
- [16] H.B. Barlow, Unsupervised learning, *Neural Comput.* 1 (1989) 295-311.
- [17] A.J. Bell, T.J. Sejnowski, An information maximization approach to blind separation and blind deconvolution, *Neural Comput.* 7 (1995) 1129-1159.
- [18] M. Belkin, P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, *Neural Comput.* 15 (6) (2003) 1373-1396.
- [19] E. Benetos, M. Kotti, C. Kotropoulos, Applying supervised classifiers based on non-negative matrix factorization to musical instrument classification, in: Proceedings IEEE International Conference on Multimedia and Expo, Toronto, Canada, 2006, pp. 2105-2108.
- [20] Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, M. Quimet, Out of sample extensions for LLE, Isomap, MDS, eigenmaps and spectral clustering, in: S. Thrun, L. Saul, B. Schölkopf (Eds.), *Advances in Neural Information Processing Systems Conference*, MIT Press, Cambridge, MA, 2004.
- [21] M. Berry, S. Dumais, G. O'Brien, Using linear algebra for intelligent information retrieval, *SIAM Rev.* 37 (1995) 573-595.
- [22] A. Beygelzimer, S. Kakade, J. Langford, Cover trees for nearest neighbor, in: Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, 2006.
- [23] C.M. Bishop, Variational principal components, in: Proceedings 9th International Conference on Artificial Neural Networks, ICANN, vol. 1, 1999, pp. 509-514.
- [24] M. Borga, Canonical correlation analysis: A tutorial, Technical Report, 2001, www.imt.liu.se/~magnus/cca/tutorial/tutorial.pdf.
- [25] M. Brand, Charting a manifold, in: *Advances in Neural Information Processing Systems*, vol. 15, MIT Press, Cambridge, MA, 2003, pp. 985-992.
- [26] J.-P. Brunet, P. Tamayo, T.R. Golub, J.P. Mesirov, Meta-genes and molecular pattern discovery using matrix factorization, *Proc. Natl. Acad. Sci.* 101 (2) (2004) 4164-4169.
- [27] C.J.C. Burges, Geometric methods for feature extraction and dimensional reduction: A guided tour, Technical Report MSR-TR-2004-55, Microsoft Research, 2004.
- [28] F. Bach, R. Jenatton, J. Mairal, G. Obozinski, Structured sparsity through convex optimization, *Stat. Sci.* 27 (4) (2012) 450-468.
- [29] D. Cai, X. He, Orthogonal locally preserving indexing, in: Proceedings 28th Annual International Conference on Research and Development in Information Retrieval, 2005.
- [30] J.-F. Cai, E.J. Candès, Z. Shen, A singular value thresholding algorithm for matrix completion, *SIAM J. Optim.* 20 (4) (2010) 1956-1982.
- [31] F. Camastra, Data dimensionality estimation methods: A survey, *Pattern Recognit.* 36 (2003) 2945-2954.
- [32] E.J. Candès, B. Recht, Exact matrix completion via convex optimization, *Found. Comput. Math.* 9 (6) (2009) 717-772.
- [33] E.J. Candès, P. Yaniv, Matrix completion with noise, *Proc. IEEE* 98 (6) (2010) 925-936.
- [34] E.J. Candès, T. Tao, The power of convex relaxation: Near-optimal matrix completion, *IEEE Trans. Inform. Theory.* 56 (3) (2010) 2053-2080.
- [35] E.J. Candès, X. Li, Y. Ma, J. Wright, Robust principal component analysis *J. ACM*, 58 (3) (2011) 1-37.
- [36] J.F. Cardoso, Infomax and maximum likelihood for blind source separation, *IEEE Signal Process. Lett.* 4 (1997) 112-114.
- [37] J.-F. Cardoso, Blind signal separation: Statistical principles, *Proc. IEEE* 9 (10) (1998) 2009-2025.

- [38] J.-F. Cardoso, High-order contrasts for independent component analysis, *Neural Comput.* 11 (1) (1999) 157-192.
- [39] L. Carin, R.G. Baraniuk, V. Cevher, D. Dunson, M.I. Jordan, G. Sapiro, M.B. Wakin, Learning low-dimensional signal models, *IEEE Signal Process. Mag.* 34 (2) (2011) 39-51.
- [40] V. Casteli, A. Thomasian, C.-S. Li, CSVD: Clustering and singular value decomposition for approximate similarity searches in high-dimensional space, *IEEE Trans. Knowl. Data Eng.* 15 (3) (2003) 671-685.
- [41] V. Cevher, P. Indyk, L. Carin, R.G. Baraniuk, Sparse signal recovery and acquisition with graphical models, *IEEE Signal Process. Mag.* 27 (6) (2010) 92-103.
- [42] V. Chandrasekaran, S. Sanghavi, P.A. Parrilo, A.S. Willsky, Rank-sparsity incoherence for matrix decomposition, *SIAM J. Optim.* 21 (2) (2011) 572-596.
- [43] C. Chatfield, A.J. Collins, *Introduction to Multivariate Analysis*, Chapman Hall, London, 1980.
- [44] P. Chen, D. Suter, Recovering the missing components in a large noisy low-rank matrix: Application to SFM, *IEEE Trans. Pattern Anal. Mach. Intell.* 26 (8) (2004) 1051-1063.
- [45] M. Chen, J. Silva, J. Paisley, C. Wang, D. Dunson, L. Carin, Compressive sensing on manifolds using nonparametric mixture of factor analysers: Algorithms and performance bounds, *IEEE Trans. Signal Process.* 58 (12) (2010) 6140-6155.
- [46] C. Chen, B. He, X. Yuan, Matrix completion via an alternating direction method, *IMA J. Numer. Anal.* 32 (2012) 227-245.
- [47] Y. Chi, Y.C. Eldar, R. Calderbank, PETRELS: Subspace estimation and tracking from partial observations, in: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 3301-3304.
- [48] S. Chouvardas, Y. Kopsinis, S. Theodoridis, An adaptive projected subgradient based algorithm for robust subspace tracking, in: Proc. International Conference on Acoustics Speech and Signal Processing, ICASSP, Florence, Italy, May 4-9, 2014.
- [49] S. Chouvardas, Y. Kopsinis, S. Theodoridis, Robust subspace tracking with missing entries: The set-theoretic approach, *IEEE Trans. Signal Process.*, to appear, 2015.
- [50] M. Chu, F. Diele, R. Plemmons, S. Ragni, Optimality, Computation and Interpretation of the Nonnegative Matrix Factorization, 2004, available at <http://www.wfu.edu/~plemmons>.
- [51] A. Cichoki, Unsupervised learning algorithms and latent variable models: PCA/SVD, CCA, ICA, NMF, in: R. Chellappa, S. Theodoridis (Eds.), *E-Reference for Signal Processing*, Academic Press, Boston, 2014.
- [52] N.M. Correa, T. Adalı, Y.-Q. Li, V.D. Calhoun, Canonical correlation analysis for group fusion and data inferences, *IEEE Signal Process. Mag.* 27 (4) (2010) 39-50.
- [53] P. Comon, Independent component analysis: A new concept, *Signal Process.* 36 (1994) 287-314.
- [54] P. Comon, C. Jutten, *Handbook of Blind Source Separation: Independent Component Analysis and Applications*, Academic Press, 2010.
- [55] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, second ed., MIT Press/McGraw-Hill, Cambridge, MA, 2001.
- [56] T. Cox, M. Cox, *Multidimensional Scaling*, Chapman & Hall, London, 1994.
- [57] V. Calhoun, T. Adali, G. Pearson, J. Pekar, A method for making group inferences from functional MRI data using independent component analysis, *Hum. Brain Mapp.* 14 (3) (2001) 140-151.
- [58] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, R. Harshman, Indexing by latent semantic analysis, *J. Soc. Inform. Sci.* 41 (1990) 391-407.
- [59] W.R. Dillon, M. Goldstein, *Multivariable Analysis Methods and Applications*, John Wiley, New York, 1984.
- [60] J.P. Dmochowski, P. Sajda, J. Dias, L.C. Parra, Correlated components of ongoing EEG point to emotionally laden attention—a possible marker of engagement? *Front. Hum. Neurosci.* 6 (2012). DOI: 10.3389/fnhum.2012.00112.

- [61] D.L. Donoho, C.E. Grimes, When does ISOMAP recover the natural parameterization of families of articulated images? Technical Report 2002-27, Department of Statistics, Stanford University, 2002.
- [62] D. Donoho, V. Stodden, When does nonnegative matrix factorization give a correct decomposition into parts? in: S. Thrun, L. Saul, B. Schölkopf (Eds.), *Advances in Neural Information Processing Systems*, MIT Press, Cambridge, MA, 2004.
- [63] S.C. Douglas, S. Amari, Natural gradient adaptation, in: S. Haykin (Ed.), *Unsupervised Adaptive Filtering, Part I: Blind Source Separation*, John Wiley & Sons, New York, 2000, pp. 13-61.
- [64] X. Doukopoulos, G.V. Moustakides, Fast and stable subspace tracking, *IEEE Trans. Signal Process.* 56 (4) (2008) 1452-1465.
- [65] V. De Silva, J.B. Tenenbaum, Global versus local methods in nonlinear dimensionality reduction, in: S. Becker, S. Thrun, K. Obermayer (Eds.), *Advances in Neural Information Processing Systems*, vol. 15, MIT Press, Cambridge, MA, 2003, pp. 721-728.
- [66] M. Elad, M. Aharon, Image denoising via sparse and redundant representations over learned dictionaries, *IEEE Trans. Image Process.* 15 (12) (2006) 3736-3745.
- [67] M. Elad, *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*, Springer, New York, 2010.
- [68] K. Engan, S.O. Aase, J.H.A. Husy, Multi-frame compression: theory and design, *Signal Process.* 80 (10) (2000) 2121-2140.
- [69] M. Fazel, H. Hindi, S. Boyd, Rank minimization and applications in system theory, in: *Proceedings American Control Conference*, vol. 4, 2004, pp. 3273-3278.
- [70] D.J. Field, What is the goal of sensory coding? *Neural Comput.* 6 (1994) 559-601.
- [71] R. Foygel, N. Srebro, Concentration-based guarantees for low-rank matrix reconstruction, in: *Proceedings, 24th Annual Conference on Learning Theory (COLT)*, 2011.
- [72] S. Gandy, B. Recht, I. Yamada, Tensor completion and low-n-rank tensor recovery via convex optimization, *Inverse Prob.* 27 (2) (2011) 1-19.
- [73] A. Ganesh, J. Wright, X. Li, E.J. Candès, Y. Ma, Dense error correction for low-rank matrices via principal component pursuit, in: *Proceedings IEEE International Symposium on Information Theory*, 2010, pp. 1513-1517.
- [74] Z. Ghahramani, M. Beal, Variational inference for Bayesian mixture of factor analysers, in: *Advances in Neural Information Processing Systems*, vol. 12, MIT Press, Cambridge, MA, 2000, pp. 449-455.
- [75] M. Girolami, *Self-organizing Neural Networks, Independent Component Analysis and Blind Source Separation*, Springer-Verlag, New York, 1999.
- [76] M. Girolami, A variational method for learning sparse and overcomplete representations, *Neural Comput.* 13 (2001) 2517-2532.
- [77] G.H. Golub, C.F. Van Loan, *Matrix Computations*, Johns Hopkins Press, Baltimore, 1989.
- [78] S. Gould, *The Mismeasure of Man*, second ed., Norton, New York, 1981.
- [79] D. Gross, Recovering low-rank matrices from few coefficients in any basis, *IEEE Trans. Inform. Theory* 57 (3) (2011) 1548-1566.
- [80] J. He, L. Balzano, J. Lui, Online robust subspace tracking from partial information, 2011, arXiv preprint arXiv:1109.3827.
- [81] J. Ham, D.D. Lee, S. Mika, B. Schölkopf, A kernel view of the dimensionality reduction of manifolds, in: *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004, pp. 369-376.
- [82] D.R. Hardoon, S. Szedmak, J. Shawe-Taylor, Canonical correlation analysis: an overview with application to learning methods, *Neural Comput.* 16 (2004) 2639-2664.
- [83] D.R. Hardoon, J. Shawe-Taylor, Sparse canonical correlation analysis, *Mach. Learn.* 83 (3) (2011) 331-353.
- [84] S. Haykin, *Neural Networks: A Comprehensive Foundation*, second ed., Prentice Hall, Upper Saddle River, NJ, 1999.

- [85] J. He, L. Balzano, J. Lui, Online robust subspace tracking from partial information, 2011, arXiv preprint arXiv:1109.3827.
- [86] J. Hérault, C. Jouten, B. Ans, Détection de grandeurs primitives dans un message composite par une architecture de calcul neuroimimétique en apprentissage non supervisé, in: Actes du Xème colloque GRETSI, Nice, France, 1985, pp. 1017-1022.
- [87] N. Hjort, C. Holmes, P. Muller, S. Walker, Bayesian Nonparametrics, Cambridge University Press, Cambridge, 2010.
- [88] H. Hotelling, Analysis of a complex of statistical variables into principal components, *J. Educ. Psychol.* 24 (1933) 417-441.
- [89] H. Hotelling, Relations between two sets of variates, *Biometrika* 28 (34) (1936) 321-377.
- [90] P.J. Huber, Projection pursuit, *Ann. Stat.* 13 (2) (1985) 435-475.
- [91] M. Hubert, P.J. Rousseeuw, K. Vanden Branden, ROBPCA: a new approach to robust principal component analysis, *Technometrics* 47 (1) (2005) 64-79.
- [92] A. Hyvärinen, Fast and robust fixed-point algorithms for independent component analysis, *IEEE Trans. Neural Netw.* 10 (3) (1999) 626-634.
- [93] A. Hyvärinen, J. Karhunen, E. Oja, Independent Component Analysis, John Wiley, New York, 2001.
- [94] X. He, P. Niyogi, Locally preserving projections, in: Proceedings Advances in Neural Information Processing Systems Conference, 2003.
- [95] B.G. Huang, M. Ramesh, T. Berg, Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments, Technical Report, University of Massachusetts, Amherst, No. 07-49, 2007.
- [96] R. Jenssen, Kernel entropy component analysis, *IEEE Trans. Pattern Anal. Mach. Intell.* 32 (5) (2010) 847-860.
- [97] J.E. Jackson, A User's Guide to Principle Components, John Wiley, New York, 1991.
- [98] I. Jolliffe, Principal Component Analysis, Springer-Verlag, New York, 1986.
- [99] M.C. Jones, R. Sibson, What is projection pursuit? *J. R. Stat. Soc. A* 150 (1987) 1-36.
- [100] C. Jutten, J. Herault, Blind separation of sources, Part I: an adaptive algorithm based on neuromimetic architecture, *Signal Process.* 24 (1991) 1-10.
- [101] C. Jutten, Source separation: From dusk till dawn, in: Proceedings 2nd International Workshop on Independent Component Analysis and Blind Source Separation, ICA'2000, Helsinki, Finland, 2000, pp. 15-26.
- [102] J. Karhunen, J. Joutsensalo, Generalizations of principal component analysis, optimization problems, and neural networks, *Neural Netw.* 8 (4) (1995) 549-562.
- [103] J. Kettenring, Canonical analysis of several sets of variables, *Biometrika* 58 (3) (1971) 433-451.
- [104] M.E. Khan, M. Marlin, G. Bouchard, K.P. Murphy, Variational bounds for mixed-data factor analysis, in: J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, A. Culotta (Eds.), Neural Information Processing Systems, NIPS, Vancouver, Canada, 2010.
- [105] P. Kidmose, D. Looney, M. Ungstrup, M.L. Rank, D.P. Mandic, A study of evoked potentials from ear-EEG, *IEEE Trans. Biomed. Eng.* 60 (10) (2013) 2824-2830.
- [106] A. Klami, S. Virtanen, S. Kaski, Bayesian canonical correlation analysis, *J. Mach. Learn. Res.* 14 (2013) 965-1003.
- [107] O.W. Kwon, T.W. Lee, Phoneme recognition using the ICA-based feature extraction and transformation, *Signal Process.* 84 (6) (2004) 1005-1021.
- [108] S.-Y. Kung, K.I. Diamantaras, J.-S. Taur, Adaptive principal component extraction (APEX) and applications, *IEEE Trans. Signal Process.* 42 (5) (1994) 1202-1217.
- [109] Y. Kopsinis, H. Georgiou, S. Theodoridis, fMRI unmixing via properly adjusted dictionary learning, in: Proceedings of the 20th European Signal Processing Conference (EUSIPCO), 2012, pp. 61-65.

