

Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series

A First Course in Machine Learning

Simon Rogers
Mark Girolami



CRC Press

Taylor & Francis Group

A CHAPMAN & HALL BOOK

A First Course in
**Machine
Learning**

Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series

SERIES EDITORS

Ralf Herbrich and Thore Graepel

Microsoft Research Ltd.

Cambridge, UK

AIMS AND SCOPE

This series reflects the latest advances and applications in machine learning and pattern recognition through the publication of a broad range of reference works, textbooks, and handbooks. The inclusion of concrete examples, applications, and methods is highly encouraged. The scope of the series includes, but is not limited to, titles in the areas of machine learning, pattern recognition, computational intelligence, robotics, computational/statistical learning theory, natural language processing, computer vision, game AI, game theory, neural networks, computational neuroscience, and other relevant topics, such as machine learning applied to bioinformatics or cognitive science, which might be proposed by potential contributors.

PUBLISHED TITLES

MACHINE LEARNING: An Algorithmic Perspective

Stephen Marsland

HANDBOOK OF NATURAL LANGUAGE PROCESSING,

Second Edition

Nitin Indurkha and Fred J. Damerau

UTILITY-BASED LEARNING FROM DATA

Craig Friedman and Sven Sandow

A FIRST COURSE IN MACHINE LEARNING

Simon Rogers and Mark Girolami

Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series

A First Course in Machine Learning



Simon Rogers
Mark Girolami



CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

MATLAB® is a trademark of The MathWorks, Inc. and is used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB® software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB® software.

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2012 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed in the United States of America on acid-free paper
Version Date: 2011908

International Standard Book Number: 978-1-4398-2414-6 (Hardback)

This book contains information obtained from authentic and highly regarded sources. Reasonable efforts have been made to publish reliable data and information, but the author and publisher cannot assume responsibility for the validity of all materials or the consequences of their use. The authors and publishers have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission to publish in this form has not been obtained. If any copyright material has not been acknowledged please write and let us know so we may rectify in any future reprint.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (<http://www.copyright.com/>) or contact the Copyright Clearance Center, Inc. (CCC), 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Visit the Taylor & Francis Web site at
<http://www.taylorandfrancis.com>

and the CRC Press Web site at
<http://www.crcpress.com>

Contents

List of Tables	xi
List of Figures	xiii
Preface	xix
1 Linear Modelling: A Least Squares Approach	1
1.1 Linear modelling	1
1.1.1 Defining the model	2
1.1.2 Modelling assumptions	3
1.1.3 Defining what a <i>good</i> model is	4
1.1.4 The least squares solution – a worked example	6
1.1.5 Worked example	9
1.1.6 Least squares fit to the Olympics data	10
1.1.7 Summary	11
1.2 Making predictions	12
1.2.1 A second Olympics dataset	12
1.2.2 Summary	15
1.3 Vector/matrix notation	15
1.3.1 Example	22
1.3.2 Numerical example	23
1.3.3 Making predictions	24
1.3.4 Summary	24
1.4 Nonlinear response from a linear model	25
1.5 Generalisation and over-fitting	28
1.5.1 Validation data	29
1.5.2 Cross-validation	29
1.5.3 Computational scaling of K -fold cross-validation	32
1.6 Regularised least squares	33
1.7 Exercises	35
Further reading	37
2 Linear Modelling: A Maximum Likelihood Approach	39
2.1 Errors as noise	39
2.1.1 Thinking generatively	40
2.2 Random variables and probability	41

2.2.1	Random variables	41
2.2.2	Probability and distributions	42
2.2.3	Adding probabilities	44
2.2.4	Conditional probabilities	44
2.2.5	Joint probabilities	45
2.2.6	Marginalisation	47
2.2.7	Aside – Bayes’ rule	49
2.2.8	Expectations	50
2.3	Popular discrete distributions	53
2.3.1	Bernoulli distribution	53
2.3.2	Binomial distribution	53
2.3.3	Multinomial distribution	54
2.4	Continuous random variables – density functions	55
2.5	Popular continuous density functions	58
2.5.1	The uniform density function	58
2.5.2	The beta density function	60
2.5.3	The Gaussian density function	61
2.5.4	Multivariate Gaussian	62
2.5.5	Summary	65
2.6	Thinking generatively...continued	65
2.7	Likelihood	67
2.7.1	Dataset likelihood	68
2.7.2	Maximum likelihood	69
2.7.3	Characteristics of the maximum likelihood solution	71
2.7.4	Maximum likelihood favours complex models	74
2.8	The bias-variance tradeoff	75
2.8.1	Summary	75
2.9	Effect of noise on parameter estimates	76
2.9.1	Uncertainty in estimates	78
2.9.2	Comparison with empirical values	81
2.9.3	Variability in model parameters – Olympics data	82
2.10	Variability in predictions	83
2.10.1	Predictive variability – an example	85
2.10.2	Expected values of the estimators	86
2.10.3	Summary	90
2.11	Exercises	90
	Further reading	93
3	The Bayesian Approach to Machine Learning	95
3.1	A coin game	95
3.1.1	Counting heads	97
3.1.2	The Bayesian way	98
3.2	The exact posterior	103
3.3	The three scenarios	104
3.3.1	No prior knowledge	104