- [110] P.L. Lai, C. Fyfe, Kernel and nonlinear canonical correlation analysis, *Int. J. Neural Syst.* 10 (5) (2000) 365-377.
- [111] S. Lafon, A.B. Lee, Diffusion maps and coarse-graining: a unified framework for dimensionality reduction, graph partitioning and data set parameterization, *IEEE Trans. Pattern Anal. Mach. Intell.* 28 (9) (2006) 1393-1403.
- [112] L.D. Lathauer, Signal Processing by Multilinear Algebra, Ph.D. Thesis, Faculty of Engineering, K.U. Leuven, Belgium, 1997.
- [113] T.W. Lee, Independent Component Analysis: Theory and Applications, Kluwer, Boston, MA, 1998.
- [114] M.H.C. Law, A.K. Jain, Incremental nonlinear dimensionality reduction by manifold learning, *IEEE Trans. Pattern Anal. Mach. Intell.* 28 (3) (2006) 377-391.
- [115] D.D. Lee, S. Seung, Learning the parts of objects by nonnegative matrix factorization, *Nature* 401 (1999) 788-791.
- [116] J.A. Lee, M. Verleysen, Nonlinear Dimensionality Reduction, Springer, New York, 2007.
- [117] K. Lee, Y. Bresler, ADMiRA: atomic decomposition for minimum rank approximation, *IEEE Trans. Inform. Theory* 56 (9) (2010) 4402-4416.
- [118] S. Lesage, R. Gribonval, F. Bimbot, L. Benaroya, Learning unions of orthonormal bases with thresholded singular value decomposition, in: *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2005.
- [119] M.S. Lewicki, T.J. Sejnowski, Learning overcomplete representations, *Neural Comput.* 12 (2000) 337-365.
- [120] Z. Liu, L. Vandenberghe, Interior-Point Method for Nuclear Norm Approximation with Application to System Identification, *SIAM J. Matrix Anal. Appl.* 31 (3) (2010) 1235-1256.
- [121] L. Li, W. Huang, I.-H. Gu, Q. Tian, Statistical modeling of complex backgrounds for foreground object detection, *IEEE Trans. Image Process.* 13 (11) (2004) 1459-1472.
- [122] Z. Lin, A. Ganesh, J. Wright, L. Wu, M. Chen, Y. Ma, Fast convex optimization algorithms for exact recovery of a corrupted low-rank matrix, in: *Intl. Workshop on Comp. Adv. in Multi-Sensor Adapt. Processing*, Aruba, Dutch Antilles, 2009.
- [123] M.A. Lindquist, The statistical analysis of fMRI data, *Stat. Sci.* 23 (4) (2008) 439-464.
- [124] G. Mateos, G.B. Giannakis, Robust PCA as bilinear decomposition with outlier-sparsity regularization, *IEEE Trans. Signal Process.* 60 (2012) 5176-5190.
- [125] M. Mesbahi, G.P. Papavassiliopoulos, On the rank minimization problem over a positive semidefinite linear matrix inequality, *IEEE Trans. Autom. Control* 42 (2) (1997) 239-243.
- [126] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, Cambridge, 1995.
- [127] L. Mackey, Deflation methods for sparse PCA, in: D. Koller, D. Schuurmans, Y. Bengio, L. Bottou (Eds.), *Advances in Neural Information Processing Systems*, vol. 21, 2009, pp. 1017-1024.
- [128] G. Mao, B. Fidan, B.D.O. Anderson, Wireless sensor network localization techniques, *Comput. Netw.* 51 (10) (2007) 2529-2553.
- [129] M. Mardani, G. Mateos, G.B. Giannakis, Subspace Learning and Imputation for Streaming Big Data Matrices and Tensors, 2014, arXiv preprint arXiv:1404.4667.
- [130] J. Mairal, M. Elad, G. Sapiro, Sparse Representation for Color Image Restoration, *IEEE Trans. Image Process.* 17 (1) (2008) 53-69.
- [131] J. Mairal, F. Bach, J. Ponce, G. Sapiro, Online learning for matrix factorization and sparse coding, *J. Mach. Learn. Res.* 11 (2010).
- [132] M. Novey, T. Adali, Complex ICA by negentropy maximization, *IEEE Trans. Neural Netw.* 19 (4) (2008) 596-609.
- [133] M.A. Nicolaou, S. Zafeiriou, M. Pantic, A unified framework for probabilistic component analysis, in: *European Conference Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD'14)*, Nancy, France, 2014.

- [134] B.A. Olshausen, B.J. Field, Sparse coding with an overcomplete basis set: a strategy employed by v1, *Vis. Res.* 37 (1997) 3311-3325.
- [135] P. Paatero, U. Tapper, R. Aalto, M. Kulmala, Matrix factorization methods for analysis diffusion battery data, *J. Aerosol Sci.* 22 (Supplement 1) (1991) 273-276.
- [136] P. Paatero, U. Tapper, Positive matrix factor model with optimal utilization of error, *Environmetrics* 5 (1994) 111-126.
- [137] K. Pearson, On lines and planes of closest fit to systems of points in space, in: *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science, Sixth Series*, vol. 2, 1901, pp. 559-572.
- [138] A.T. Poulsen, S. Kamronn, L.C. Parra, L.K. Hansen, Bayesian correlated component analysis for inference of joint EEG activation, in: *4th International Workshop on Pattern Recognition in Neuroimaging*, 2014.
- [139] V. Perlberg, G. Marrelec, Contribution of exploratory methods to the investigation of extended large-scale brain networks in functional MRI: methodologies, results, and challenges, *Int. J. Biomed. Imaging* 2008 (2008) 1-14.
- [140] H. Qui, E.R. Hancock, Clustering and embedding using commute times, *IEEE Trans. Pattern Anal. Mach. Intell.* 29 (11) (2007) 1873-1890.
- [141] B. Recht, A simpler approach to matrix completion, *J. Mach. Learn. Res.* 12 (2011) 3413-3430.
- [142] R. Rosipal, L.J. Trejo, Kernel partial least squares regression in reproducing kernel Hilbert spaces, *J. Mach. Learn. Res.* 2 (2001) 97-123.
- [143] R. Rosipal, N. Krämer, Overview and recent advances in partial least squares, in: C. Saunders, M. Grobelnik, S. Gunn, J. Shawe-Taylor (Eds.), *Subspace, Latent Structure and Feature Selection*, Springer, New York, 2006.
- [144] S. Roweis, EM algorithms for PCA and SPCA, in: M.I. Jordan, M.J. Kearns, S.A. Solla (Eds.), *Advances in Neural Information Processing Systems*, vol. 10, MIT Press, Cambridge, MA, 1998, pp. 626-632.
- [145] S.T. Roweis, L.K. Saul, Nonlinear dimensionality reduction by locally linear embedding, *Science* 290 (2000) 2323-2326.
- [146] D.B. Rubin, D.T. Thayer, EM algorithm for ML factor analysis, *Psychometrika* 47 (1) (1982) 69-76.
- [147] C.A. Rencher, *Multivariate Statistical Inference and Applications*, John Wiley & Sons, New York, 2008.
- [148] S. Sanei, *Adaptive Processing of Brain Signals*, John Wiley, New York, 2013.
- [149] L.K. Saul, S.T. Roweis, An introduction to locally linear embedding, <http://www.cs.toronto.edu/~roweis/lle/papers/lleintro.pdf>.
- [150] N. Sebro, T. Jaakola, Weighted low-rank approximations, in: *Proceedings of the ICML Conference*, 2003, pp. 720-727.
- [151] B. Schölkopf, A. Smola, K.R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, *Neural Comput.* 10 (1998) 1299-1319.
- [152] F. Seidel, C. Hage, M. Kleinsteuber, pROST: A smoothed ℓ_p -norm robust online subspace tracking method for background subtraction in video, in: *Machine Vision and Applications*, 2013, pp. 1-14.
- [153] F. Sha, L.K. Saul, Analysis and extension of spectral methods for nonlinear dimensionality reduction, in: *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany, 2005.
- [154] Y. Shuicheng, D. Xu, B. Zhang, H.-J. Zhang, Q. Yang, S. Lin, Graph embedding and extensions: a general framework for dimensionality reduction, *IEEE Trans. Pattern Anal. Mach. Intell.* 29 (1) (2007) 40-51.
- [155] M. Signoretto, R. Van de Plas, B. De Moor, J.A.K. Suykens, Tensor versus matrix completion: a comparison with application to spectral data, *IEEE Signal Process. Lett.* 18 (7) (2011) 403-406.
- [156] P. Smaragdis, J.C. Brown, Nonnegative matrix factorization for polyphonic music transcription, in: *Proceedings IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2003.
- [157] X. Su, T.M. Khoshgoftaar, A survey of collaborative filtering techniques, *Adv. Artif. Intell.* 2009 (2009) 1-19.

- [158] S. Sra, I.S. Dhillon, Non-negative matrix approximation: algorithms and applications, Technical Report TR-06-27, University of Texas at Austin, 2006.
- [159] C. Spearman, The proof and measurement of association between two things, *Am. J. Psychol.* 100 (3-4) (1987) 441-471 (republished).
- [160] G.W. Stewart, An updating algorithm for subspace tracking, *IEEE Trans. Signal Process.* 40 (6) (1992) 1535-1541.
- [161] A. Szymkowiak-Have, M.A. Girolami, J. Larsen, Clustering via kernel decomposition, *IEEE Trans. Neural Netw.* 17 (1) (2006) 256-264.
- [162] J. Sun, S. Boyd, L. Xiao, P. Diaconis, The fastest mixing Markov process on a graph and a connection to a maximum variance unfolding problem, *SIAM Rev.* 48 (4) (2006) 681-699.
- [163] J.B. Tenenbaum, V. De Silva, J.C. Langford, A global geometric framework for dimensionality reduction, *Science* 290 (2000) 2319-2323.
- [164] S. Theodoridis, K. Koutroumbas, *Pattern Recognition*, fourth ed., Academic Press, Boston, 2009.
- [165] M.E. Tipping, C.M. Bishop, Probabilistic principal component analysis, *J. R. Stat. Soc. B* 21 (3) (1999) 611-622.
- [166] M.E. Tipping, C.M. Bishop, Mixtures probabilistic principal component analysis, *Neural Comput.* 11 (2) (1999) 443-482.
- [167] K.C. Toh, S. Yun, An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems, *Pac. J. Optim.* 6 (2010) 615-640.
- [168] J.A. Tropp, Literature survey: Nonnegative matrix factorization, Unpublished note, 2003, <http://www.personal.umich.edu/~jtropp/>.
- [169] C.G. Tsinos, A.S. Lalos, K. Berberidis, Sparse subspace tracking techniques for adaptive blind channel identification in OFDM systems, in: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 3185-3188.
- [170] N. Ueda, R. Nakano, Z. Ghahramani, G.E. Hinton, SMEM algorithm for mixture models, *Neural Comput.* 12 (9) (2000) 2109-2128.
- [171] G. Varoquaux, A. Gramfort, F. Pedregosa, V. Michel, B. Thirion, Multi-subject dictionary learning to segment an atlas of brain spontaneous activity, in: *Information Processing in Medical Imaging*, Springer, Berlin/Heidelberg.
- [172] A.E. Waters, A.C. Sankaranarayanan, R.G. Baraniuk, SpaRCS: recovering low-rank and sparse matrices from compressive measurements, in: *Advances in Neural Information Processing Systems (NIPS)*, Granada, Spain, 2011.
- [173] K.Q. Weinberger, L.K. Saul, Unsupervised learning of image manifolds by semidefinite programming, in: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, Washington, DC, USA, 2004, pp. 988-995.
- [174] J. Westerhuis, T. Kourti, J. MacGregor, Analysis of multiblock and hierarchical PCA and PLS models, *J. Chemometr.* 12 (1998) 301-321.
- [175] H. Wold, Nonlinear estimation by iterative least squares procedures, in: F. David (Ed.), *Research Topics in Statistics*, John Wiley, New York, 1966, pp. 411-444.
- [176] J. Wright, Y. Peng, Y. Ma, A. Ganesh, S. Rao, Robust principal component analysis: exact recovery of corrupted low-rank matrices by convex optimization, in: *Neural Information Processing Systems (NIPS)*, 2009.
- [177] J. Wright, A. Ganesh, K. Min, Y. Ma, Compressive Principal Component Pursuit, 2012, arXiv:1202.4596.
- [178] D. Weenink, Canonical Correlation Analysis, Institute of Phonetic Sciences, University of Amsterdam, Proceedings, vol. 25, 2003, pp. 81-99.
- [179] D.M. Witten, R. Tibshirani, T. Hastie, A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis, *Biostatistics* 10 (3) (2009) 515-534.
- [180] L. Wolf, T. Hassner, Y. Taigman, Effective unconstrained face recognition by combining multiple descriptors and learned background statistics, *IEEE Trans. Pattern Anal. Mach. Intell.* 33 (10) (2011) 1978-1990.

- [181] W. Xu, X. Liu, Y. Gong, Document clustering based on nonnegative matrix factorization, in: Proceedings 26th Annual International ACM SIGIR Conference, ACM Press, New York, 2003, pp. 263-273.
- [182] H. Xu, C. Caramanis, S. Sanghavi, Robust PCA via outlier pursuit, *IEEE Trans. Inform. Theory* 58 (5) (2012) 3047-3064.
- [183] J. Ye, Generalized low rank approximation of matrices, in: Proceedings of the 21st International Conference on Machine Learning, Banff, Alberta, Canada, 2004, pp. 887-894.
- [184] S.K. Yu, V. Yu, K.H.-P. Tresp, M. Wu, Supervised probabilistic principal component analysis, in: Proceedings International Conference on Knowledge Discovery and Data Mining, 2006.
- [185] M. Yaghoobi, T. Blumensath, M.E. Davies, Dictionary learning for sparse approximations with the majorization method, *IEEE Trans. Signal Process.* 57 (6) (2009) 2178-2191.
- [186] B. Yang, Projection approximation subspace tracking, *IEEE Trans. Signal Process.* 43 (1) (1995) 95-107.
- [187] S. Zafeiriou, A. Tefas, I. Buciu, I. Pitas, Exploiting discriminant information in non-negative matrix factorization with application to frontal face verification, *IEEE Trans. Neural Netw.* 17 (3) (2006) 683-695.
- [188] Z. Zhou, X. Li, J. Wright, E.J. Candès, Y. Ma, Stable principal component pursuit, in: Proceedings, IEEE International Symposium on Information Theory, 2010, pp. 1518-1522.
- [189] H. Zou, T. Hastie, R. Tibshirani, Sparse principal component analysis, *J. Comput. Graph. Stat.* 15 (2) (2006) 265-286.

This page intentionally left blank

LINEAR ALGEBRA

A

CHAPTER OUTLINE

A.1 Properties of Matrices	1023
<i>Matrix inversion lemmas</i>	<i>1024</i>
<i>Matrix derivatives</i>	<i>1024</i>
A.2 Positive Definite and Symmetric Matrices	1025
A.3 Wirtinger Calculus	1026
References.....	1027

A.1 PROPERTIES OF MATRICES

Let A , B , C , and D be matrices of appropriate sizes. Invertibility is always assumed, whenever a matrix inversion is performed. The following properties hold true.

- $(AB)^T = B^T A^T$.
- $(AB)^{-1} = B^{-1}A^{-1}$.
- $(A^T)^{-1} = (A^{-1})^T$.
- $\text{trace}\{AB\} = \text{trace}\{BA\}$.
- From the previous, we readily get

$$\text{trace}\{ABC\} = \text{trace}\{CAB\} = \text{trace}\{BCA\}.$$

- $\det(AB) = \det(A)\det(B)$, where $\det(\cdot)$ denotes the determinant of a square matrix. As a consequence, the following is also true.
- $\det(A^{-1}) = \frac{1}{\det(A)}$.
- Let A and B be two $m \times l$ matrices. Then

$$\det(I_m + AB^T) = \det(I_l + A^T B).$$

A by-product is the following

$$\det(I + ab^T) = 1 + a^T b,$$

where a , $b \in \mathbb{R}^l$.

Matrix inversion lemmas

- Woodbury's identity:

$$(A + BD^{-1}C)^{-1} = A^{-1} - A^{-1}B(D + CA^{-1}B)^{-1}CA^{-1}.$$

- $(I + AB)^{-1}A = A(I + BA)^{-1}$.
- $(A^{-1} + B^T C^{-1}B)^{-1}B^T C^{-1} = AB^T(BAB^T + C)^{-1}$.
- The following two inversion lemmas for partitioned matrices are particularly useful:

$$\begin{bmatrix} A & D \\ C & B \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + E\Delta^{-1}F & -E\Delta^{-1} \\ -\Delta^{-1}F & \Delta^{-1} \end{bmatrix}$$

where $\Delta := B - CA^{-1}D$, $E := A^{-1}D$, $F := CA^{-1}$.

Also

$$\begin{bmatrix} A & D \\ C & B \end{bmatrix}^{-1} = \begin{bmatrix} \Delta^{-1} & -\Delta^{-1}E \\ -F\Delta^{-1} & B^{-1} + F\Delta^{-1}E \end{bmatrix},$$

where $\Delta := A - DB^{-1}C$, $E := DB^{-1}$, $F := B^{-1}C$. Matrix Δ is also known as the *Schur complement*.

For complex matrices, the transposition becomes the Hermitian one.

Matrix derivatives

- $\frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \mathbf{a}$.
- $\frac{\partial \mathbf{x}^T A \mathbf{x}}{\partial \mathbf{x}} = (A + A^T)\mathbf{x}$,

which becomes $2A\mathbf{x}$ if A is symmetric.

- $\frac{\partial (AB)}{\partial \mathbf{x}} = \frac{\partial A}{\partial \mathbf{x}}B + A\frac{\partial B}{\partial \mathbf{x}}$.
- $\frac{\partial A^{-1}}{\partial \mathbf{x}} = -A^{-1}\frac{\partial A}{\partial \mathbf{x}}A^{-1}$.
- $\frac{\partial \ln|A|}{\partial \mathbf{x}} = \text{trace}\{A^{-1}\frac{\partial A}{\partial \mathbf{x}}\}$,

where $|\cdot|$ denotes the determinant, and matrices A and B are functions of x .

- $\frac{\partial \text{trace}\{AB\}}{\partial A} = B^T$,

where $\left[\frac{\partial}{\partial A}\right]_{ij} := \frac{\partial}{\partial A(i,j)}$.

- $\frac{\partial \text{trace}\{A^T B\}}{\partial A} = B$.
- $\frac{\partial \text{trace}\{ABA^T\}}{\partial A} = A(B + B^T)$.
- $\frac{\partial \ln|A|}{\partial A} = (A^T)^{-1}$.
- $\frac{\partial A\mathbf{x}}{\partial \mathbf{x}} = A^T$,

where by definition $\left[\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right]_{ij} = \frac{\partial y_i}{\partial x_j}$.

More on matrix identities can collectively be found in [2].

A.2 POSITIVE DEFINITE AND SYMMETRIC MATRICES

- An $l \times l$ real symmetric matrix A is called *positive definite* if, for every nonzero vector \mathbf{x} , the following is true:

$$\mathbf{x}^T A \mathbf{x}. \quad (\text{A.1})$$

If equality with zero is allowed, A is called *positive semidefinite*. The definition is extended to complex Hermitian symmetric matrices, A , if $\forall \mathbf{x} \in \mathbb{C}$,

$$\mathbf{x}^H A \mathbf{x} > 0.$$

- It is easy to show that all eigenvalues of such a matrix are positive. Indeed, let λ_i be one eigenvalue and \mathbf{u}_i the corresponding unit norm eigenvector ($\mathbf{u}_i^T \mathbf{u}_i = 1$). Then, by the respective definitions

$$A \mathbf{u}_i = \lambda_i \mathbf{u}_i, \quad (\text{A.2})$$

or

$$0 < \mathbf{u}_i^T A \mathbf{u}_i = \lambda_i. \quad (\text{A.3})$$

Since the determinant of a matrix is equal to the product of its eigenvalues, we conclude that the determinant of a positive definite matrix is also positive.

- Let A be an $l \times l$ symmetric matrix, $A^T = A$. Then the eigenvectors corresponding to distinct eigenvalues are orthogonal. Indeed, let $\lambda_i \neq \lambda_j$ be two such eigenvalues. From the definitions we have

$$A \mathbf{u}_i = \lambda_i \mathbf{u}_i, \quad (\text{A.4})$$

$$A \mathbf{u}_j = \lambda_j \mathbf{u}_j. \quad (\text{A.5})$$

Multiplying Eq. (A.4) on the left by \mathbf{u}_j^T and the transpose of Eq. (A.5) on the right by \mathbf{u}_i , we obtain

$$\mathbf{u}_j^T A \mathbf{u}_i - \mathbf{u}_j^T A^T \mathbf{u}_i = 0 = (\lambda_i - \lambda_j) \mathbf{u}_j^T \mathbf{u}_i. \quad (\text{A.6})$$

Thus, $\mathbf{u}_j^T \mathbf{u}_i = 0$. Furthermore, it can be shown that even if the eigenvalues are not distinct, we can still find a set of orthogonal eigenvectors. The same is true for Hermitian matrices, in case we deal with more general complex-valued matrices.

- Based on this, it is now straightforward to show that a symmetric matrix A can be diagonalized by the similarity transformation

$$U^T A U = \Lambda, \quad (\text{A.7})$$

where matrix U has as its columns the unit norm eigenvectors ($\mathbf{u}_i^T \mathbf{u}_i = 1$) of A , that is,

$$U = [\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_l], \quad (\text{A.8})$$

and Λ is the diagonal matrix with elements being the corresponding eigenvalues of A . From the orthonormality of the eigenvectors, it is obvious that $U^T U = I$ and $U U^T = I$; that is, U is an orthogonal matrix, $U^T = U^{-1}$. The proof is similar for Hermitian complex matrices as well.

A.3 WIRTINGER CALCULUS

Let a function

$$f : \mathbb{C} \mapsto \mathbb{C}, \quad (\text{A.9})$$

and let

$$f(z) = f_r(x, y) + j f_i(x, y), \quad z = x + jy, \quad x, y \in \mathbb{R}.$$

Then, the Wirtinger derivative or W -derivative of f at a point $c \in \mathbb{C}$ is defined as

$$\frac{\partial f}{\partial z}(c) := \frac{1}{2} \left(\frac{\partial f_r}{\partial x}(c) + \frac{\partial f_i}{\partial y}(c) \right) + \frac{j}{2} \left(\frac{\partial f_i}{\partial x}(c) - \frac{\partial f_r}{\partial y}(c) \right), \quad (\text{A.10})$$

and the conjugate Wirtinger derivative or CW -derivative as

$$\frac{\partial f}{\partial z^*}(c) := \frac{1}{2} \left(\frac{\partial f_r}{\partial x}(c) - \frac{\partial f_i}{\partial y}(c) \right) + \frac{j}{2} \left(\frac{\partial f_i}{\partial x}(c) + \frac{\partial f_r}{\partial y}(c) \right), \quad (\text{A.11})$$

provided that the involved derivatives exist. In this case, we say that f is differentiable in the real sense. This definition has been extended to gradients, for vector-valued functions as well as to Fréchet derivatives in complex Hilbert spaces [1]. The following properties are valid:

- If f has a Taylor series expansion with respect to z (i.e., it is holomorphic) around c , then

$$\frac{\partial f}{\partial z^*}(c) = 0.$$

- If f has a Taylor series expansion with respect to z^* around c , then

$$\frac{\partial f}{\partial z}(c) = 0.$$

- $\left(\frac{\partial f}{\partial z}(c) \right)^* = \frac{\partial f^*}{\partial z^*}(c).$
- $\left(\frac{\partial f}{\partial z^*}(c) \right)^* = \frac{\partial f^*}{\partial z}(c).$

- Linearity: If f and g are differentiable in the real sense, then

$$\frac{\partial (af + bg)}{\partial z}(c) = a \frac{\partial f}{\partial z}(c) + b \frac{\partial g}{\partial z}(c),$$

and

$$\frac{\partial (af + bg)}{\partial z^*}(c) = a \frac{\partial f}{\partial z^*}(c) + b \frac{\partial g}{\partial z^*}(c).$$

- Product rule:

$$\frac{\partial (fg)}{\partial z}(c) = \frac{\partial f}{\partial z}(c)g(c) + f(c) \frac{\partial g}{\partial z}(c),$$

and

$$\frac{\partial (fg)}{\partial z^*}(c) = \frac{\partial f}{\partial z^*}(c)g(c) + f(c) \frac{\partial g}{\partial z^*}(c).$$

- Division rule: If $g(c) \neq 0$,

$$\frac{\partial \left(\frac{f}{g}\right)}{\partial z} \Big|_c = \frac{\frac{\partial f}{\partial z}(c)g(c) - f(c)\frac{\partial g}{\partial z}(c)}{g^2(c)},$$

and

$$\frac{\partial \left(\frac{f}{g}\right)}{\partial z^*} \Big|_c = \frac{\frac{\partial f}{\partial z^*}(c)g(c) - f(c)\frac{\partial g}{\partial z^*}(c)}{g^2(c)}.$$

- Let

$$f : \mathbb{C} \mapsto \mathbb{R}.$$

If z_o is a local optimal of the real-valued f , then

$$\frac{\partial f}{\partial z}(z_o) = \frac{\partial f}{\partial z^*}(z_o) = 0.$$

Indeed, in this case $f_t = 0$ and the Wirtinger derivative becomes

$$\frac{\partial f}{\partial z}(z_o) = \frac{1}{2} \left(\frac{\partial f_r}{\partial x}(z_o) - j \frac{\partial f_r}{\partial y}(z_o) \right) = 0,$$

as at the optimal point both derivatives on the left hand side become zero. Similar is the proof for the CW-derivative.

REFERENCES

- [1] P. Bouboulis, S. Theodoridis, Extension of Wirtinger's calculus to reproducing kernel Hilbert spaces and the complex kernel LMS, *IEEE Trans. Signal Process.* 53(3) (2011) 964-978.
- [2] K.B. Petersen, M.S. Pedersen, The Matrix Cookbook, 2013, <http://www2.imm.dtu.dk/pubdb/p.php?3274>.

This page intentionally left blank

PROBABILITY THEORY AND STATISTICS

B

CHAPTER OUTLINE

B.1 Cramér-Rao Bound	1029
B.2 Characteristic Functions	1030
B.3 Moments and Cumulants	1030
B.4 Edgeworth Expansion of a pdf.....	1031
Reference	1032

B.1 CRAMÉR-RAO BOUND

Let \mathbf{x} denote a random vector and let \mathcal{X} be a set of corresponding observations, $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$. The corresponding joint pdf is parameterized in terms of the parameter vector $\boldsymbol{\theta} \in \mathbb{R}^l$. The log-likelihood is defined as,

$$L(\boldsymbol{\theta}) := \ln p(\mathcal{X}; \boldsymbol{\theta}).$$

Define the *Fisher's information matrix*

$$J := \begin{bmatrix} \mathbb{E}\left[\frac{\partial^2 L(\boldsymbol{\theta})}{\partial \theta_1^2}\right] & \mathbb{E}\left[\frac{\partial^2 L(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_2}\right] & \dots & \mathbb{E}\left[\frac{\partial^2 L(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_l}\right] \\ \vdots & \vdots & \dots & \vdots \\ \mathbb{E}\left[\frac{\partial^2 L(\boldsymbol{\theta})}{\partial \theta_l \partial \theta_1}\right] & \mathbb{E}\left[\frac{\partial^2 L(\boldsymbol{\theta})}{\partial \theta_l \partial \theta_2}\right] & \dots & \mathbb{E}\left[\frac{\partial^2 L(\boldsymbol{\theta})}{\partial \theta_l^2}\right] \end{bmatrix}. \quad (\text{B.1})$$

Let $I := J^{-1}$ and let $I(i, i)$ denote the i th diagonal element of I . If $\hat{\theta}_i$ is any *unbiased* estimator of the i th component, θ_i , of $\boldsymbol{\theta}$, then the corresponding variance of the estimator,

$$\sigma_{\hat{\theta}_i}^2 \geq I(i, i). \quad (\text{B.2})$$

This is known as the Cramér-Rao lower bound, and if an estimator achieves this bound it is said to be *efficient* and it is *unique*.

B.2 CHARACTERISTIC FUNCTIONS

Let $p(x)$ be the probability density function of a random variable x . The associated *characteristic function* is defined as the integral

$$\Phi(\Omega) = \int_{-\infty}^{+\infty} p(x) \exp(j\Omega x) dx = \mathbb{E} [\exp(j\Omega x)]. \quad (\text{B.3})$$

If $j\Omega$ is changed into s , the resulting integral becomes

$$\Phi(s) = \int_{-\infty}^{+\infty} p(x) \exp(sx) dx = \mathbb{E} [\exp(sx)], \quad (\text{B.4})$$

and it is known as the *moment generating function*.

The function

$$\Psi(\Omega) = \ln \Phi(\Omega), \quad (\text{B.5})$$

is known as the *second characteristic function* of x .

The joint characteristic function of l random variables is defined by

$$\Phi(\Omega_1, \Omega_2, \dots, \Omega_l) = \int_{-\infty}^{+\infty} \dots \int_{-\infty}^{+\infty} p(x_1, x_2, \dots, x_l) \exp \left(j \sum_{i=1}^l \Omega_i x_i \right) dx. \quad (\text{B.6})$$

The logarithm of the above is the second joint characteristic function of the l random variables.

B.3 MOMENTS AND CUMULANTS

Taking the n th-order derivative of $\Phi(s)$ in Eq. (B.4) we obtain

$$\frac{d^n \Phi(s)}{ds^n} := \Phi^{(n)}(s) = \mathbb{E} [x^n \exp(sx)], \quad (\text{B.7})$$

and hence for $s = 0$

$$\Phi^{(n)}(0) = \mathbb{E}[x^n] := m_n, \quad (\text{B.8})$$

where m_n is known as the n th-order moment of x . If the moments of all orders are finite, the Taylor series expansion of $\Phi(s)$ near the origin exists and is given by

$$\Phi(s) = \sum_{n=0}^{+\infty} \frac{m_n}{n!} s^n. \quad (\text{B.9})$$

Similarly, the Taylor expansion of the second generating function results in

$$\Psi(s) = \sum_{n=1}^{+\infty} \frac{\kappa_n}{n!} s^n, \quad (\text{B.10})$$

where

$$\kappa_n := \frac{d^n \Psi(0)}{ds^n}, \quad (\text{B.11})$$

and are known as the *cumulants* of the random variable x . It is not difficult to show that $\kappa_0 = 0$. For a zero mean random variable, it turns out that

$$\kappa_1(x) = \mathbb{E}[x] = 0, \quad (\text{B.12})$$

$$\kappa_2(x) = \mathbb{E}[x^2] = \sigma^2, \quad (\text{B.13})$$

$$\kappa_3(x) = \mathbb{E}[x^3], \quad (\text{B.14})$$

$$\kappa_4(x) = \mathbb{E}[x^4] - 3\sigma^4. \quad (\text{B.15})$$

That is, the first three cumulants are equal to the corresponding moments. The fourth-order cumulant is also known as *kurtosis*. For a Gaussian process all cumulants of order higher than two are zero. The kurtosis is commonly used as a measure of the non-Gaussianity of a random variable. For random variables described by (unimodal) pdfs with spiky shapes and heavy tails, known as leptokurtic or super-Gaussian, κ_4 is positive, whereas, for random variables associated with pdfs with a flatter shape, known as platykurtic or sub-Gaussian, κ_4 is negative. Gaussian variables have zero kurtosis. The opposite is not always true, in the sense that there exist non-Gaussian random variables with zero kurtosis; however, this can be considered rare.

Similar arguments hold for the expansion of the joint characteristic functions for multivariate pdfs. For zero mean random variables, $x_i, i = 1, 2, \dots, l$, the cumulants of order up to four are given by

$$\kappa_1(x_i) = \mathbb{E}[x_i] = 0, \quad (\text{B.16})$$

$$\kappa_2(x_i, x_j) = \mathbb{E}[x_i x_j], \quad (\text{B.17})$$

$$\kappa_3(x_i, x_j, x_k) = \mathbb{E}[x_i x_j x_k], \quad (\text{B.18})$$

$$\kappa_4(x_i, x_j, x_k, x_r) = \mathbb{E}[x_i x_j x_k x_r] - \mathbb{E}[x_i x_j] \mathbb{E}[x_k x_r] \quad (\text{B.19})$$

$$- \mathbb{E}[x_i x_k] \mathbb{E}[x_j x_r] - \mathbb{E}[x_i x_r] \mathbb{E}[x_j x_k]. \quad (\text{B.20})$$

Thus, once more, the cumulants of the first three orders are equal to the corresponding moments. If all variables coincide, we talk about *auto-cumulants*, and otherwise about *cross-cumulants*, i.e.,

$$\kappa_4(x_i, x_i, x_i, x_i) = \kappa_4(x_i),$$

that is, the fourth order auto-cumulant of x_i is identical to its kurtosis. It is not difficult to see that if the zero mean random variables are mutually independent, their cross-cumulants are zero. *This is also true for the cross-cumulants of all orders.*

B.4 EDGEWORTH EXPANSION OF A PDF

Taking into account the expansion in Eq. (B.10), the definition given in Eq. (B.5), and taking the inverse Fourier of $\Phi(\Omega)$ in Eq. (B.3) we can obtain the following expansion of $p(x)$ for a zero mean unit variance random variable x :

$$\begin{aligned} p(x) = g(x) & \left(1 + \frac{1}{3!} \kappa_3(x) H_3(x) + \frac{1}{4!} \kappa_4(x) H_4(x) + \frac{10}{6!} \kappa_3^2(x) H_6(x) \right. \\ & \left. + \frac{1}{5!} \kappa_5(x) H_5(x) + \frac{35}{7!} \kappa_3(x) \kappa_4(x) H_7(x) + \dots \right), \end{aligned} \quad (\text{B.21})$$

where $g(x)$ is the unit variance and zero mean normal pdf, and $H_k(x)$ is the Hermite polynomial of degree k . The rather strange ordering of terms is the outcome of a specific reordering in the resulting expansion, so that the successive coefficients in the series decrease uniformly. This is very important when truncation of the series is required. The Hermite polynomials are defined as

$$H_k(x) = (-1)^k \exp(x^2/2) \frac{d^k}{dx^k} \exp(-x^2/2), \quad (\text{B.22})$$

and they form a complete orthogonal basis set in the real axis, i.e.,

$$\int_{-\infty}^{+\infty} \exp(-x^2/2) H_n(x) H_m(x) dx = \begin{cases} n! \sqrt{2\pi} & \text{if } n = m \\ 0 & \text{if } n \neq m. \end{cases} \quad (\text{B.23})$$

The expansion of $p(x)$ in Eq. (B.21) is known as the *Edgeworth expansion*, and it is actually an expansion of a pdf around the normal one [1].