3.3.2	The fair coin scenario	111
3.3.3	A biased coin	114
3.3.4	The three scenarios – a summary	116
3.3.5	Adding more data	116
3.4	Marginal likelihoods	117
3.4.1	Model comparison with the marginal likelihood	118
3.5	Hyper-parameters	119
3.6	Graphical models	120
3.6.1	Summary	121
3.7	A Bayesian treatment of the Olympics 100 m data	122
3.7.1	The model	122
3.7.2	The likelihood	124
3.7.3	The prior	124
3.7.4	The posterior	124
3.7.5	A first-order polynomial	126
3.7.6	Making predictions	129
3.8	Marginal likelihood for polynomial model order selection	131
3.9	Chapter summary	133
3.10	Exercises	133
	Further reading	137
4	Bayesian Inference	139
4.1	Non-conjugate models	139
4.2	Binary responses	140
4.2.1	A model for binary responses	140
4.3	A point estimate – the MAP solution	143
4.4	The Laplace approximation	149
4.4.1	Laplace approximation example: Approximating a gamma density	150
4.4.2	Laplace approximation for the binary response model	151
4.5	Sampling techniques	154
4.5.1	Playing darts	154
4.5.2	The Metropolis–Hastings algorithm	156
4.5.3	The art of sampling	164
4.6	Summary	165
4.7	Exercises	165
	Further reading	167
5	Classification	169
5.1	The general problem	169
5.2	Probabilistic classifiers	170
5.2.1	The Bayes classifier	170
5.2.1.1	Likelihood – class-conditional distributions	171
5.2.1.2	Prior class distribution	171
5.2.1.3	Example – Gaussian class-conditionals	172

5.2.1.4	Making predictions	173
5.2.1.5	The naive Bayes assumption	175
5.2.1.6	Example – classifying text	175
5.2.1.7	Smoothing	177
5.2.2	Logistic regression	179
5.2.2.1	Motivation	180
5.2.2.2	Nonlinear decision functions	181
5.2.2.3	Nonparametric models - the Gaussian process	182
5.3	Nonprobabilistic classifiers	183
5.3.1	K -nearest neighbours	183
5.3.1.1	Choosing K	184
5.3.2	Support vector machines and other kernel methods	186
5.3.2.1	The margin	186
5.3.2.2	Maximising the margin	187
5.3.2.3	Making predictions	190
5.3.2.4	Support vectors	191
5.3.2.5	Soft margins	192
5.3.2.6	Kernels	193
5.3.3	Summary	197
5.4	Assessing classification performance	198
5.4.1	Accuracy – 0/1 loss	198
5.4.2	Sensitivity and specificity	198
5.4.3	The area under the ROC curve	199
5.4.4	Confusion matrices	201
5.5	Discriminative and generative classifiers	203
5.6	Summary	203
5.7	Exercises	203
	Further reading	205
6	Clustering	207
6.1	The general problem	207
6.2	K -means clustering	208
6.2.1	Choosing the number of clusters	210
6.2.2	Where K -means fails	212
6.2.3	Kernelised K -means	212
6.2.4	Summary	214
6.3	Mixture models	215
6.3.1	A generative process	216
6.3.2	Mixture model likelihood	217
6.3.3	The EM algorithm	219
6.3.3.1	Updating π_k	220
6.3.3.2	Updating μ_k	221
6.3.3.3	Updating Σ_k	222
6.3.3.4	Updating q_{nk}	223
6.3.3.5	Some intuition	224

6.3.4	Example	225
6.3.5	EM finds local optima	226
6.3.6	Choosing the number of components	228
6.3.7	Other forms of mixture components	230
6.3.8	MAP estimates with EM	232
6.3.9	Bayesian mixture models	233
6.4	Summary	234
6.5	Exercises	234
	Further reading	237
7	Principal Components Analysis and Latent Variable Models	239
7.1	The general problem	239
7.1.1	Variance as a proxy for interest	239
7.2	Principal components analysis	242
7.2.1	Choosing D	247
7.2.2	Limitations of PCA	247
7.3	Latent variable models	248
7.3.1	Mixture models as latent variable models	248
7.3.2	Summary	249
7.4	Variational Bayes	249
7.4.1	Choosing $Q(\theta)$	251
7.4.2	Optimising the bound	252
7.5	A probabilistic model for PCA	252
7.5.1	$Q_\tau(\tau)$	254
7.5.2	$Q_{\mathbf{x}_n}(\mathbf{x}_n)$	256
7.5.3	$Q_{\mathbf{w}_m}(\mathbf{w}_m)$	257
7.5.4	The required expectations	258
7.5.5	The algorithm	258
7.5.6	An example	260
7.6	Missing values	260
7.6.1	Missing values as latent variables	262
7.6.2	Predicting missing values	264
7.7	Non-real-valued data	264
7.7.1	Probit PPCA	264
7.7.2	Visualising parliamentary data	268
7.7.2.1	Aside – relationship to classification	272
7.8	Summary	273
7.9	Exercises	273
	Further reading	275
	Glossary	277
	Index	283

List of Tables

1.1	Synthetic dataset for linear regression example.	9
1.2	Olympics men's 100 m data.	11
1.3	Olympics women's 100 m data.	13
1.4	Some useful identities when differentiating with respect to a vector.	21
2.1	Events we might want to model with random variables. . . .	42
5.1	Likelihood and priors for $\mathbf{x}_{\text{new}} = [2, 0]^T$ for the Gaussian class- conditional Bayesian classification example.	174
5.2	A binary confusion matrix.	201
5.3	Confusion matrix for the 20 class newsgroup data.	202

List of Figures

1.1	Winning men's 100 m times at the Summer Olympics since 1896.	2
1.2	Effect of varying w_0 and w_1 in the linear model defined by Equation 1.1.	4
1.3	Example loss function of one parameter (w).	5
1.4	Data and function for the worked example of Section 1.1.5.	10
1.5	The least squares fit ($f(x; w_0, w_1) = 36.416 - 0.013x$) to the men's 100 m Olympics dataset.	12
1.6	Zoomed-in plot of the winning time in the Olympics men's 100 m sprint from 1980 showing predictions for both the 2012 and 2016 Olympics.	13
1.7	Women's Olympics 100 m data with a linear model that minimises the squared loss.	14
1.8	Male and female functions extrapolated into the future.	14
1.9	Example of linear and quadratic models fitted to a dataset generated from a quadratic function.	26
1.10	8th order polynomial fitted to the Olympics 100 m men's sprint data.	27
1.11	Least squares fit of $f(x; \mathbf{w}) = w_0 + w_1x + w_2 \sin(\frac{x-a}{b})$ to the 100 m sprint data ($a = 2660, b = 4.3$).	28
1.12	Training and validation loss for Olympics men's 100 m data.	29
1.13	Generalisation ability of 1st, 4th and 8th order polynomials on Olympics men's 100 m data.	30
1.14	Cross-validation.	30
1.15	Mean LOOCV loss as polynomials of increasing order are fitted to the Olympics men's 100 m data.	31
1.16	The training, testing and leave-one-out loss curves obtained for a noisy cubic function where a sample size of 50 is available for training and LOOCV estimation.	32
1.17	Effect of varying the regularisation parameter λ for a 5th order polynomial function.	34
2.1	Linear fit to the Olympics men's 100 m data with errors highlighted.	40
2.2	Dataset generated from a linear model.	41