REFERENCE

- [1] A. Papoulis, A.U. Pillai, *Probability, Random Variables and Stochastic Processes*, fourth ed., McGraw Hill, New York, 2002.

HINTS ON CONSTRAINED OPTIMIZATION



CHAPTER OUTLINE

C.1 Equality Constraints	1033
C.2 Inequality Constraints	1035
<i>The Karush-Kuhn-Tucker (KKT) conditions</i>	1035
<i>Min-Max duality</i>	1036
<i>Saddle point condition</i>	1037
<i>Lagrangian duality</i>	1037
<i>Convex programming</i>	1038
<i>Wolfe dual representation</i>	1039
References	1039

C.1 EQUALITY CONSTRAINTS

We will first focus on linear equality constraints and then generalize to the nonlinear case. The problem is cast as

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & J(\boldsymbol{\theta}), \\ \text{s.t.} \quad & A\boldsymbol{\theta} = \mathbf{b}, \end{aligned}$$

where A is an $m \times l$ matrix and $\mathbf{b}, \boldsymbol{\theta}$ are $m \times 1$ and $l \times 1$ vectors, respectively. It is assumed that the cost function $J(\boldsymbol{\theta})$ is twice continuously differentiable and it is, in general, a nonlinear function. Furthermore, we assume that the rows of A are linearly independent, hence A has full row rank. This assumption is known as the *regularity assumption*.

Let $\boldsymbol{\theta}_*$ be a local minimizer of $J(\boldsymbol{\theta})$ over the set $\{\boldsymbol{\theta}: A\boldsymbol{\theta} = \mathbf{b}\}$. It can be shown (e.g., [5]) that, at this point, there exists a $\boldsymbol{\lambda}$ such as the gradient of $J(\boldsymbol{\theta})$ is written as

$$\frac{\partial}{\partial \boldsymbol{\theta}} (J(\boldsymbol{\theta})) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_*} = A^T \boldsymbol{\lambda}, \tag{C.1}$$

where $\boldsymbol{\lambda} := [\lambda_1, \dots, \lambda_m]^T$. Taking into account that

$$\frac{\partial}{\partial \boldsymbol{\theta}} (A\boldsymbol{\theta}) = A^T, \tag{C.2}$$

Eq. (C.1) states that, at a constrained minimum, the gradient of the cost function is a linear combination of the gradients of the constraints. We can get a better feeling of this result by mobilizing a simple example and exploiting geometry. Let us consider a single constraint,

$$\mathbf{a}^T \boldsymbol{\theta} = b.$$

Equation (C.1) then becomes

$$\frac{\partial}{\partial \boldsymbol{\theta}} (J(\boldsymbol{\theta}_*)) = \lambda \mathbf{a},$$

where the parameter λ is now a scalar. Figure C.1 shows an example of isovalue contours of $J(\boldsymbol{\theta}) = c$ in the two-dimensional space ($l = 2$). The constrained minimum coincides with the point where the straight line “meets” the isovalue contours for the first time, as one moves from small to large values of c . This is the point where the line is tangent to an isovalue contour; hence, at this point, the gradient of the cost function is in the direction of \mathbf{a} .

Let us now define the function

$$L(\boldsymbol{\theta}, \lambda) = J(\boldsymbol{\theta}) - \lambda^T (\mathbf{A}\boldsymbol{\theta} - \mathbf{b}) \quad (\text{C.3})$$

$$= J(\boldsymbol{\theta}) - \sum_{i=1}^m \lambda_i (\mathbf{a}_i^T \boldsymbol{\theta} - b_i), \quad (\text{C.4})$$

where \mathbf{a}_i^T , $i = 1, 2, \dots, m$, are the rows of \mathbf{A} . $L(\boldsymbol{\theta}, \lambda)$ is known as the *Lagrangian function* and the coefficients, λ_i , $i = 1, 2, \dots, m$, as the *Lagrange multipliers*. The optimality condition (C.1), together with the constraints, which the minimizer has to satisfy, can now be written in a compact form as

$$\nabla L(\boldsymbol{\theta}, \lambda) = \mathbf{0}, \quad (\text{C.5})$$

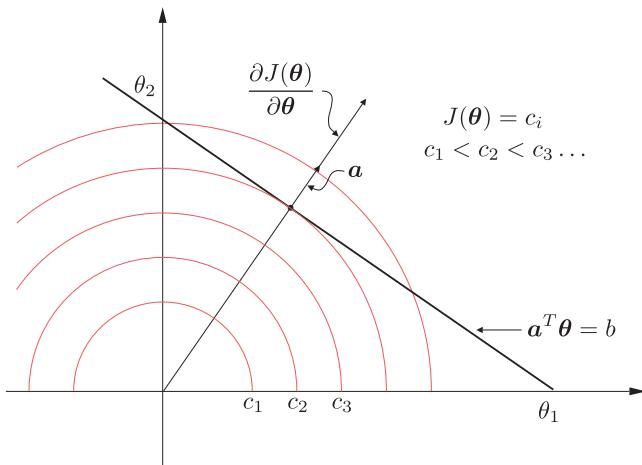


FIGURE C.1

At the minimizer, the gradient of the cost function is in the direction of the gradient of the constraint function.

where ∇ denotes the gradient operation with respect to both θ and λ . Indeed, equating with zero the derivatives of the Lagrangian with respect to θ and λ gives, respectively,

$$\begin{aligned}\frac{\partial}{\partial \theta} J(\theta) &= A^T \lambda \\ A\theta &= b.\end{aligned}$$

The above is a set of $m + l$ unknowns, i.e., $(\theta_1, \dots, \theta_l, \lambda_1, \dots, \lambda_m)$, with $m + l$ equations, whose solution provides the minimizer θ_* and the corresponding Lagrange multipliers.

Similar arguments hold for nonlinear equation constraints. Let us consider the problem

$$\begin{aligned}&\text{minimize } J(\theta) \\ &\text{subject to } f_i(\theta) = 0, \quad i = 1, 2, \dots, m.\end{aligned}$$

The minimizer is again a *stationary point* of the corresponding Lagrangian

$$L(\theta, \lambda) = J(\theta) - \sum_{i=1}^m \lambda_i f_i(\theta),$$

and results from the solution of the set of $m + l$ equations

$$\nabla L(\theta, \lambda) = \mathbf{0}.$$

The regularity condition for nonlinear constraints requires the gradients of the constraints $\frac{\partial}{\partial \theta} (f_i(\theta))$ to be linearly independent.

C.2 INEQUALITY CONSTRAINTS

The general problem can be cast as follows:

$$\begin{aligned}\min_{\theta} \quad & J(\theta) \\ \text{s.t.} \quad & f_i(\theta) \geq 0, \quad i = 1, 2, \dots, m.\end{aligned}\tag{C.6}$$

Each one of the constraints defines a region in \mathbb{R}^l . The intersection of all these regions defines the area in which the constrained minimum, θ_* , must lie. This is known as the *feasible region* and the points in it (candidate solutions) as *feasible points*. The type of the constraints control the type of the feasible region, i.e., whether it is convex or not.

Assuming that each one of the functions in the constraints is concave, then we can write each one of the constraints in Eq. (C.6) as $-f_i(\theta) \leq 0$. Now, each of the constraints becomes a convex function and the inequalities define the respective zero level sets (Chapter 8); however, these are convex sets, hence the feasible region is a convex one. For more on these issues, the interested reader may refer, for example, to [1].

Note that this is also valid for linear inequality constraints, because a linear function can be considered either convex or concave.

The Karush-Kuhn-Tucker (KKT) conditions

This is a set of *necessary* conditions, which a local minimizer θ_* of the problem given in Eq. (C.6) has to satisfy. If θ_* is a point that satisfies the regularity condition, then there exists a vector λ of Lagrange multipliers so that the following are valid:

- $$(1) \quad \frac{\partial}{\partial \theta} L(\theta, \lambda) \Big|_{\theta=\theta_*} = \mathbf{0},$$
- $$(2) \quad \lambda_i \geq 0, \quad i = 1, 2, \dots, m,$$
- $$(3) \quad \lambda_i f_i(\theta_*) = 0, \quad i = 1, 2, \dots, m.$$
- (C.7)

Actually, there is a fourth condition concerning the Hessian of the Lagrangian function, which is not of interest to us. The above set of equations is also part of the sufficiency conditions; however, for the sufficient conditions, there are a few subtle points and the interested reader is referred to more specialized textbooks, e.g., [1–5].

Conditions (3) in (C.7) are known as *complementary slackness conditions*. They state that at least one of the two factors in the products is zero. In the case where, in each one of the equations, only one of the two factors is zero, i.e., either λ_i or $f_i(\theta_*)$, we talk about *strict complementarity*.

Having now discussed all these nice properties, the major question arises: how can one compute a constrained (local) minimum? Unfortunately, this is not always an easy task. A straightforward approach would be to assume that some of the constraints are active (equality to zero holds) and some inactive, and check if the resulting Lagrange multipliers of the active constraints are nonnegative. If not, then choose another combination of constraints and repeat the procedure until one ends up with nonnegative multipliers. However, in practice, this may require a prohibitive amount of computation. Instead, a number of alternative approaches have been proposed. To this end, we will review some basics from the game theory and use these to reformulate the KKT conditions. This new setup can be useful in a number of cases in practice.

Min-Max duality

Let us consider two players, namely X and Y , playing a game. Player X will choose a strategy, say, x and simultaneously player Y will choose a strategy y . As a result, X will pay to Y the amount $\mathcal{F}(x, y)$, which can also be negative, i.e., X wins. Let us now follow their thinking, prior to their final choice of strategy, assuming that the players are good professionals.

X : If Y knew that I was going to choose x , then, because he/she is a clever player, he/she would choose y to make her/his profit maximum, i.e.,

$$\mathcal{F}^*(x) = \max_y \mathcal{F}(x, y).$$

Thus, in order to make my *worst-case payoff* to Y minimum, I have to choose x so as to minimize $\mathcal{F}^*(x)$, i.e.,

$$\min_x \mathcal{F}^*(x).$$

This problem is known as the *min-max* problem because it seeks the value

$$\min_x \max_y \mathcal{F}(x, y).$$

Y : X is a good player, so if he/she knew that I was going to play y , he/she would choose x to make her/his payoff minimum, i.e.,

$$\mathcal{F}_*(y) = \min_x \mathcal{F}(x, y).$$

Thus, in order to make my *worst-case profit* maximum I must choose y that maximizes $\mathcal{F}_*(y)$, i.e.,

$$\max_y \mathcal{F}_*(y).$$

This is known as the *max-min* problem, as it seeks the value

$$\max_y \min_x \mathcal{F}(x, y).$$

The two problems are said to be *dual to each other*. The first is known to be the *primal*, whose objective is to minimize $\mathcal{F}^*(x)$ and the second is the *dual* problem with the objective to maximize $\mathcal{F}_*(y)$.

For any x and y , the following is valid:

$$\mathcal{F}_*(y) := \min_x \mathcal{F}(x, y) \leq \mathcal{F}(x, y) \leq \max_y \mathcal{F}(x, y) := \mathcal{F}^*(x), \quad (\text{C.8})$$

which easily leads to

$$\max_y \min_x \mathcal{F}(x, y) \leq \min_x \max_y \mathcal{F}(x, y). \quad (\text{C.9})$$

Saddle point condition

Let $\mathcal{F}(x, y)$ be a function of two vector variables with $x \in X \subseteq \mathbb{R}^l$ and $y \in Y \subseteq \mathbb{R}^l$. If a pair of points (x_*, y_*) , with $x_* \in X, y_* \in Y$ satisfies the condition

$$\mathcal{F}(x_*, y) \leq \mathcal{F}(x_*, y_*) \leq \mathcal{F}(x, y_*), \quad (\text{C.10})$$

for every $x \in X$ and $y \in Y$, we say that it satisfies the *saddle point condition*. It is not difficult to show (e.g., [5]) that a pair (x_*, y_*) satisfies the saddle point conditions if and only if

$$\max_y \min_x \mathcal{F}(x, y) = \min_x \max_y \mathcal{F}(x, y) = \mathcal{F}(x_*, y_*). \quad (\text{C.11})$$

Lagrangian duality

We will now use all the above in order to formulate our original cost function minimization problem as a min-max task of the corresponding Lagrangian function. Under certain conditions, this formulation can lead to computational savings when computing the constrained minimum. The optimization task of interest is

$$\begin{aligned} & \text{minimize } J(\theta), \\ & \text{subject to } f_i(\theta) \geq 0, \quad i = 1, 2, \dots, m. \end{aligned}$$

The Lagrangian function is

$$L(\theta, \lambda) = J(\theta) - \sum_{i=1}^m \lambda_i f_i(\theta). \quad (\text{C.12})$$

Let

$$L^*(\theta) = \max_{\lambda} L(\theta, \lambda). \quad (\text{C.13})$$

However, because $\lambda \geq \mathbf{0}$ and $f_i(\theta) \geq 0$, the maximum value of the Lagrangian occurs if the summation in Eq. (C.12) is zero (either $\lambda_i = 0$ or $f_i(\theta) = 0$ or both) and

$$L^*(\theta) = J(\theta). \quad (\text{C.14})$$

Therefore, our original problem is equivalent with

$$\min_{\theta} J(\theta) = \min_{\theta} \max_{\lambda \geq \mathbf{0}} L(\theta, \lambda). \quad (\text{C.15})$$

As we already know, the dual problem of the above is

$$\max_{\lambda \geq \mathbf{0}} \min_{\theta} L(\theta, \lambda). \quad (\text{C.16})$$

Convex programming

A large class of practical problems obeys the following two conditions:

$$(1) \quad J(\theta) \text{ is convex,} \quad (\text{C.17})$$

$$(2) \quad f_i(\theta), i = 1, 2, \dots, m, \text{ are concave.} \quad (\text{C.18})$$

This class of problems turns out to have a very useful and mathematically tractable property.

Theorem C.1. *Let θ_* be a minimizer of such a problem, which is also assumed to satisfy the regularity condition. Let λ_* be the corresponding vector of Lagrange multipliers. Then (θ_*, λ_*) is a saddle point of the Lagrangian function, and, as we know, this is equivalent to*

$$L(\theta_*, \lambda_*) = \max_{\lambda \geq \mathbf{0}} \min_{\theta} L(\theta, \lambda) = \min_{\theta} \max_{\lambda \geq \mathbf{0}} L(\theta, \lambda). \quad (\text{C.19})$$

Proof. Because $f_i(\theta)$ are concave, $-f_i(\theta)$ are convex, so the Lagrangian function

$$L(\theta, \lambda) = J(\theta) - \sum_{i=1}^m \lambda_i f_i(\theta),$$

for $\lambda_i \geq 0$, is also convex. Note, now, that for concave function constraints of the form $f_i(\theta) \geq 0$, the feasible region is convex (see comments made before). The function $J(\theta)$ is also convex. Hence, every local minimum is also a global one; thus for any θ

$$L(\theta_*, \lambda_*) \leq L(\theta, \lambda_*). \quad (\text{C.20})$$

Furthermore, the complementary slackness conditions suggest that

$$L(\theta_*, \lambda_*) = J(\theta_*), \quad (\text{C.21})$$

and for any $\lambda \geq \mathbf{0}$

$$L(\theta_*, \lambda) := J(\theta_*) - \sum_{i=1}^m \lambda_i f_i(\theta_*) \leq J(\theta_*) = L(\theta_*, \lambda_*). \quad (\text{C.22})$$

Combining Eqs. (C.20) and (C.22) we obtain

$$L(\theta_*, \lambda) \leq L(\theta_*, \lambda_*) \leq L(\theta, \lambda_*). \quad (\text{C.23})$$

In other words, *the solution (θ_*, λ_*) is a saddle point.* □

This is a very important theorem and it states that the constrained minimum of a convex programming problem can also be obtained as a maximization task applied on the Lagrangian. This leads us to the following very useful formulation of the optimization task.