2.3 An example of the probability distribution function for a binomial random variable when $N = 50$ and $q = 0.7$	54
2.4 An example of the uniform pdf.	59
2.5 Effect of increasing the number of samples on the approximation to the expectation given in Equation 2.25 where $p(y) = \mathcal{U}(0, 1)$	60
2.6 Examples of beta pdfs with three different pairs of parameters.	61
2.7 Three Gaussian pdfs with different means and variances. . . .	61
2.8 Example surface (left) and contour (right) plots for two different two-dimensional Gaussian pdfs.	63
2.9 Dataset generated from a linear model with Gaussian errors.	66
2.10 Likelihood function for the year 1980.	68
2.11 Model complexity example with Olympics men's 100 m data.	74
2.12 Data generated from the model given in Equation 2.39 and the true function.	76
2.13 Variability in $\hat{\mathbf{w}}$ for 10,000 datasets generated from the model described in Equation 2.39.	77
2.14 Functions inferred from 10 datasets generated from the model given in Equation 2.39 as well as the true function.	77
2.15 Two example datasets with different noise levels and the corresponding likelihood function.	81
2.16 Ten samples of \mathbf{w} using the distribution given in Equation 2.48.	83
2.17 (a) Example data set. (b), (c) and (d) Predictive error bars for a linear, cubic and 6th order model, respectively.	85
2.18 Examples of functions with parameters drawn from a Gaussian with mean $\hat{\mathbf{w}}$ and covariance $\text{cov}\{\hat{\mathbf{w}}\}$ for the example data set shown in Figure 2.17(a).	86
2.19 Evolution of the theoretical and empirical estimates of $\mathbf{E}_{p(\mathbf{t} \mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \widehat{\sigma^2} \right\}$ as the number of data points increases.	89
3.1 The binomial density function when $N = 10$ and $r = 0.5$	96
3.2 The binomial density function when $N = 10$ and $r = 0.9$	97
3.3 Examples of the likelihood $p(y_N r)$ as a function of r for two scenarios.	99
3.4 Examples of prior densities, $p(r)$, for r for three different scenarios.	100
3.5 Examples of three possible posterior distributions $p(r y_N)$	102
3.6 Evolution of $p(r y_N)$ as the number of observed coin tosses increases.	106
3.7 Evolution of expected value (a) and variance (b) of r as coin toss data is added to the posterior.	108
3.8 The posterior after six and seven tosses.	109
3.9 Posterior distribution after observing 10 tosses and 20 tosses	111

3.10 Evolution of the posterior $p(r y_N)$ as more coin tosses are observed for the fair coin scenario.	112
3.11 Evolution of $\mathbf{E}_{p(r y_N)}\{R\}$ (a) and $\text{var}\{R\}$ (b) as the 20 coin tosses are observed for the fair coin scenario.	113
3.12 Evolution of the posterior $p(r y_N)$ as more coin tosses are observed for the biased coin scenario.	115
3.13 Evolution of $\mathbf{E}_{p(r y_N)}\{R\}$ (a) and $\text{var}\{R\}$ (b) as the 20 coin tosses are observed for the biased coin scenario.	116
3.14 The posterior densities for the three scenarios after 100 coin tosses and 1000 coin tosses.	117
3.15 Marginal likelihood contours for the coin example.	119
3.16 Graphical model examples.	121
3.17 Graphical model for the Bayesian model of the Olympics men's 100 m data.	123
3.18 Olympics data with rescaled x values.	126
3.19 Gaussian prior used for the Olympics 100 m data (a) and some functions created with samples drawn from the prior (b).	127
3.20 Evolution of the posterior density and example functions drawn from the posterior for the Olympics data after increasing numbers of observations have been added.	128
3.21 Posterior density (a) and sampled functions (b) for the Olympics data when all 27 data points have been added.	129
3.22 Posterior density (a) and sampled functions (b) for the Olympics data when all 27 data points have been added with more realistic noise variance, $\sigma^2 = 0.05$	130
3.23 Predictive distribution for the winning time in the men's 100 m sprint at the 2012 London Olympics.	131
3.24 Dataset sampled from the function $t = 5x^3 - x^2 + x$ (a) and marginal likelihoods for polynomials of increasing order (b).	132
3.25 Marginal likelihoods for the 3rd order polynomial example with $\Sigma_0 = \sigma_0^2 \mathbf{I}$ as σ_0^2 is decreased.	133
4.1 An example of a dataset with a binary response.	140
4.2 The sigmoid function that squashes a real value to always be between 0 and 1.	142
4.3 Evolution of the components of \mathbf{w} throughout the Newton-Raphson procedure to find the \mathbf{w} corresponding to the maximum of the posterior density.	147
4.4 Inferred function in the binary response example.	148
4.5 Examples of the Laplace approximation to the gamma density function given in Equation 4.14.	152
4.6 The Laplace approximation for the binary problem.	152
4.7 Decision boundaries sampled from the Laplace approximation and the predictive probability contours.	153
4.8 A dartboard.	155

4.9	Two examples of random walks where the distribution over the next location is a Gaussian centred at the current location.	158
4.10	The Metropolis–Hastings algorithm.	159
4.11	Example of the Metropolis–Hastings algorithm in operation.	160
4.12	Results of applying the MH sampling algorithm to the binary response model.	162
4.13	Two densities that would be tricky to sample from with MH.	164
5.1	Three class classification dataset.	172
5.2	Three class classification dataset with the density contours for the three class-conditional distributions fitted using Equations 5.4 and 5.5.	173
5.3	Contour plots of the classification probabilities for the Bayesian classifier with Gaussian class-conditional distributions.	174
5.4	Density contours for Gaussian class-conditionals with the naive Bayes assumption.	176
5.5	Contour plots of the classification probabilities for the Bayesian classifier with Gaussian class-conditional distributions and the naive Bayes assumption.	176
5.6	Graphical representation of the predictive probabilities for the Bayesian classifier on the 20 newsgroups data.	179
5.7	Binary data and classification probability contours for the logistic regression model described by Equation 5.10.	181
5.8	Cartoon depicting the operation of KNN ($K = 3$).	183
5.9	Binary classification dataset and decision boundaries for $K = 1$ and $K = 5$	184
5.10	Second binary classification dataset and decision boundaries for $K = 5$ and $K = 39$	185
5.11	Using cross-validation to find the best value of K	186
5.12	The classification margin γ , defined as the perpendicular distance from the decision boundary to the closest points on either side.	187
5.13	Illustrating the steps taken to compute the margin.	188
5.14	Decision boundary and support vectors for a linear SVM. . . .	191
5.15	Decision boundary and support vectors for a linear SVM. . . .	192
5.16	Decision boundary and support vectors for a linear SVM with a soft margin for two values of the margin parameter C	194
5.17	A binary dataset for which a linear decision boundary would not be appropriate.	194
5.18	Decision boundary and support vectors for the dataset in Figure 5.17 using a Gaussian kernel with the kernel parameter $\gamma = 1$ and $C = 10$	196
5.19	Decision boundary and support vectors for the dataset in Figure 5.17 using a Gaussian kernel with different values of the kernel parameter γ and $C = 10$	197

5.20 ROC curves for the SVMs shown in Figures 5.19(a) and 5.19(b).	200
5.21 ROC curve for the SVM shown in Figure 5.18.	201
6.1 Synthetic dataset for clustering examples.	209
6.2 Illustration of the K -means algorithm.	211
6.3 $\log D$ as K increases for the data shown in Figure 6.1.	212
6.4 Two datasets in which K -means fails to capture the clear cluster structure.	213
6.5 Result of applying kernelised K -means to the data shown in Figure 6.4(a).	215
6.6 Synthetic dataset for clustering examples.	216
6.7 Generating data from two Gaussians.	218
6.8 The synthetic clustering data encountered earlier in the chapter.	225
6.9 Example of the Gaussian mixture algorithm in action.	227
6.10 The data on which K -means failed and the successful mixture model solution.	228
6.11 The log likelihood L increases with the number of components, K	229
6.12 Result of 10-fold cross-validation for a Gaussian mixture model on the data shown in Figure 6.8.	229
6.13 An example binary dataset with $N = 100$ objects and $D = 10$ dimensions.	231
6.14 $K = 5$ clusters extracted from the data shown in Figure 6.13 using the mixture model with binary components.	232
7.1 The idea of projection.	240
7.2 Examples showing the variance of different projections of two synthetic two-dimensional datasets.	241
7.3 Synthetic PCA example where one projected dimension is all that is required.	245
7.4 Synthetic PCA example where two projected dimensions are required.	246
7.5 Graphical representation of the probabilistic PCA model.	253
7.6 Synthetic probabilistic PCA example.	260
7.7 Variational Bayesian PPCA model with missing values.	262
7.8 Graphical representation of probit PCA model.	266
7.9 Standard principal components visualisation of the 2005 MP voting data.	269
7.10 Number of votes cast versus distance from the origin (of the PCA plot).	269
7.11 Probabilistic binary principal components visualisation of the 2005 MP voting data.	270

7.12 Number of votes cast versus distance from the origin for the probabilistic binary PCA.	270
7.13 Visualisation of the small parties using the two PCA methods.	271
7.14 Covariance matrix visualisation for the 20 MPs corresponding to the highest uncertainty.	271
7.15 Example of four of the votes – each MP is displayed as a circle or square depending on how they voted.	272

Preface

Machine learning is rapidly becoming one of the most important areas of general practice, research and development activity within computing science. This is reflected in the scale of the academic research area devoted to the subject and the active recruitment of machine learning specialists by major international banks and financial institutions as well as companies such as Microsoft[®], Google[®], Yahoo[®] and Amazon[®].

This growth can be partly explained by the increase in the quantity and diversity of measurements we are able to make of the world. A particularly fascinating example arises from the wave of new biological measurement technologies that preceded the sequencing of the first genomes. It is now possible to measure the detailed molecular state of an organism in ways that would have been hard to imagine only a short time ago. Such measurements go far beyond our understanding of these organisms and machine learning techniques have been heavily involved in the distillation of useful structures from them.

This book is based on material presented in a machine learning course in the School of Computing Science at the University of Glasgow, UK. The course, presented to final year undergraduates and taught by postgraduates, is made up of 20 hour-long lectures and 10 hour-long laboratory sessions. In such a short teaching period, it is impossible to cover more than a small fraction of the material that now comes under the banner of machine learning. Our intention when teaching this course, therefore, is to present the core mathematical and statistical techniques required to understand some of the most popular machine learning algorithms and then present a few of these algorithms that span the main problem areas within machine learning: classification, clustering and projection. At the end of the course, the students should have the knowledge and confidence to be able to explore machine learning literature to find methods that are more appropriate for them. The same is hopefully true of readers of this book.

Due to the varying mathematical literacy of students taking the course, we assume only very minor mathematical pre-requisites. An undergraduate student from computer science, engineering, physics (or any other numerical subject) should have no problem. This does not preclude those without such experience – additional mathematical explanations appear throughout the text in comment boxes. In addition, important equations have been highlighted – it is worth spending time understanding these equations before proceeding.