Wolfe dual representation

A convex programming problem is equivalent to

$$\max_{\lambda \geq \mathbf{0}} L(\theta, \lambda), \quad (\text{C.24})$$

$$\text{s.t. } \frac{\partial}{\partial \theta} L(\theta, \lambda) = \mathbf{0}. \quad (\text{C.25})$$

The last equation guarantees that θ is a minimum of the Lagrangian.

Example C.1. Consider the quadratic problem

$$\text{minimize } \frac{1}{2} \theta^T \theta,$$

$$\text{subject to } A\theta \geq b,$$

This is a convex programming problem; hence, the Wolfe dual representation is valid:

$$\text{maximize } \frac{1}{2} \theta^T \theta - \lambda^T (A\theta - b)$$

$$\text{subject to } \theta - A^T \lambda = \mathbf{0}.$$

For this example, the equality constraint has an analytic solution (this is not, however, always possible). Solving with respect to θ , we can eliminate it from the maximizing function and the resulting dual problem involves only the Lagrange multipliers,

$$\max_{\lambda} -\frac{1}{2} \lambda^T A A^T \lambda + \lambda^T b,$$

$$\text{s.t. } \lambda \geq \mathbf{0}.$$

This is also a quadratic problem but the set of constraints is now simpler.

REFERENCES

- [1] M.S. Bazaraa, C.M. Shetty, Nonlinear Programming: Theory and Algorithms, John Wiley, New York, 1979.
- [2] D.P. Bertsekas, M.A. Belmont, Nonlinear Programming, Athena Scientific, Belmont, MA, 1995.
- [3] R. Fletcher, Practical Methods of Optimization, second ed., John Wiley, New York, 1987.
- [4] D.G. Luenberger, Linear and Nonlinear Programming, Addison Wesley, Reading, MA, 1984.
- [5] S.G. Nash, A. Sofer, Linear and Nonlinear Programming, McGraw-Hill, New York, 1996.

This page intentionally left blank

Index

Note: Page numbers followed by *f* indicate figures and *t* indicate tables.

A

- Active constraints, 544
- Active set, 692
- Adaptive algorithm, 162
- Adaptive Boosting (AdaBoost) algorithm, 307–311
- Adaptive coordinate descent scheme, 259–261
- Adaptive CoSaMP (AdCoSaMP) algorithm, 479–480, 484^f
- Adaptive decision feedback equalization, 202–204
- Adaptive gradient (ADAGRAD) algorithm, 368
- Adaptive line element (ADALINE), 881
- Adaptive projected subgradient method (APSM), 349–350
 - algorithm, 350
 - asymptotic consensus, 358
 - combine-then-adapt diffusion, 357–358
 - constrained learning, 356
 - convergence of, 351–356
 - distributed algorithms, 357–358
 - hyperslabs, 352
 - parameters, 352–356
 - projection operation, 351
 - SpAPSM, 480–484, 481^f, 484^f
- Adaptive signal processing, 5
- Adapt-then-combine DiLMS, 215–216, 216^f
- Additive models approach, 568–570
- Ad hoc networks, 210
- ADMM algorithm. *See* Alternating direction method of multipliers (ADMM) algorithm
- Affine projection algorithm (APA), 188–194, 201
 - convergence, 353
 - curves for, 355^f
 - geometric interpretation of, 189–191
 - normalized LMS, 193–194
 - orthogonal projections, 191–194
 - set-membership, 354
 - widely-linear, 195–196
- Affine set, 415–416
- Airborne Visible/Infrared Imaging Spectrometer (AVIRIS), 696–699
- ALMA algorithm, 560
- Alternating direction method of multipliers (ADMM) algorithm, 220–221, 387–388
- Alternating optimization, 608–609
- Amino-acids, proteinogenic, 314–315

B

- Amplitude beam-pattern, 149^f
- Analog signal
 - Fourier transform, 435^f
 - sampling process, 432^f, 434–435
- Analog-to-information sampling, 434–435
- Analysis of variance (ANOVA), 570
- APA. *See* Affine projection algorithm (APA)
- Approximate inference
 - block methods, 809–813
 - loopy belief propagation, 813–816
 - variational methods, 804–809
- Approximation error, 94, 374–376, 511
- APSM. *See* Adaptive projected subgradient method (APSM)
- Arithmetic averaging rule, 305
- ARMA model. *See* Autoregressive-moving average (ARMA) model
- AR models. *See* Autoregressive (AR) models
- Assumed density filtering (ADF), 682
- Asymptotic distribution, of LS estimator, 238–239
- Augmented Lagrangian method, 387–388
- Authorship identification, 570–573
- Autocorrelation matrix, 114–115
- Autocorrelation sequence, 33–35
- Auto-cumulants, 1021
- Autoencoders, 919–920, 925–927
- Automatic relevance determination (ARD), 655–656
- Autoregressive hidden Markov model, 829
- Autoregressive (AR) models, 38–40
- Autoregressive-moving average (ARMA) model, 40
- Autoregressive process estimation, 153
- Auxiliary particle filtering, 862–868
- Auxiliary variable Markov chain Monte Carlo methods, 735
- Average mutual information, 44–45, 47
- Average risk, Bayesian classification, 278–280
- Averaging method, 187
- Averaging rule, 217

- Backpropagation algorithm (*Continued*)
 preprocess input variables, 894
 target values, 894
- Backtracking, 783–785
- Backward errors, 138–140
- Backward MSE optimal predictors, 134–138
- Bagging, 303–304
- Bag-of-words approach, 571
- Bandpass filter, 37–38
- Base learner, 307
- Base transition matrices, 724
- Basis pursuit, 407, 418
- Basis pursuitde-noising (BPDN), 408
- Basis vector, 395
- Batch learning, 376–379
- Batch processing methods, 162
- Baum–Welch algorithm, 827–828
- Bayesian approach, 5
 to regression, 589–593
 to sparsity-aware learning, 655–661
- Bayesian classification, 276–280
 average risk, 278–280
 designing classifiers, 282
 equiprobable Gaussian classes, 284f
 Gaussian distributed classes, 283f
 implicitly forms hypersurfaces, 281f
M-class problem, 278–279, 284, 293–294
 misclassification error, 277–280
 reject option, 279
- Bayesian decision theory, 276
- Bayesian inference, 84–89, 87f
- Bayesian information criterion (BIC), 599
- Bayesian learning, 11–12
 neural networks, 902–903
 regularization, 75
 variational approximation, 640–645
- Bayesian networks (BNs)
 causality, 753–755
 cause–effect relationships, 753–755
 completeness, 761–762
d-separation, 755–758, 758f
 faithfulness, 761–762
 graphs, 749–753
 I-maps, 761–762
 independent variables, 751f
 joint pdf, 836–837
 Kalman filtering, 852–854
 latent Markov model, 818f
 linear Gaussian models, 759–760
- multiple-cause networks, 760, 761f
- naive Bayes classifier, 835f
- set of findings and diseases, 806f
- sigmoidal, 758–759, 759f, 760
- soundness, 761–762
- triangulated graph, 800–801, 800f
- Bayesian regression, 690–692
 computational considerations, 692
 hyperparameters, 691–692
- Bayes’s theorem, 13, 276–277, 586
- Beamforming, 145–148
- Belief propagation, 782
- Bernoulli distribution, 18
- Best linear unbiased estimator (BLUE), 144–145, 237–238
- Beta distribution, 25–26
- Bethe entropy approximation, 813–814
- Bethe free energy cost, 813–814
- Between-classes scatter matrix, 296
- Biased estimation, 64–67
- Biasor, 57–58
- Bias-variance dilemma/tradeoff, 77–81
- BIC. *See* Bayesian information criterion (BIC)
- Big data problems, 162
- Big data tasks, 376
- Binary classifier, 307–308
- Binomial deviance function, 311–313
- Binomial distribution, 18–19
- Bipartite graph, 769
- Blind source separation (BSS), 964–965
- Blocking Gibbs sampling, 734
- Block methods, 809–813
- Block processing techniques, 197
- Block sparsity, 468
- BLUE. *See* Best linear unbiased estimator (BLUE)
- BNs. *See* Bayesian networks (BNs)
- Boltzmann machines, 767
 graph nodes representing, 812f
 mean field approximation, 810–813
 MRF, 809f
 variational approximation, 807–809
- Boolean approach, bag-of-words, 571
- Boosting approach, 307–313
- Boosting trees, 313–314
- Bootstrap Aggregating, 303–304
- Bootstrap techniques, 93
- Box–Müller method, 713–714
- Bregman divergence, 389
- Burn-in phase, Metropolis method, 730

C

- Calculus of variations, 641
 Canonical correlation analysis (CCA)
 content-based image retrieval, 953
 correlation coefficient, 951–952
 eigenvalue-eigenvector problem, 952, 953–954
 goals, 951
 optimization task, 951–952
 PLS method, 954–955
 Capon beamforming, 148
 CART. *See* Classification and regression trees (CART)
 Cauchy-Schwartz inequality, 34, 396
 Cauchy sequence, 397
 Causality, 753–755
 Cause-effect relationships, 753–755
 Centralized networks, 209, 210f
 Central limit theorem, 24–25
 Chain graph, 773–776
 Change-point detection, 737–738
 Channel equalization, 126–132
 Channel identification, 144–145
 Characteristic functions, 1020
 Chinese restaurant process (CRP), 683–684
 Chi-squared distribution, 27
 Cholesky factorization, 140, 255
 Circular condition, 115–116
 Class assignment rule, 303
 Classification, 2–3, 60–64
 Bayesian classification, 276–280
 decision (hyper)surfaces, 280–290
 discrete nature, 275–276
 Gaussian random process, 692–693
 generative *vs.* discriminative learning, 63–64
 logistic regression model for, 662–666
 M-class problem, 278–279, 284, 293–294
 POCS, 347–349
 protein folding prediction, 316, 318
 trees, 300–304, 301f
 two-class task, 277, 286, 290f, 296–297, 312
 unstable, 303
 Classification and regression trees (CART), 300, 317f, 318
 Classifiers, 2, 3f
 combining, 304–307
 experimental comparisons, 304–305
 goal to design, 60–61
 schemes for combining, 305–307
 Class imbalance problem, 552
 Class label variable, 61

- Clifford algebras, 552
 Clique, 764f, 769f
 message passing, 801–804
 potentials, 763
 Closed convex set, 345–346
 associated projection operator, 338–339
 finite number, 341
 Hilbert space, 334, 337, 339–340, 344
 infinite, 349–356
 nonempty intersection, 342
 sliding window, 349–350
 Clustering, 3, 64, 617–620
 Cocktail party problem, 963–966
 Codon, 314–315
 Collapsed Gibbs sampling, 735
 Combine-then-adapt diffusion APSM, 357–358
 Combine-then-adapt diffusion LMS, 217
 Common sense reasoning, 753–754
 Communications channel, 43–44
 Compatibility functions/factors, 762
 Complementary slackness conditions, 1026
 Complete dictionaries, 414–415
 Complex linear space, 394
 Complex networks, 211
 Complex random variables, 16–17
 Complex-valued case
 adaptive decision feedback equalization, 202–204
 least mean fourth algorithm, 196–197
 mean-square-error loss function, 175–176, 194–195
 sign-error LMS, 196
 transform-domain LMS, 197–201
 widely-linear APA, 195–196
 widely-linear LMS, 195
 Complex-valued data, widely linear RLS, 254–255
 Complex-valued variables
 extension to, 111–118
 widely linear, 113–116
 Wirtinger calculus, 116–118
 Composite mirror descent, 389
 Compressed sensing (CS), 404
 analog-to-information conversion, 434–436
 definition, 430–431
 description, 431–436
 dimensionality reduction, 433–434
 sparse signal representation, 487–488
 stable embeddings, 433–434
 sub-Nyquist sampling, 434–436
 Compressed sensing matching pursuit (CSMP) algorithms, 455–456, 460, 479–480

- Compressive sampling matching pursuit (CoSaMP), [455–456](#), 480
- Computational considerations, Bayesian regression, [692](#)
- Computation, of lower bound, [650–651](#)
- Concave, [667](#), [668–669](#)
- Concentration parameter, [684](#)
- Conditional entropy, [45](#)
- Conditional independencies, [749](#), [752–753](#)
- Conditional information, [43–44](#), [47](#)
- Conditional log-likelihood, [835–836](#)
- Conditional pdf, [632–633](#), [634–637](#)
- Conditional probabilities, [12–13](#)
- Conditional random fields (CRFs), [767–768](#)
- Conditional Random Markov Field, [768](#)
- Conditional restricted Boltzmann machine (CRBM), [920–922](#)
- Conjugate function, [666–667](#)
- Conjugate prior, [89](#)
- Dirichlet distribution, [604](#)
 - gamma distribution, [601](#)
 - Gaussian-gamma form, [603](#)
- Conjugate Wirtinger's derivative, [117](#)
- Consensus-based algorithms, [221](#)
- Consensus-based distributed schemes, [220–222](#)
- Consensus matrix, [223–224](#)
- Consensus strategy, [221–222](#)
- Consistent estimator, [31](#)
- Constrained-based path, [837](#)
- Constrained learning, [356](#)
- Constrained linear estimation, [145–148](#)
- Continuous random variables, [14](#)
- average mutual information, [47](#)
 - conditional information, [47](#)
 - entropy for, [46–47](#)
 - generalization, [45](#)
- Kullback-Leibler divergence, [47–48](#)
- relative entropy, [47–48](#)
- Continuous-time signal, [29](#)
- Continuous variables
- beta distribution, [25–26](#)
 - central limit theorem, [24–25](#)
 - Dirichlet distribution, [27–49](#)
 - exponential distribution, [25](#)
 - gamma distribution, [26–27](#)
 - Gaussian distribution, [20–24](#)
 - uniform distribution, [20](#)
- Contrastive divergence (CD), [911](#)
- Convergence
- affine projection algorithm, [353](#)
 - APSM, [351–356](#)
- connection, [756](#)
- distributed learning, [181–186](#), [218–219](#)
- distributions, [49–51](#)
- error vector, [181–186](#)
- issues, Metropolis method, [731–732](#)
- in mean, [182–183](#), [218](#)
- NORMA, [559–560](#)
- performance, [218–219](#)
- stochastic (*see* Stochastic convergence)
- Convex, [330–333](#)
- duality, [666–671](#)
 - online learning, [367–370](#)
 - optimization techniques, [458](#), [460](#), [478](#), [487](#)
 - programming, [1028–1029](#)
 - separating hyperplane, [370](#)
 - strictly, [331](#)
 - theory, [328](#)
- Convex set, [329](#)
- closed (*see* Closed convex set)
 - Hilbert space, [334](#), [337](#)
 - strongly attracting nonexpansive mapping, [356](#)
 - theory, [328](#)
- Convolution matrices, [132](#)
- Coordinate descent (CD), [258–261](#), [459](#)
- Correlated component analysis, [953](#)
- Correlation, [15](#)
- Correlation matrix, [15–16](#)
- CoSaMP. *See* Compressive sampling matching pursuit (CoSaMP)
- Cosparsity, [488–490](#)
- Cost function
- backpropagation algorithm, [896–897](#)
 - isovalue curves, [164^f](#)
 - surface, [107–108](#), [109^f](#)
 - two-dimensional parameter space, [164^f](#)
- Countably infinite, [683](#)
- Coupon collector's problem, [992](#)
- Covariance, [15](#)
- functions, [688–689](#)
 - Kalman algorithm, [152](#)
 - matrix, [15–16](#), [175^f](#)
- Cover's theorem, [514–517](#)
- Cramér-Rao bound, [67–72](#), [1019](#)
- CRFs. *See* Conditional random fields (CRFs)
- Cross-correlation vector, [128](#), [129](#), [171–174](#)
- Cross-cumulants, [1021](#)
- Cross-entropy cost function, [896](#)
- Cross-entropy error, [292](#)
- Cross-spectral density, [120–121](#)