Students attending this course often find the practical sessions very useful. Experimenting with the various algorithms and concepts helps transfer them from an abstract set of equations into something that could be used to solve real problems. We have attempted to transfer this to the book through an extensive collection of MATLAB®/Octave¹ scripts, available from the associated web page and referenced throughout the text. These scripts enable the user to recreate plots that appear in the book and investigate changing model specifications and parameter values.

Finally, the machine learning methods that are covered in this book are our choice of those we feel students should understand. In limited space and time, we think that it is more worthwhile to give detailed descriptions and derivations for a small number of algorithms than attempt to cover many algorithms at a lower level of detail – many people will not find their favourite algorithms within this book!

MATLAB® is a registered trademark of The MathWorks, Inc.
For product information, please contact:

The MathWorks, Inc.
3 Apple Hill Drive
Natick MA 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7001
E-mail: info@mathworks.com
Web: www.mathworks.com

Simon Rogers and Mark Girolami

¹A free mathematical software environment, available from www.gnu.org/software/octave/

Linear Modelling: A Least Squares Approach

An important and general problem in Machine Learning, which has wide application, is *learning* or *inferring* a functional relationship between a set of *attribute* variables and associated *response* or *target* variables so that we can predict the response for any set of attributes. For example, we may wish to build a model that can perform disease diagnosis. To do this we would use a dataset comprised of measurements (attributes, e.g. blood pressure, heart rate, weight etc.) taken from patients with known disease states (responses, healthy or diseased). In a completely different example, we may wish to make recommendations to customers. In this case, we could build a **model** from descriptors of items a particular customer had previously bought (attributes) and whether or not the customer ultimately liked the product (response). This would enable us to predict which objects a customer would like and hence make recommendations. There are many more important application areas that we will come across throughout this text.

1.1 Linear modelling

To begin with we will consider, using a practical example, the most straightforward of *learning* problems, linear modelling¹ – learning a **linear** relationship between attributes and responses. Figure 1.1 shows the gold medal winning time for the men’s 100 m at each of the Olympics Games held since 1896. Our aim is to use this data to *learn* a model of the functional dependence (if one exists) between Olympics year and 100 m winning time and use this model to make predictions about the winning times in future games. Clearly the year is not the only factor that affects the winning time and if we are interested in using our predictions seriously we may want to take other things

¹The type of modelling we will consider here is often known as *regression* and was originally used in the context of genetics by Francis Galton (1877) when studying how intelligence is passed on (or not, as the case may be) from generation to generation. The term was then adopted by statisticians who developed Galton’s work within a statistical context.

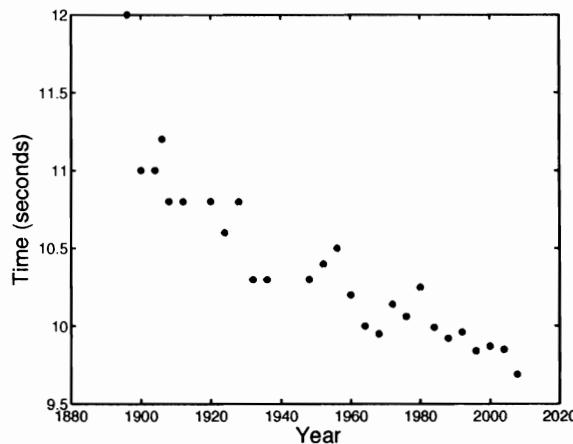


FIGURE 1.1: Winning men's 100 m times at the Summer Olympics since 1896. Note that the two world wars interrupted the games in 1914, 1940 and 1944.

into account (the recent form of the main competitors is an obvious example). However, examining Figure 1.1 we can see that there is at least a statistical dependence between year and winning time (it may not be a *causal* dependence – elapsing years are not directly *causing* the drop in winning times) and this is enough to help us introduce and develop the main ideas of linear modelling.

1.1.1 Defining the model

We can begin by defining our model as a function which maps our *input* attributes, in this case Olympics year, to our *output* or target values – winning time. For our attributes, we will use the numerical value of the year – e.g. 1980 – although there are alternative formulations (e.g. years since the first modern games, $1980 - 1896 = 84$) that would make no real difference to the underlying assumptions.

There are many **functions** that could be used to define this mapping. In general, this function will take as an input x (the Olympics year) and will return t (the winning time in seconds). In other words, t is a function of x . Mathematically, we will write this as $t = f(x)$. In some cases, all we will need to know to evaluate our function is x . For example, if $f(x) = \sin(x)$ or, say, $f(x) = x$, we can compute t for any x . In general, we will need to be more flexible and it is likely that our model will have a set of associated **parameters**. For example, $t = ax$ has a parameter called a that needs to be defined somehow. Learning model parameters from a suitable dataset is a common theme in Machine Learning. We will use $t = f(x; a)$ to denote a function $f(\cdot)$ that acts on x and has a parameter a .

Comment 1.1 – Linear relationships: The equation:

$$y = mx + c,$$

where m and c are constant, defines a linear relationship between x and y . It is called linear because the relationship between x and y could be visualised as a straight line. The following equations are nonlinear due to the more complex forms in which we find the variables x and y :

$$y = mx^2 + c, \quad y = \sin(x), \quad \sqrt{y} = mx + c.$$

The values of m and c do not affect the linearity of the relationship. For example, the following still represent linear relationships between x and y :

$$y = mx + c^2, \quad y = x \sin(m) + c.$$

1.1.2 Modelling assumptions

To help us choose which particular model to use, we need to make some assumptions. Our principal assumption at this stage is the following:

The relationship between x and t is linear (see Comment 1.1).

This could be stated alternatively as:

The data in Figure 1.1 could be adequately modelled with a straight line.

Or:

The winning time drops by the same amount every M years.