- Cross-validation, 92–93
CS. See Compressed sensing (CS)
 CSMP algorithms. *See* Compressed sensing matching pursuit (CSMP) algorithms
C-SVM, 547
 Cumulant generating function, 815
 Cumulants, 1020–1021
 Cumulative distribution function (cdf), 14, 19f, 713f
 Cumulative loss, 186–188, 371
 Cuprite data set, 696–697
 Curse of dimensionality, 89–91, 90f
 Curve fitting problem, 54–55, 55f
 Cyclic coordinate descent (CCD), 258–261
 Cyclic path, 210
- D**
- DAG. *See* Directed acyclic graph (DAG)
 Dantzig selector, 472
 Data sets, 91
 De-blurring, 4–5, 4f
 Decentralized networks, 210–211
 Decision feedback equalizer (DFE), 202–203, 203f, 204f
 Decision surface, 60–61, 280–281, 282
 Gaussian distribution, 282–287
 naive Bayes classifier, 287–288
 nearest neighbor rule, 288–290
 Decision trees, 304
 CART, 317f
 protein folding prediction classification, 318
 Decomposition, analysis of variance, 570
 Deconvolution, 121–124, 126–132
 Deep belief network (DBN), 916–918, 928
 Deep learning, 877
 block diagram, 906f
 character recognition, 923–925
 CRBM, 920–922
 Gaussian visible units, 918–919
 issues, 903
 stacked autoencoder, 919–920
 training, 905–908
 Deflation procedure, 954–955
 Degeneracy phenomenon, 858–860
 Degree of node k , 211–212
 Deming regression, 262
 De-noising, 438–439, 439f
 Denoising autoencoder, 920
 Density function, 14
 Dependent random variable, 57–58
- DFE. *See* Decision feedback equalizer (DFE)
 Dictionary learning (DL), 414–415
 codebook update, 967–968
 image de-noising, 970, 971f
 optimization problem, 966
 sparse coding, 967
 Difference equation, 38–39
 Diffusion gradient descent, 215
 Diffusion LMS (DiLMS), 211–218
 adapt-then-combine, 215–216, 216f
 combine-then-adapt, 217
 Dimensionality reduction, 243–244, 433–434
 Directed acyclic graph (DAG), 749
 Bayesian network, 749, 751
 d-separation, 758f
 independencies, 762f
 moralization on, 772f
 Directed graphs, 749, 772
 Dirichlet distribution, 27–49, 603–604
 Dirichlet process (DP), 684–686
 Discrete cosine transform (DCT), 412–413, 413f
 Discrete distributions
 cumulative distribution function, 713f
 generating samples from, 711–712
 resampling, 847–849
 Discrete random variables, 12–13
 codewords, 42–43
 entropy/average mutual information, 44–45
 information, 42–43
 mutual/conditional information, 43–44
 Discrete-time random process, 29
 Discrete-time stochastic process, 29f
 Discrete variables
 Bernoulli distribution, 18
 binomial distribution, 18–19
 multinomial distribution, 19–20
 Discrete wavelet transform (DWT), 412–413, 414–415
 Discriminant functions, 282
 Discriminative learning
 generative vs., 63–64
 hidden Markov model, 828–829
 Disjoint subsets, 302–303
 Distributed learning
 consensus-based schemes, 220–222
 convergence, 181–186, 218–219
 cooperation strategies, 209–211
 diffusion LMS, 211–218
 LMS, 208–222
 steady-state performance, 218–219

Distributed sparsity-promoting algorithms, 483
 α -Divergence, 682
 Diverging connection, 756
 Division algebra, 552
 DNA sequences, 314–318
 Doubly stochastic matrix, 213–214
 D -separation, BNs, 755–758, 758f
 Dual frames, 498
 Dynamic Bayesian networks, 832–833
 Dynamic graphical models, 816–818

E

Echo canceller, 125f
 Echolocation signals, time-frequency analysis, 493–497
 Eckart-Young-Mirsky theorem, 242
 Edgeworth expansion, PDF, 1021–1022
 Eigenvalues
 covariance matrix, 175f
 unequal, 172f, 174f
 EKF. *See* Extended Kalman filter (EKF)
 Elastic net regularization, 472
 EM algorithm. *See* Expectation-maximization (EM) algorithm
 Empirical bayes method, 600
 Empirical loss functions, 93–94
 Energy conservation method, 187
 Entropy, 44–45, 302–303
 binary random variable, 46f
 continuous random variable, 46–47
 differential entropy, 47
 relative, 47–48
 Epigraph, 332, 405–407, 406f
 Equality constraints, 1023–1029
 Equalizer, 127, 127f
 Ergodicity, 31
 Ergodic Markov chain Monte Carlo methods, 723–728
 Erlang distribution, 27
 Error bounds, NORMA, 559–560
 Error-correcting codes, 770–772
 Error covariance matrix, 141, 150–152
 Errors-in-variables regression models, 262
 Error vector
 convergence, 181–186
 covariance matrix, 183–184
 Estimation
 error, 94, 374–376
 interpretation power, 407
 nonparametric modeling and, 95–97
 Euclidean distance, 283, 285

Euclidean norm, 395, 404–405
 descent directions, 166f
 graphs, 250f
 Euclidean space, 109
 Evidence function, 593–595, 596–600
 Exact inference methods
 chain graph, 773–776
 trees, 777–778
 Excess mean-square error, 184
 Expectation-maximization (EM) algorithm, 598
 convergence criterion, 607
 description, 606–608
 E-step, 607, 623
 linear regression, 610–612
 lower bound maximization view, 608–610
 missing data, 608
 Monte Carlo methods, 720–721
 M-step, 607, 623
 Newton-type searching techniques, 607
 online versions, 609
 Expectation propagation, 679–683
 Expectation step, hidden Markov model, 825–827
 Expected loss, 93–94, 177–178
 Expected risk, 177–178
 Expected value, 15
 Explaining away, 756
 Exponential distribution, 25, 711
 Exponential family
 advantage, 600
 of probability distributions, 600–606, 644–645
 Exponentially weighted isometry property (ERIP), 480
 Exponentially weighted least-squares cost function, 245–246
 time-iterative computations, 246–247
 time updating, 247–248
 Extended Kalman filter (EKF), 152, 854
 Extreme Learning Machines (ELMs), 900

F

Factor analysis, 972, 977–980
 Factor graphs, 768–772
 Factorial hidden Markov model (FHMM), 829–832
 Factorization
 pdf, 643
 theorem, 71
 Far-end speech signal, 125
 Fast iterative shrinkage-thresholding algorithm (FISTA), 459, 461
 Fast Newton transversal filter (FNTF) algorithm, 257

- Fast proximal gradient splitting algorithm, 386
FDR. *See* Fisher's discriminant ratio (FDR)
Feasibility set, 349
Feasible points, 1025
Feasible region, 1025
Feature generation
 phases, 295–296, 295f
 stage, 2
Feature map, 517
Feature selection
 phases, 295–296, 295f
 stage, 2
Feature space, 2, 517
Feature variable, 60–61
Feature vector, 2, 60–61
Feed-forward neural networks, 882–886
 deep learning, 914–915
 hidden layer, 884
 multilayer, 882–886
 output neuron, 884
 universal approximation property, 899–902
Fill-in edge, 798
Finite rate of innovation sampling, 436
First order convexity condition, 331
Fisher-Neyman factorization theorem, 71
Fisher's discriminant ratio (FDR), 296–297
Fisher's information matrix, 1019
Fisher's linear discriminant, 294–300
FISTA. *See* Fast iterative shrinkage-thresholding algorithm (FISTA)
Fixed interval, 866
Fixed lag smoothing, 866
Fixed point set, 339
Focal underdetermined system solver (FOCUSS) algorithm, 472
Forward-backward algorithm, 827
Forward-backward splitting algorithms, 385–386
Forward MSE optimal predictors, 134–138
Fourier transforms, 33
 analog signal, 435f
 software packages to, 122–123
Frames theory, 497–502
Free energy, 608
Frequency approach, bag-of-words, 571
Frequentist techniques, 586
Frobenius norm, 265
Functional brain networks (FBN), 998
Functional magnetic resonance imaging (fMRI)
 BOLD contrast, 998–999
 functional brain networks, 998
 goals, 999
 ICA, 1000
 scanning procedure, 1000, 1000f
Function transformation, 711–715

G

- Gabor frames,** 490–492, 493, 496f
Gabor transform, 490–492
Gabor type signal expansion, 414–415
Gamma distribution, 26–27
Gating functions, 621
Gaussian distribution, 183–184, 276
 continuous variables, 20–24
 decision surfaces, 282–287
 hypersurfaces, 282–287
 isovalue contours for, 23f
 multivariate, 21–22, 24
 pdf, 22f, 24
 sub-gaussian distribution, 196–197
Gaussian-gamma distribution, 603, 603f
Gaussian-gamma pair, 601–602
Gaussian Gaussian-gamma pair, 602–603
Gaussian kernel, 520, 521f, 665f, 688
Gaussian mixture modeling, 613–620, 651–654
Gaussian noise case, nonwhite, 84
Gaussian pdf, 655–656
 computational advantages, 759–760
 conditional, 632–633, 634–637
 joint, 632–634
 marginal pdf, 633–634
 with quadratic form exponent, 631
Gaussian processes (GP), 687–693
Gauss-Markov theorem, 143–145, 237–238
Generalization, 91
Generalization error, 80–81
Generalized forward-backward algorithm, 782
Generalized linear models, 510–511
Generalized maximum likelihood, 600
Generalized Rayleigh ratio, 296–297
Generalized thresholding (GT), 483
Generative learning, 63–64
Generic particle filtering, 860–861
Genes, 315–316
Geometric averaging rule, 305
Gibbs distribution, 763
 cliques, 763, 769f
 I-map, 764

- Gibbs sampling, 733–735
 blocking, 734
 change-point detection, 738
 collapsed, 735
 slice-sampling algorithm, 735
- Gini index, 303
- Givens rotations, 256
- Global decomposition, likelihood function, 836–837
- Gradient averaging, 378
- Gradient descent algorithm, 163, 165, 166f, 173f
- Gradient descent scheme, 887–894
 adaptive momentum, 893
 algorithm, 891–892
 backpropagation algorithm, 891–892
 gradient computation, 889
 iteration-dependent step-size, 893
 logistic sigmoid neuron, 887
 momentum term, 893
 paramount importance, 895
 pattern-by-pattern/online mode, 892
 quickprop algorithm, 895
- Gradient vector, 163–165, 165f
- Gram-Schmidt orthogonalization, 138, 256
- Graph embedding, 989
- Graphical models
 dynamic, 816–818
 for error-correcting codes, 770–772
 learning structure, 837
 need for, 746–748
 parameter estimation, 833–837
 probabilistic, 751
 undirected, 762–768
- Graphs
 bipartite, 769
 definitions, 749–753
 direction/undirected, 749
 factor, 768–772
 triangulated, 796–804
 undirected (*see* Undirected graph)
- Graph theory, 746
- Greedy algorithms, 451–456
 CSMP, 455–456, 460
 LARS, 454
 OMP, 451, 453
- H**
- Halfspace, 347–348
- Hamiltonian Monte Carlo methods, 736
- Hammerstein model, 511–514
- Hard thresholding
 function, 456–457, 459, 460
 operation, 409–411, 410f
- Head-to-head connection, 756
- Head-to-tail connection, 755
- Heat bath algorithm, 733
- Heavy-tailed distribution, 671
- Hermitian operation, 16–17
- Hessian matrix, 292, 293–294, 377–378
- Hidden Markov model (HMM), 816, 817–818
 autoregressive, 828
 discriminative learning, 828–829
 expectation step, 825–827
 FHMM, 829–832
 inference, 821–825
 left-to-right type, 819, 820f
 maximization step, 827
 parameters, 821, 825–828
 sum product algorithm, 821
 time-varying dynamic Bayesian networks, 832–833
 transition probability, 818, 819
 variable duration, 829
 Viterbi reestimation, 827–828
- Hidden variables, 606
- Hierarchical Bayesian modeling, 647, 695–696
- Hierarchical mixture of expert (HME), 625, 626f
- Hierarchical priors, 599
- High-definition television (HDTV) system, 412–413
- Hilbert space, 329, 397
 closed convex set, 334, 337, 339–340, 344
 convex set, 334
- Hinge loss function, 348–349, 349f, 538–539, 558–559
- Histogram technique, 95–96
- HME. *See* Hierarchical mixture of expert (HME)
- HMM. *See* Hidden Markov model (HMM)
- Homotopy algorithm, 454
- Householder reflections, 256
- Huber loss function, 530–531, 531f
- Hyperparameters, 599, 600
 Gaussian process, 691–692
 support vector machine, 550–551
- Hyperplane, 60, 61–62
- Hyperprior, 647
- Hyper rectangles, 300–301
- Hyperslab, 345–346
- Hyperspectral image unmixing (HSI), 693–699
 experimental results, 696–699
 hierarchical Bayesian modeling, 695–696

Hyperspectral remote sensing, 693–694
 Hypersurfaces, 280–281, 282
 Gaussian distribution, 282–287
 naive Bayes classifier, 287–288
 nearest neighbor rule, 288–290
 Hypothesis class, 371
 Hypothesis space, 528–529

I

IIR. *See* Infinite impulse response (IIR)
 Ill conditioning, 74–76
 Image deblurring, 121–124
 I-maps
 BNs, 761–762
 Markov Random Fields, 763–765
 Importance sampling (IS), 718–720
 Impulse response function, 411–412, 412f
 Incremental networks, 210
 Incremental topology, 211f
 Independence assumption, 182
 Independent component analysis (ICA), 944
 ambiguities, 958
 cocktail party problem, 963–966
 Edgeworth expansion, 959–960
 fourth-order cumulants, 957–958
 Gaussian distributions, 956
 gradient ascent scheme, 960–961
 Infomax principle, 962
 Kullback-Leibler divergence, 959–960
 maximum likelihood, 962
 mixture variables, 955
 mutual information, 959–960
 natural gradient, 961–962
 negentropy, 963
 non-Gaussian distributions, 958–959
 Riemannian metric tensor, 961–962
 tensorial methods, 958
 unmixing/separating matrix, 955–956
 Inequality constraints, 1025–1029
 Inference, 684, 821–825
 Infinite impulse response (IIR), 120–121
 Information filtering scheme, 152
 Information projection (I-projection), 679
 Information theory, 41
 continuous random variables, 45–48
 discrete random variables, 42–45
 Inner product space, 395
 Innovations process, 152
 Input space, 2

Input vector, 57–58
 Intercausal reasoning, 756
 Intercept, 57–58
 Interference cancellation, 124–125
 Interior point methods, 358–359
 Interpretation power of estimator, 407
 Intersymbol interference (ISI), 127, 412–413
 Intrinsic dimensionality, 90–91, 938, 938f, 939
 Invariant distribution, 722
 Inverse Fourier transform, 35
 Inverse problems, 74–76
 Inverse system identification, 126
 Invertible transformation, 17, 667
 IRLS. *See* Iterative reweighted Least Squares scheme (IRLS)
 IS. *See* Importance sampling (IS)
 ISI. *See* Intersymbol interference (ISI)
 Ising model, 765–767
 Isodata algorithm, 618
 Isometric mapping (ISOMAP), 987–991
 IST algorithms. *See* Iterative shrinkage/thresholding (IST) algorithms
 algorithms
 Iterative hard thresholding (IHT), 466–467, 466f
 Iterative refinement algorithm, 383–384
 Iterative reweighted Least Squares scheme (IRLS), 293, 471–472
 Iterative shrinkage/thresholding (IST) algorithms, 456–462
 Iterative soft thresholding (IST) algorithms, 466–467, 466f

J

Jacobian matrix, of transformation, 17–18
 Joint distribution, 748, 749
 Joint Gaussian pdf, 632–634
 Jointly distributed random variables, 77
 Jointly sufficient statistics, 71
 Joint pdf, 68, 71, 836–837
 Joint probabilities, 12–13
 Join tree, construction, 799–801
 Junction tree, 798, 801–804

K

Kalman filtering, 149, 851–854, 853f
 Kalman gain, 150–152, 246–247, 248, 257
 Karush-Kuhn-Tucker (KKT) conditions, 1025–1026
 Kernel APMS (KAPMS) algorithm, 560–565
 classification, 561–565
 nonlinear equalization, 564–565
 quantized, 562–563, 565
 regression, 560–561