Examining Figure 1.1 we can see that this assumption is not perfectly satisfied. However, it is our hope that it is adequate and will produce a model that is useful in the sense that it can make predictions regarding winning times in the future.

The simplest model that satisfies our assumptions is

$$t = f(x) = x,$$

the winning time is equal to the Olympics year. The fact that x takes values greater than or equal to 1880 and t values less than or equal to 12, and that the winning time is decreasing as the year increases tells us that this model is inadequate. Adding a single parameter results in:

$$t = f(x; w) = wx,$$

where w can be either positive or negative. This enhanced model lets us produce a straight line with any gradient through the choice of w . This is an increase in flexibility but it is still limited by the fact that at year 0, the model predicts a winning time of $w \times 0 = 0$. Looking at the data we can see that this is not realistic – following the general trend of the data, the winning time at year 0 is actually going to be quite a large number. Adding one more parameter to the model overcomes this limitation:

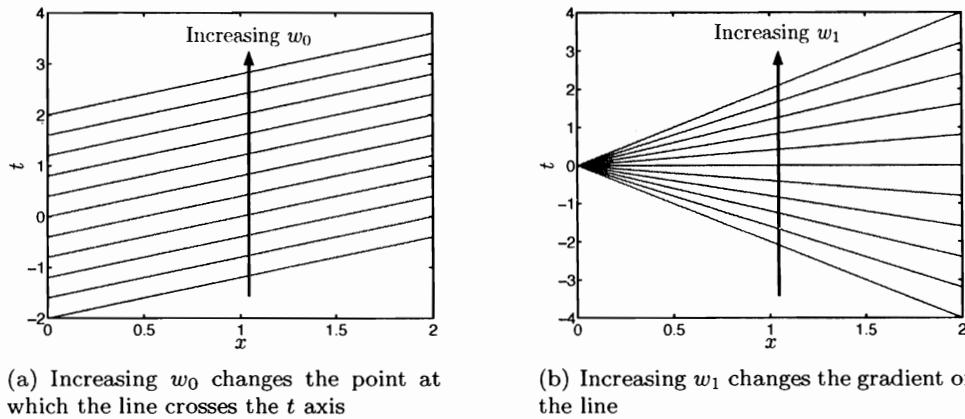


FIGURE 1.2: Effect of varying w_0 and w_1 in the linear model defined by Equation 1.1.

$$t = f(x; w_0, w_1) = w_0 + w_1 x. \quad (1.1)$$

This is the standard equation for a straight line that many readers will have encountered before. The *learning* task now involves using the data in Figure 1.1 to choose suitable values for the two parameters w_0 and w_1 . These two parameters are often known as the intercept (w_0 , where the line intercepts the t -axis) and the gradient (w_1 , the gradient of the line) and the effect of varying them can be seen in Figure 1.2 (MATLAB[®] script: `plotlinear.m`) (see Exercise EX 1.1).

1.1.3 Defining what a *good* model is

In order to choose values of w_0 and w_1 that are somehow *best*, we need to define what *best* means. Common sense would suggest that the *best* solution consists of the values of w_0 and w_1 that produce a line that passes as close as possible to *all* of the data points. A common way of measuring how close a particular model gets to one of the data points is the squared difference between the true winning time and the winning time predicted by the model. Using x_n, t_n to denote the n th Olympics year and winning time, respectively, the squared difference is defined as:

$$(t_n - f(x_n; w_0, w_1))^2.$$

The smaller this number is, the closer the model, at x_n , is to t_n . Squaring the difference is important. Without it, we could indefinitely reduce this quantity by continually increasing $f(x_n; w_0, w_1)$.

This expression is known as the *squared loss function*, as it describes how much accuracy we are losing through the use of $f(x_n; w_0, w_1)$ to model t_n . Throughout this text, we will use $\mathcal{L}_n()$ to denote loss functions. In this case,

$$\mathcal{L}_n(t_n, f(x_n; w_0, w_1)) = (t_n - f(x_n; w_0, w_1))^2 \quad (1.2)$$

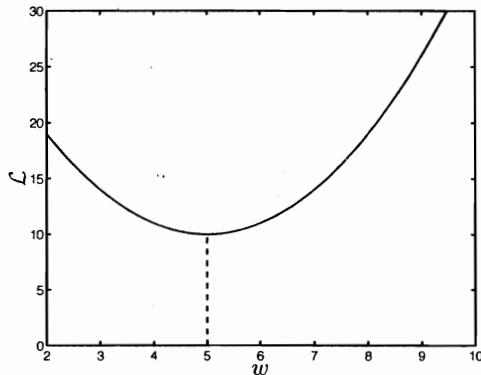


FIGURE 1.3: Example loss function of one parameter (w). The dashed line shows the value of w that minimised the loss ($w = 5$).

is the loss for year n . Loss is always positive and the lower the *loss*, the better our function describes the data. As we want a low *loss* for all of the N years, we consider the *average loss* across the whole dataset, given as

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(t_n, f(x_n; w_0, w_1)). \quad (1.3)$$

This is the average of the loss values at each of the N years. The lower it is, the better. We will therefore tune w_0 and w_1 to produce the model that results in the lowest value of the average loss, \mathcal{L} . Finding these *best* values for w_0 and w_1 can be expressed mathematically as

$$\operatorname{argmin}_{w_0, w_1} \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(t_n, f(x_n; w_0, w_1)).$$

The term argmin is the mathematical shorthand for ‘find the argument that minimises...’ In this instance, the argument(s) are the values of w_0 and w_1 and the expression to be minimised is the average loss. Figure 1.3 shows a hypothetical loss that is a function of a single parameter, w . The value of w that minimises \mathcal{L} is $w = 5$. Historically, minimisation of the squared loss is the basis of the *Least Squares* errors method of function approximation which dates back to methods developed by Gauss and Legendre (1809) when predicting planetary motion.

Other loss functions exist that are suitable for regression. For example, a common alternative is the absolute loss:

$$\mathcal{L}_n = |t_n - f(x_n; w_0, w_1)|.$$

The squared loss is a very common choice, in part due to the fact that it makes finding the best values of w_0 and w_1 relatively straightforward – we can derive an **analytical** solution. However, modern computational power has reduced the importance of mathematical convenience – there is no longer any excuse for choosing a convenient loss function over one more suited to the data. This notwithstanding, our aim is to

introduce general modelling concepts for which the squared loss will be adequate. It is worth bearing in mind that others are available and, in many cases, will be more appropriate.

1.1.4 The least squares solution – a worked example

To recap, our dataset consists of $n = 1, \dots, N$ observations, each of which consists of a year x_n and a time in seconds t_n .

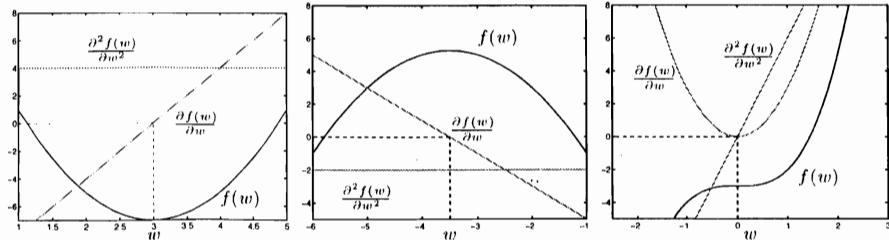
We are going to attempt to find a functional relationship using a linear model defined as

$$f(x; w_0, w_1) = w_0 + w_1 x \quad (1.4)$$

and we have decided that we will use the least squares loss function to choose suitable values of w_0 and w_1 . Substituting the linear model into the expression for average loss and multiplying out the brackets results in

$$\begin{aligned} \mathcal{L} &= \frac{1}{N} \sum_{n=1}^N \mathcal{L}_n(t_n, f(x_n; w_0, w_1)) \\ &= \frac{1}{N} \sum_{n=1}^N (t_n - f(x_n; w_0, w_1))^2 \\ &= \frac{1}{N} \sum_{n=1}^N (t_n - (w_0 + w_1 x_n))^2 \\ &= \frac{1}{N} \sum_{n=1}^N (w_1^2 x_n^2 + 2w_1 x_n w_0 - 2w_1 x_n t_n + w_0^2 - 2w_0 t_n + t_n^2) \\ &= \frac{1}{N} \sum_{n=1}^N (w_1^2 x_n^2 + 2w_1 x_n (w_0 - t_n) + w_0^2 - 2w_0 t_n + t_n^2) \end{aligned} \quad (1.5)$$

Comment 1.2 – Turning points: We can find turning points (that might correspond to minima) of a function, $f(w)$, by searching for points where the gradient of the function, $\frac{\delta f(w)}{\delta w}$, is zero. To determine whether or not a turning point corresponds to a maximum, minimum or saddle point, we can examine the second derivative, $-\frac{\delta^2 f(w)}{\delta w^2}$. If, at a turning point \hat{w} , the second derivative is positive, we know that this turning point is a minimum. The following three plots show three example functions along with their first and second derivatives:



In general, a function may have several turning points. An interesting special case is functions whose second derivative is a positive constant – these correspond to functions that have only one minimum.

Differentiating the loss function: At a minimum of \mathcal{L} , the **partial derivatives** with respect to w_1 and w_0 must be zero (see Comment 1.2). Therefore, computing the partial derivatives, equating them to zero and solving for w_0 and w_1 will give us a potential minimum. Starting with w_1 , we know that terms in Equation 1.5 that do not include w_1 can be ignored (as their partial derivative with respect to w_1 will be zero). Removing these terms leaves

$$\frac{1}{N} \sum_{n=1}^N [w_1^2 x_n^2 + 2w_1 x_n w_0 - 2w_1 x_n t_n].$$

Before we take the partial derivatives, we will re-arrange the expression to make it a little simpler. In particular, taking terms that are not indexed by n outside of the sum and re-arranging results in

$$w_1^2 \frac{1}{N} \left(\sum_{n=1}^N x_n^2 \right) + 2w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n (w_0 - t_n) \right).$$

Taking the partial derivative with respect to w_1 gives us the following expression:

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n^2 \right) + \frac{2}{N} \left(\sum_{n=1}^N x_n (w_0 - t_n) \right). \quad (1.6)$$

Now we do the same for w_0 . Removing non w_0 terms leaves:

$$\frac{1}{N} \sum_{n=1}^N [w_0^2 + 2w_1 x_n w_0 - 2w_0 t_n].$$

Again, we will re-arrange it a bit before we differentiate. Moving terms not indexed by n outside of the summation (noting that $\sum_{n=1}^N w_0^2 = N w_0^2$) results in

$$w_0^2 + 2w_0 w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n \right) - 2w_0 \frac{1}{N} \left(\sum_{n=1}^N t_n \right).$$

Taking the partial derivative with respect to w_0 results in:

$$\frac{\partial \mathcal{L}}{\partial w_0} = 2w_0 + 2w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n \right) - \frac{2}{N} \left(\sum_{n=1}^N t_n \right). \quad (1.7)$$

Equating the derivatives to zero: We now have expressions for the partial derivatives of the loss with respect to both w_0 and w_1 . To find the values of w_0 and w_1 that correspond to a turning point (hopefully a minimum), we must set these expressions to zero and solve for w_0 and w_1 . It's easiest to start with the expression for w_0 . Setting Equation 1.7 to zero and solving for w_0 :

$$\begin{aligned} 2w_0 + 2w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n \right) - \frac{2}{N} \left(\sum_{n=1}^N t_n \right) &= 0 \\ 2w_0 &= \frac{2}{N} \left(\sum_{n=1}^N t_n \right) - w_1 \frac{2}{N} \left(\sum_{n=1}^N x_n \right) \\ w_0 &= \frac{1}{N} \left(\sum_{n=1}^N t_n \right) - w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n \right). \end{aligned}$$