- Kernel Hilbert spaces, 152
 Kernel LMS (KLMS), 553–556
 Kernel perceptron algorithm, 881–882
 Kernels, 96–97
 construction, 523–524
 covariance functions, 688–689
 function, 520–525
 matrix, 519, 688–689
 ridge regression, 528–530, 537–538
 trick, 517, 532, 537–538
 Kikuchi energy, 815–816
k-means algorithm, 618, 619
k-nearest neighbor density estimation, 97
k-nearest neighbor (*k*-NN) rule, 288–290
k-rank matrix approximation, 242
 KRRLS, 565
k-spectrum kernel, 525
 Kullback-Leibler distance, 305
 Kullback-Leibler (KL) divergence, 47–48
 EM algorithm, 608–609, 610^f
 mean field approximation, 642, 643
 minimizing, 680–681
 Kurtosis, 958, 1020–1021
- L**
- Labeled faces in the wild (LFW) database, 947
 Lagrange multipliers, 1024–1025
 Lagrangian, 205
 duality, 1027–1028
 function, 1024–1025
 Laplacian approximation, 662, 664
 evidence function, 596–600
 method, 596–600
 Laplacian kernel, 521
 Laplacian pdf, 668–670, 670^f, 671, 672^f
 Large scale tasks, 376
 LARS-LASSO algorithm, 454, 462
 Latent Markov model, 816–817, 818^f
 Latent variables, 606–610
 Lattice-ladder algorithm, 132
 forward/backward MSE optimal predictors, 134–138
 orthogonality of optimal backward errors, 138–140
 Toeplitz matrix, 133
 LDA. *See* Linear discriminant analysis (LDA)
 Learning, 1
 curve, 171–174
 from data, 1
 deep (*see* Deep learning)
 sparsity-aware (*see* Sparsity-aware learning)
 Least absolute shrinkage and selection operator (LASSO), 407–411
 adaptive norm-weighted, 477–478
 asymptotic performance, 475–477
 elastic net regularization, 472
 group, 467–468
 LARS algorithm, 454
 regularized cost function, 458
 Least angle regression (LARS) algorithm, 454, 466–467
 Least mean fourth (LMF) algorithm, 196–197
 Least-mean-square (LMS)
 adaptive algorithm, 179–188
 algorithm, 179–180, 368
 consensus matrix, 223–224
 convergence, 181–186, 199^f, 200^f, 201^f
 cumulative loss bounds, 186–188
 diffusion, 211–218
 distributed learning, 208–222
 H ∞ optimality of, 187
 linearly constrained, 204–206
 normalized, 193–194
 parameter estimation, 209
 recursion, 213
 relatives of, 196
 sign-error, 196
 steady-state performance, 181–186, 206
 target localization, 222–223
 time-varying model, 206–207
 tracking performance, 206–208
 transform-domain, 197–201
 widely-linear, 195
 Least modulus method, 530–531
 Least-squares (LS) estimator
 asymptotic distribution of, 238–239
 BLUE, 237–238
 covariance matrix, 236–237
 Cramer-Rao bound, 238
 loss criterion, 276, 308^f, 311
 unbiased, 236
 Least-squares method
 classifier, 61–62
 computational aspects, 255–257
 fitting plane, 60^f
 linear classifier, 63^f
 linear regression, 234–236, 235^f
 loss function, 56–57, 58–59, 59^f
 minimization task, 72–73
 optimal, 59

- regularization, 72–73
 ridge regression, 243–245
 unregularization, 72–73
- Leave-one-out (LOO) cross-validation method, 92–93
 Levenberg-Marquardt method, 376–377
 Levinson algorithm, 132–140
 Levinson-Durbin algorithm, 137
 Likelihood function, 82
 Linear classifier, 63f, 283
 Linear congruential generator, 709–710
 Linear convergence, 167
 Linear discriminant analysis (LDA), 286, 291
 Linear discriminant, Fisher's, 294–300
 Linear dynamical systems (LDS), 817–818
 Linear filtering, 35–36, 118–120
 Linear Gaussian models, 759–760
 Linear ϵ -insensitive loss function, 346, 347f, 530–537, 559
 Linear independency, 394
 Linear inverse problems, 438
 Linear kernel, 688
 Linearly constrained LMS, 204–206
 Linearly constrained minimum variance (LMV), 148
 Linearly separable classes, 515–517
 classes, 540–545
 probability, 515f
 two-dimensional plane, 516f
- Linear regression, 57–60
 Bayesian approach, 589–593
 dependencies, 646f
 EM algorithm, 610–612
 MAP estimator, 588–589
 ML estimator, 587
 nonwhite Gaussian noise case, 84
 variational Bayesian approach to, 645–651
- Linear space, 393
 Linear time invariant (LTI), 35–36, 512–514
 Linear varieties, 343, 343f
- LMF algorithm. *See* Least mean fourth (LMF) algorithm
 LMS. *See* Least-mean-square (LMS)
 LMV. *See* Linearly constrained minimum variance (LMV)
- ℓ_0 norm minimizer, 417–418
 equivalence, 426–429
 uniqueness, 422–426
- ℓ_1 norm minimizer, 418
 characterization, 419
 equivalence, 426–429
- ℓ_2 norm minimizer, 416f, 417
- Local independencies, 749–750
 Local linear embedding (LLE), 986–987
- Log-concave function, 805–807
 Log-convex function, 808
 Logistic regression, 290–294, 662–666
 Logistic sigmoid function, 290–291, 662, 887
 Log-likelihood function, 82–83, 292
 Log-loss function, 311–313
 Log-odds ratio, 311
 Log-partition, 815
 Loopy belief propagation, 813–816
 Loss functions
 empirical, 93–94
 expected, 93–94
 mean-square-error (*see* Mean-square-error (MSE) loss function)
 optimizing, 106
 parametric modeling, 56
- Loss matrix, 279–280
 Lower bound, computation, 650–651
 Low-rank matrix factorization method
 matrix completion, 991–994
 robust PCA, 995–996
- LTI. *See* Linear time invariant (LTI)
 LTIFIR filter, 137
- M**
- Magnetic resonance imaging (MRI), sparsity-promoting learning, 473–474
 Mahalanobis distance, 283, 285
 Majority voting rule, 306
 Majorization-minimization techniques, 458, 471
 Manifold learning, 434
 MAP estimator. *See* Maximum a posteriori probability (MAP) estimator
 Marginal pdf, 633–634
 Marginal probabilities, 13, 849
 Markov blanket, 758
 Markov chain Monte Carlo methods
 auxiliary variable, 735
 building, 724
 detailed balanced condition, 723
 ergodic, 723–728
 invariant distribution, 722
 reversible jump, 736
 transition probabilities matrix, 721–722
- Markov condition
 causality, 753–755
 completeness, 761–762
 definitions, 749
 d -separation, 755–758, 758f

- Markov condition (*Continued*)
 faithfulness, 761–762
 graphs, 749–753
 I-maps, 761–762
 linear Gaussian models, 759–760
 multiple-cause networks, 760, 761f
 soundness, 761–762
- Markov networks, 762
- Markov Random Fields (MRF), 762
 Boltzmann machine, 809f
 I-maps, 763–765
 independencies, 763–765
 Ising model, 765–767
- MARTs. *See* Multiple additive regression trees (MARTs)
- Matching Pursuit, 451
 CoSaMP, 455–456, 480
 CSMP algorithms, 455–456, 460, 479–480
 OMP, 451, 453, 466–467
- Matrices
 derivatives, 1014
 inversion lemmas, 1014
 positive definite and symmetric, 1015
 properties, 1013–1014
- Matrix completion, 991–994
 applications, 997
 collaborative filtering task, 996–997
- Maximal cliques, 763, 764f, 768
- Maximal spanning tree algorithm, 799
- Maximum a posteriori probability (MAP) estimator, 88–89, 588–589, 592
- Maximum entropy (ME) method, 47, 605–606
- Maximum likelihood (ML) method, 82–84, 82f, 277
 estimator, 587
 Type II, 600
- Maximum margin classifiers, 540–545
- Maximum variance unfolding method, 989
- Max-product algorithms, 782–789
- Max-sum algorithms, 782–789
- McCulloch-Pitts neuron, 880–881
- MDA. *See* Mirror descent algorithms (MDA)
- Mean, 15–17
- Mean field approximation, 641–645, 810–813
- Mean field equation, 811
- Mean field factorization, 810
- Mean square deviation (MSD), 358, 359f
- Mean-square-error (MSE), 65
 cost function, 176
 curves, 260f, 262f
 estimation, 77–78
- iteration functions, 565, 566f
 linear estimator, 178–179
 local cost function, 212–213
 values, 76, 76t
- Mean-square error linear estimation, 105–106, 141–148
 complex-valued variables, 111–118
 constrained linear estimation, 145–148
 cost function, 107–108, 108f
 deconvolution, 121–124, 126–132
 Gauss-Markov theorem, 143–145
 geometric viewpoint, 109–111
 interference cancellation, 124–125
 Kalman filtering, 149
 Lattice-ladder algorithm, 132–140
 Levinson algorithm, 132–140
 linear filtering, 118–120, 119f, 120–124
 minimum, 110f
 normal equations, 106–108
 optimal equalizer, 130–131
 system identification, 125–126
- Mean-square-error (MSE) loss function
 complex-valued case, 175–176
 cost function, 167
 cross-correlation vector, 171–174
 error curves, 171–174
 gradient descent algorithm, 173f
 learning curve, 171–174
 minimum eigenvalue, 169–171
 parameter error vector convergence, 171
 time constant, 169
 time-varying step sizes, 174–176
- Mean-square sense, convergence in, 49
- Measurement noise, 149–150
- Mercedes Benz (MB) frame, 499–500
- Message-passing algorithms, 460–462
 exact inference methods, 773–789
 junction tree, 801–804
 max-product algorithms, 782–789
 max-sum algorithms, 782–789
 sum-product algorithm, 778–782
 two-way, 801–803
- Metropolis-Hastings algorithm, 729, 730, 735, 736
- Metropolis method, 728–729
 burn-in phase, 730
 convergence issues, 731–732
- MIMO systems. *See* Multiple-input-multiple-output (MIMO) systems
- Minimum distance classifiers, 285–287

- Minimum variance distortionless response (MVDR)
beamforming, 148
- Minimum variance unbiased estimator (MVUE), 66–67, 144–145, 238
- Min-Max duality, 1026–1027
- Mirror descent algorithms (MDA), 388–389
- Misclassification error, Bayesian classification, 277–280
- Mixing linear regression models, 622–625
HME, 625, 626f
mixture of experts, 624–625
- Mixing logistic regression models, 625–627
- Mixing of learners, 621
- Mixing time, 730
- Mixture of experts, 621, 621f, 624–625
- Mixture of factor analyzers (MSA), 978–979
- Mixture scatter matrix, 296
- ML method. *See* Maximum likelihood (ML) method
- Mode, 169–171, 170f
- Model-based Compressed Sensing, 468–469
- Modulated wideband converter (MWC), 435–436
- Moment generating function, 1020
- Moment matching, 680–681, 682
- Moment projection (M-projection), 680
- Moments, 1020–1021
- Monte Carlo methods, 709
advantages, 736–737
change-point detection, 737–738
concepts, 708–710
EM algorithm, 720–721
Gibbs sampling, 733–735
Hamiltonian function, 736
importance sampling, 718–720
Markov chain, 721–728
Metropolis method, 728–732
random sampling, 711–715
rejection sampling, 715–718
- Moore-Penrose pseudo-inverse, 234–236
- Moreau envelop, 379, 381f, 460
- Moreau-Yosida regularization, 379
- Moving average model, 40
- Multichannel estimation, 112–113, 141
- Multiclass Fisher's discriminant, 299–300
- Multiclass generalizations, SVM, 552–553
- Multidimensional Scaling (MDS), 946
- Multinomial distribution, 19–20
- Multinomial resampling, 847
- Multiple additive regression trees (MARTs), 314
- Multiple-cause Bayesian networks, 760, 761f, 805–807
- Multiple-input-multiple-output (MIMO) systems, 412–413
- Multiple kernel learning (MKL), 567–568
- Multiple measurement vectors (MMV), 659
- Multipulse signals, 436
- Multitask learning, 548
- Multivariate Gaussian distribution, 21–22, 24
- Multivariate linear regression (MLR), 955
- Mutual coherence, 424–426
- Mutual information, 43–44
- MVDR beamforming. *See* Minimum variance distortionless response (MVDR) beamforming
- MVUE. *See* Minimum variance unbiased estimator (MVUE)
- MWC. *See* Modulated wideband converter (MWC)
- N**
- Naive Bayes classifier, 287–288
- Naive online R_{reg} minimization algorithm (NORMA), 556–560
- Natural gradient, 376–377
- Natural parameters, 600
- Near-end speech signal, 125
- Nearest neighbor rule, 288–290
- Near-to-Toeplitz, 256–257
- NESTA algorithm, 461, 472–473, 490, 495–496
- Neural networks
backpropagation algorithm, 886–897
Bayesian learning, 902–903
feed-forward, 882–886
gradient descent scheme, 887–894
perceptron algorithm, 876
pruning, 897–899
synapses, 876
- Newton's iterative minimization method, 248–251
- Newton's scheme, 293
- NLMS. *See* Normalized least mean square (NLMS)
- Noise cancellation, 127, 127f
- Noisy-OR model, 746–747, 805–807
- Nonempty set, 683
- Noninformative/objective priors, 599
- Nonlinear dimensionality reduction
ISOMAP, 987–991
kernel PCA, 980–982
Laplacian eigenmaps, 982–986
local linear embedding, 986–987
- Nonlinear filter, 512f
- Nonlinear manifold learning, 979
- Nonnegative garrote, 411, 412f
- Nonnegative matrix factorization (NMF), 971–972
- Non-negative real function, 38
- Nonoverlapping training sets, 93

- Nonparametric Bayesian modeling, 54, 95–97, 683–686
 estimation, 95–97
 representer theorem, 528
- Nonparametric sparsity-aware learning, 568–570
- Nonseparable classes, SVM, 545–548
- Nonsmooth convex cost functions
 linearly separable, 364
 minimizing, 362–367
 optimizing, 358–370
 subdifferentials, 359–362
 subgradients, 359–362, 363–365
- Nonstationary environments, LMS, 206–208
- Nonwhite Gaussian noise, 84
- Norm, 395
 definition, 404–405
 ℓ_0 minimizer, 417–418, 422–429
 ℓ_1 minimizer, 418, 419, 426–429
 ℓ_2 minimizer, 416f, 417
 searching for, 404–407
- Normal distribution, 20–24
- Normal equations, 110
- Normal factor graph (NFG), 770
- Normalized graph Laplacian matrix, 984–985
- Normalized least mean square (NLMS)
 convex analytic path, 353
 stochastic gradient descent, 193–194, 201, 202f
- Normed linear space, 395
- Nucleobases, 314–315
- Nucleotides, 314–316
- O**
- OBCT. *See* Ordinary binary classification trees (OBCT)
- Observations, 57–58
- Occam’s razor rule, 78–79, 593–600, 643
- OCR systems. *See* Optical character recognition (OCR) systems
- OMP. *See* Orthogonal matching pursuit (OMP)
- One-against-all, 552
- One-against-one, 552
- One pixel camera, 432–433
- Online cyclic coordinate descent time weighted LASSO (OCCD-TWL), 478, 484f
- Online learning
 approximation error, 374–376
 batch vs., 376–379
 and big data applications, 374–379
 convex, 367–370
 estimation error, 374–376
 expected loss/risk function, 374
 optimization error, 374–376
 techniques, 162
- Online perceptron algorithm, 880
- Optical character recognition (OCR) systems, 2, 923
- Optimal brain damage technique, 898
- Optimal brain surgeon method, 898
- Optimal linear estimation, 124
- Optimization error, 374–376
- Order statistics, 30
- Ordinary binary classification trees (OBCT), 300–301, 300f, 304
- Ordinary-differential-equation approach (ODE), 187
- Ornstein–Uhlenbeck kernel, 689
- Orthogonality
 geometric viewpoint, 109–111
 optimal backward errors, 138–140
- Orthogonal matching pursuit (OMP), 451
 algorithm, 466–467
 recover optimal sparse solutions, 453
- Orthogonal projection, 109, 110
- Outlier, 537
- Output variable, 57–58
- Overcomplete dictionaries, 414–415
- Overdetermined system, 240–242
- Overfitting, 74–76
- P**
- Pairwise MRFs undirected graphs, 766, 767f
- Parallelogram law, 396
- Parameter error vector convergence, 171
- Parametric functional form, 54–55
- Parametric modeling, 53
 curve fitting problem, 54–55, 55f
 deterministic point of view, 54–57
 loss function, 56
 nonnegative function, 56
- Parity-check bits, 770–771
- Parseval tight frame, 498–499
- Partial least-squares (PLS) method, 954–955
- Particle filtering, 854
 auxiliary, 862–868
 degeneracy phenomenon, 858–860
 generic, 860–861
 one-dimensional random walk model, 857–858
 SIS, 855, 856
 state-space model, 853f, 854–855
- Parzen windows, 96–97
- Path, 749

Pattern, 60–61
 Pattern recognition, 60–61, 91, 295–296
 Peak signal-to-noise ratio (PSNR), 438–439
 Perceptron algorithm, 364–365, 876, 878
 Perceptron cost, 877–882
 Perfect elimination sequence, 798
 Perron-Frobenius theorem, 722
 Persistent contrastive divergence (PCD) algorithm, 913, 914
 PGM. *See* Projected gradient method (PGM)
 pmf. *See* Probability mass function (pmf)
 POCS. *See* Projections onto convex sets (POCS)
 Poisson process, 737–738
 Polynomial kernel
 homogeneous, 520
 inhomogeneous, 520, 522f
 Population-based methods, 735–736
 Positive definite, 16, 34, 1015
 Positive definite kernel, 519, 520
 Positive semidefinite, 1015
 Posteriori probability, 63–64, 276
 Potential functions, 96–97, 762
 Potts model, 766
 Power spectral density (PSD), 33–38, 120–121
 definition, 35
 physical interpretation of, 37–38
 Prediction, 118, 186
 Preprocessing stage, 32–33
 Primal estimated subgradient solver for SVM (PEGASOS)
 algorithm, 369–370, 551
 Principal axes/directions, 244–245
 Principal component pursuit (PCP), 995
 Principal components regression, 244–245
 Principia Mathematica, 754–755
 Principle component analysis (PCA)
 eigenimages/eigenfaces, 947, 947f, 948f
 feature generation, 943–944
 latent semantics indexing, 943
 latent variables, 944–949
 LFW database, 947
 low-rank matrix factorization method, 942
 minimum error interpretation, 943
 mutually uncorrelated, 943–944
 online subspace tracking, 949, 950f
 optimization task, 940–941
 principle directions, 940
 supervised PCA, 947
 SVD decomposition, 941, 942
 Probabilistic PCA (PPCA), 974–977

Probability density function (pdf), 10
 beta distribution, 26f
 definition, 18f
 edgeworth expansion, 1021–1022
 gamma distribution, 27f
 Gaussian, 22f, 24
 uniform distribution, 21f
 Probability distributions
 exponential family, 600–606, 644–645
 random walk chain, 726–728, 727f, 728f
 Probability mass function (pmf), 12, 19f
 Probit regression, 294
 Process noise, 149–150
 Product rule of probability, 13
 Projected gradient method (PGM), 365–366
 Projected Landweber method, 366
 Projected subgradient method, 366–367
 Projection approximation subspace tracking
 (PAST), 949
 Projections onto convex sets (POCS)
 algorithm, 344
 analytical expressions, 335–336
 classification, 347–349
 concepts, 333–335
 fundamental theorem, 341–344
 halfspace, 336f
 hyperplane, 336f
 intersection, 345–346
 linear varieties, 343, 343f
 nonempty intersection, 341, 342
 nonexpansiveness property, 340f, 343–344
 parallel version, 344
 product spaces, 344
 properties, 337–341
 regression, 345–347
 relaxed, 339–340, 340f, 341f
 weak convergence, 342
 Property sets, 349
 Proportionate NLMS, 194
 Protein folding prediction, 314–318
 Proteinogenic amino-acids, 314–315
 Proximal forward-backward splitting operator, 386–387
 Proximal gradient splitting algorithms, 385–386
 Proximal mapping, 460
 Proximal operators, 379–385
 minimization, 383–385
 properties, 382–383
 splitting methods, 385–389
 subdifferential mapping, 384–385

- Pruning, neural networks
 convolutional networks, 899
 early stopping, 898–899
 optimal brain damage technique, 898
 weight decay, 897
 weight elimination, 897
 weight sharing, 898
- Pruning tree, 303–304
- PSD. *See* Power spectral density (PSD)
- Pseudo covariance, 114–115
- Pseudo-inverse matrix, 240–242
- Pseudorandom generator, 709–710, 711
- Q**
- QR factorization, 255–256
- Quadratic discriminant analysis (QDA), 286
- Quadratic form exponent, pdfs with, 631
- Quadratic ϵ -insensitive loss function, 530–531, 531f, 536
- Quantized KLMS (QKLMS), 555–556, 565
- Quasi-stationary process, 818
- Quickprop algorithm, 895
- R**
- Random demodulator (RD), 432, 434–435, 436
- Random field, 121
- Random forests, 303–304
- Random-modulation pre-integrator (RMPI), 434–435
- Random number generation, 709–710
- Random sampling, 711–715
- Random signal, 29
- Random variables
 axiomatic definition, 11–12
 complex, 16–17
 continuous (*see* Continuous random variables)
 discrete (*see* Discrete random variables)
 geometric interpretation of, 109–111
 probability and, 10–18
 relative frequency definition, 11
 transformation of, 17–18
- Random vector, 15
- Rao-Blackwellization technique, 866
- Rao-Blackwell theorem, 70, 71, 735
- Rate of convergence, 457–458
- Rational quadratic kernel, 689
- Rayleigh fading channel, 342
- RD. *See* Random demodulator (RD)
- Real linear space, 394
- Recursive least-squares (RLS) algorithm, 234, 245–248
- convergence curve, 354
 fast versions, 256–257
 Newton's method, 251
 simulation examples, 259, 260–261
 steady state performance, 252–254
 widely linear, 254–255
- Reduced convex hull interpretation, 548
- Regression, 3–8
 Bayesian, 690–692
 deming, 262
 errors-in-variables, 262
 input-output relation, 57f
 KAPSM algorithm, 560–561
 least-squares linear, 234–236, 235f
 linear, 57–60
 linear ϵ -insensitive loss function, 559
 POCS, 345–347
 principal components, 244–245
 ridge, 243–245
- Regressor, 57–58
- Regret analysis, 367–368, 370–374
- Regularity assumption, 1023
- Regularization, 72–76
 Regularized dual averaging (ARD) algorithm, 388
 Regularized particle filter, 866
 Rejection sampling, 715–718
 Relative entropy, 47–48, 896–897
 Relevance vector machine (RVM), 661–666
 Relevance vectors, 662
 Representer theorem, 525–528
 nonparametric, 528
 semiparametric, 527
- Reproducing kernel Hilbert spaces (RKHS), 95, 517–518, 662
 authorship identification, 570–573
 definition, 510
 generalized linear models, 510–511
 KAPSM algorithm, 560–565
 kernel functions, 520–525
 kernel LMS, 553–556
 kernel trick, 517
 NORMA, 556–560
 properties, 519–520
 representer theorem, 525–528
 ridge regression, 528–530, 537–538
 theoretical highlights, 519–520
- Resampling, 847–849, 851
- Restricted Boltzmann machine (RBM), 905–906, 908–914
- Restricted isometry property (RIP), 427–429
- Reversible jump Markov chain Monte Carlo algorithms, 736

- Ridge regression, 72–73, 243–245
 kernels, 528–530, 537–538
 principal components regression, 244–245
- Right stochastic matrix, 213–214
- Ring networks, 210
- Ring topology, 211*f*
- RIP. *See* Restricted isometry property (RIP)
- RKHS. *See* Reproducing kernel Hilbert spaces (RKHS)
- RLS algorithm. *See* Recursive least-squares (RLS) algorithm
- Robbins-Monro algorithm, 177–179
- Robust loss functions, 311–313
- Robust PCA
 applications of, 997
 low-rank matrix factorization, 995–996
- Robust sparse signal recovery, 429–430
- Running intersection property, 798
- S**
- Saddle point condition, 1027
- Salienties, 898
- Sample mean, 31
- Sample sequences, 29
- Sample space, 12
- Sampling-importance-resampling (SIR), 860–861
- SCAD. *See* Smoothly clipped absolute deviation (SCAD)
- Scatter matrices, 295–296
- Schur algorithm, 140
- Schur complement, 133–134
- Search direction, 163
- Second order convexity condition, 331
- Segmental k -means training algorithm, 827–828
- Semiparametric representer theorem, 527
- Semisupervised learning, 3, 64, 202–203
- Separator, 799, 801–804
- Separator nodes, 802–803
- Sequential importance sampling (SIS), 845–846, 850
 importance sampling revisited, 846–847
 particle filtering, 855, 856
 resampling, 847–849, 851
 sequential sampling, 849–851
- Serial connection, 755
- Set-membership algorithms, 354
- Shepp-Logan image phantom, 474*f*
- Shrinkage methods, 314
- Sigmoidal Bayesian networks, 758–759, 759*f*, 760
- Sigmoid link function, 290, 291*f*
- Signal compression, 412–413
- Signal processing, filtering, 118
- Signal restoration, 438
- Sign-error LMS, 196
- Sinc kernel, 523
- Single-layered feed-forward networks (SLFNs), 900
- Single-stage auxiliary particle filter, 865–867
- Singular value decomposition (SVD), 239–242, 941, 942
- SIS. *See* Sequential importance sampling (SIS)
- Slab method, 660–661
- Slack variables, 532
- SLDS. *See* Switching linear dynamic systems (SLDS)
- Slice-sampling algorithm, 735
- Small scale tasks, 376
- Smoothing, 118, 852, 866
- Smoothly clipped absolute deviation (SCAD), 411, 412*f*, 478
- Softmax activation function, 624, 896–897
- Soft thresholding, 380–381
 function, 456–457
 operation, 409–411, 410*f*
- Soundness, 761
- Sparse adaptive projection subgradient method (SpAPSM), 480–484, 484*f*, 484*f*
- Sparse analysis representation, 485–486
- Sparse Bayesian Learning (SBL), 657–660
- Sparse factor analysis, 977
- Sparse modeling, 404
- Sparse reconstruction by separable approximation (SpaRSA) algorithm, 458–459
- Sparse signal representation, 411–415, 487–488
- Sparse solutions, 453, 475
- Sparsity-aware learning, 385, 404
 Bayesian approach to, 655–661
 concave, 675–676
 cost function, 675–676
 Cramer-Rao bound, 679
 de-noising, 438–439
 geometric interpretation, 419–422
 least absolute shrinkage/selection operator, 407–411
 ℓ_0 norm minimizer, 417–418, 422–429
 ℓ_1 norm minimizer, 418, 419, 426–429
 ℓ_2 norm minimizer, 416*f*, 417
 models, 485–490
 nondecreasing, 675–676
 parameter identifiability, 678–679
 robust sparse signal recovery, 429–430
 searching for norm, 404–407
 techniques, 404
 variational parameters, 677
 variations on, 467–474
- Sparsity-aware regression, 671–675

- Sparsity-promoting algorithms, 356, 450
 adaptive norm-weighted LASSO, 477–478
 AdCoSaMP algorithm, 479–480
 distributed, 483
 frames theory, 497–502
 greedy algorithms, 451–456
 iterative shrinkage/thresholding algorithms, 456–462
 LASSO, 475–477
 magnetic resonance imaging, 473–474
 phase transition behavior, 464–465, 465f
 practical hints, 462–467
 SpAPSM, 480–484
 Spectral signature, 694
 Spectral unmixing (SU), 694–695
 Spike method, 660–661
 Spline kernels, 521
 Split Levinson algorithm, 137
 Splitting criterion, 300–301, 302–303
 Squared-error loss function, 234
 Squared exponential kernel, 688
 Squashing function, 887
 SSS. *See* Strict-sense stationarity (SSS)
 Stable embedding, 433–434
 Stacking, 306
 State equation, 149–150
 State-observation models, 816–817
 State-space models, 149–150, 816–817
 Kalman filters, 853f
 particle filters, 853f, 854–855
 Stationarity, 30
 Stationary iterative/iterative relaxation methods,
 456–457
 Statistical filtering, 119f
 Statistical independence, 13
 Statistical signal processing, 5
 Steady-state performance, 218–219
 distributed learning, 218–219
 improving, 219
 LMS in stationary environments, 181–186
 of RLS, 252–254
 Steepest descent method, 163–167, 293
 Stick-breaking construction, 685–686
 Stochastic approximation, 177–179, 251
 Stochastic convergence
 almost everywhere, 49
 distribution, 49–51
 everywhere, 48
 mean square sense, 49
 probability, 49
 Stochastic EM, 720–721
 Stochastic gradient descent schemes, 178–179
 Stochastic processes, 29
 autoregressive models, 38–40
 first/second order statistics, 30
 power spectral density, 33–38
 stationarity/ergodicity, 30–33
 Stochastic volatility model, 867
 Stop-splitting rule, 303
 Strict-sense stationarity (SSS), 30, 31
 String kernels, 525
 Strongly convex auxiliary function, 388
 Structured sparsity, 467, 468–469
 Subband adaptive filters, 198
 Subdifferential mapping
 proximal operators, 384–385
 resolvent of, 384–385
 Sub-Gaussian distribution, 196–197
 Subgradient algorithm, 359–362, 363–365
 generic scheme, 365
 regret analysis of, 372–374
 Subjective priors, 599
 Sublinear global rate of convergence, 457–458
 Sub-Nyquist sampling
 analog-to-information conversion, 434–436
 definition, 434–435
 Sufficient statistics, 70–72
 Sum-product algorithm, 778–782, 821
 Supervised learning, 3, 64
 Support vector machine (SVM), 369, 538–539, 665
 applications, 550
 division and clifford algebras, 552
 hyperparameters, 550–551
 linearly separable classes, 540–545
 multiclass generalizations, 552–553
 nonseparable classes, 545–548
 one-against-all, 552
 one-against-one, 552
 PEGASOS, 551
 performance, 550
 Support vector regression (SVR), 662
 linear ϵ -insensitive loss function, 530–537, 559
 optimization task, 530–531
 Support vectors, 533, 543
 Switching linear dynamic systems (SLDS), 832
 Synapses, 876
 Systematic resampling, 848
 System identification, 125–126

T

- Tail-to-tail connection, 756
- Test error, 80–81
- Thinning process, 731
- Tight frames, 498–499
- Time-adaptive algorithm, 162
- Time-and-norm-weighted LASSO (TNWL), 477–478
- Time constant, 169, 185–186
- Time-frequency analysis
 - echolocation signals, 493–497
 - Gabor frames, 490–492, 493
 - Gabor transform, 490–492
 - time-frequency resolution, 492–493
- Time-frequency resolution, 492–493
- Time sequential nature, 140
- Time-shifted versions, 132
- Time-shift structure, 256–257
- Time varying signal, 483–484
- Time varying statistics, 149
- Time-varying step sizes, 174–176
- TNWL. *See* Time-and-norm-weighted LASSO (TNWL)
- Toeplitz matrix, 40, 133
- Total-least-squares (TLS) method, 261–268
- Training data set, 57–58
- Training deep networks
 - backpropagation, 907
 - distributive representation, 907
 - feed-forward networks, 914–915
 - restricted Boltzmann machine, 905–906, 908–914
 - sparsity, 907
- Training error, 80–81, 92f
- Training set, 2
- Transform-domain LMS, 197–201
- Transition probability matrix, 722–723, 724, 725
 - detailed balanced condition, 723
 - hidden Markov model, 818, 819
 - Markov chains, 724, 725–726
 - properties, 722–723
- Transversal implementation, LTIFIR filter, 137
- Tree reweighted belief propagation, 815
- Trees
 - boosting, 313–314
 - classification, 300–304, 301f
 - exact inference methods, 777–778
- Triangle inequality, 395
- Triangulated graphs, 796–804
 - Bayesian network, 800–801, 800f
 - undirected graph, 796, 797f, 799

Two-stage-thresholding (TST) algorithms, 460, 466–467, 466f

Tychonoff-Phillips regularization, 72

Type I estimator, 592

Type II maximum likelihood, 600

U

- Unbiased estimation, 31, 65–67
- Underdetermined system, 240–242
- Undirected graph
 - perfect elimination sequence, 798
 - triangulated graph, 796, 797f, 799
- Undirected graphical models, 762–768
 - CRFs, 767–768
 - independencies/I-maps in Markov random fields, 763–765
 - Ising model, 765–767
- Uniform distribution, 20, 712f
- Union of subspaces, 433
- Unit vector, 193f
- Unobserved random variable, 57–58
- Unscented Kalman filters, 152
- Unsupervised learning, 3, 64
- Update direction, 163

V

- Validation, 91–93
- Value similarity (VS), 572–573
- Variable duration HMM, 829
- Variable elimination, 781
- Variance, 15–17
- Variational approximation methods, 804–805
 - Bayesian learning, 640–645
 - block methods, 809–813
 - Boltzmann machine, 807–809
 - multiple-cause networks, 805–807
 - noisy-OR model, 805–807
- Variational Bayesian approach
 - to Gaussian mixture modeling, 651–654
 - to linear regression, 645–651
- Variational bound approximation method, 666–671
- Variational bound Bayesian path, 671–675
- Variational inference techniques, 736–737
- Variational message passing, 812
- Variational method, 670
- VC-dimension of classifier, 91
- Vector space model (VSM), 570–571
- Vector spaces, 109, 394
- Vertex component analysis (VCA) algorithm, 697
- Visual tracking, 867–868

Viterbi algorithm, 825
 Viterbi reestimation, 827–828
 insufficient training data set, 828
 scaling, 828
 Volterra model, 511–514
 Volterra series expansion, 511
 v -SVM, 547

W

Weak convergence, 342, 397
 Weierstrass theorem, 510–511
 Welch bound, 424–425
 Well-posed problems, 74
 White Gaussian noise, 238
 White noise
 LS estimator, 237–238
 sequence, 38, 42f
 Widely-linear APA, 195–196
 Widely linear complex-valued estimation, 113–116
 Widely-linear LMS, 195
 Wide-sense stationary (WSS), 30, 31
 cross-correlation, 34

real random processes, 119
 two-dimensional random process, 121
 Wiener-Hammerstein model, 512–514, 512f
 Wiener-Hopf equations, 110
 Wiener model, 511–514, 512f
 Wireless sensor networks (WSNs), 208
 Wirtinger calculus, 116–118, 175, 1016–1017
 Wirtinger derivative, 117
 Wishart distribution, 601
 Within-class scatter matrix, 296
 Wolfe dual representation, 1029
 Woodbury’s matrix inversion formula, 246–247
 WSS. *See* Wide-sense stationary (WSS)

X

X-ray mammography, 2

Y

Yule-Walker equations, 39–40

Z

Zero mean values, 32–33, 106–107