Denoting the average winning time as $\bar{t} = \frac{1}{N} \sum_{n=1}^N t_n$ and the average Olympics year as $\bar{x} = \frac{1}{N} \sum_{n=1}^N x_n$ we can rewrite our expression for the value of w_0 at the turning point ($\widehat{w_0}$) as

$$\widehat{w_0} = \bar{t} - w_1 \bar{x}. \quad (1.8)$$

What insight can we gain from this expression? This new expression is a rearrangement of our original model ($t_n = w_0 + w_1 x_n$) where t_n and x_n have been replaced by their average values \bar{t} and \bar{x} . Consider the value of our function averaged over the N data points. This is given by:

$$\frac{1}{N} \sum_{n=1}^N f(x_n; w_0, w_1) = \frac{1}{N} \sum_{n=1}^N (w_0 + w_1 x_n) = w_0 + w_1 \bar{x}.$$

The average winning time is given by \bar{t} so, in using Equation 1.8, we are choosing $\widehat{w_0}$ to ensure that the average value of the function is equal to the average winning time. Intuitively, matching the averages in this way seems very sensible.

Before we use Equation 1.6 to get an expression for $\widehat{w_1}$ (the value of w_1 at the turning point – see Comment 1.2), it is worth briefly examining the second derivatives to ensure that this is a minimum. Differentiating Equation 1.6 again with respect to w_1 and Equation 1.7 again with respect to w_0 results in:

$$\begin{aligned} \frac{\delta^2 \mathcal{L}}{\delta w_1^2} &= \frac{2}{N} \sum_{n=1}^N x_n^2 \\ \frac{\delta^2 \mathcal{L}}{\delta w_0^2} &= 2. \end{aligned} \quad (1.9)$$

Both of these quantities must be positive. This tells us that there will be only one turning point and it will correspond to a minimum of the loss.

This process has supplied us with an expression for the value of $\widehat{w_0}$ – the value of w_0 that minimises the loss. This expression depends on w_1 implying that for any particular w_1 , we know the best w_0 . Substituting our expression for the best w_0 value (Equation 1.8) into Equation 1.6 and rearranging, we obtain an expression that only includes w_1 terms:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_1} &= w_1 \frac{2}{N} \left(\sum_{n=1}^N x_n^2 \right) + \frac{2}{N} \left(\sum_{n=1}^N x_n (\widehat{w_0} - t_n) \right) \\ &= w_1 \frac{2}{N} \left(\sum_{n=1}^N x_n^2 \right) + \frac{2}{N} \left(\sum_{n=1}^N x_n (\bar{t} - w_1 \bar{x} - t_n) \right) \\ &= w_1 \frac{2}{N} \left(\sum_{n=1}^N x_n^2 \right) + \bar{t} \frac{2}{N} \left(\sum_{n=1}^N x_n \right) - w_1 \bar{x} \frac{2}{N} \left(\sum_{n=1}^N x_n \right) - \frac{2}{N} \left(\sum_{n=1}^N x_n t_n \right). \end{aligned}$$

We can simplify this expression by using $\bar{x} = (1/N) \sum_{n=1}^N x_n$ as before and gathering together w_1 terms:

$$\frac{\partial \mathcal{L}}{\partial w_1} = 2w_1 \left[\left(\frac{1}{N} \sum_{n=1}^N x_n^2 \right) - \bar{x} \bar{x} \right] + 2\bar{t} \bar{x} - 2 \frac{1}{N} \left(\sum_{n=1}^N x_n t_n \right).$$

Finally, we can get an expression for \widehat{x}_1 by setting this partial derivative to zero and solving for w_1 :

$$\begin{aligned} 2w_1 \left[\left(\frac{1}{N} \sum_{n=1}^N x_n^2 \right) - \bar{x}\bar{x} \right] + 2\bar{t}\bar{x} - 2\frac{1}{N} \left(\sum_{n=1}^N x_n t_n \right) &= 0 \\ 2w_1 \left[\left(\frac{1}{N} \sum_{n=1}^N x_n^2 \right) - \bar{x}\bar{x} \right] &= 2\frac{1}{N} \left(\sum_{n=1}^N x_n t_n \right) - 2\bar{t}\bar{x} \\ \widehat{w}_1 &= \frac{\frac{1}{N} \left(\sum_{n=1}^N x_n t_n \right) - \bar{t}\bar{x}}{\left(\frac{1}{N} \sum_{n=1}^N x_n^2 \right) - \bar{x}\bar{x}}. \end{aligned}$$

It is helpful to now define some new average quantities. The first, $(1/N) \sum_{n=1}^N x_n^2$ is the average squared value of the data and we will denote this \bar{x}^2 . Note that this quantity is not the same as $(\bar{x})^2$. The second is $(1/N) \sum_{n=1}^N x_n t_n$ (which, similarly is not the same as $\bar{x}\bar{t}$). We will denote this as $\bar{x}\bar{t}$. Substituting these into our expression for w_1 gives:

$$\widehat{w}_1 = \frac{\bar{x}\bar{t} - \bar{x}\bar{t}}{\bar{x}^2 - (\bar{x})^2} \quad (1.10)$$

Equations 1.10 and 1.8 provide everything required to compute the best parameter values. First \widehat{w}_1 Equation 1.10 is substituted into Equation 1.8 to calculate \widehat{w}_0 (MATLAB script: `fitlinear.m`).

1.1.5 Worked example

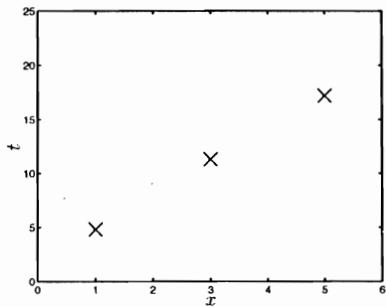
Before we fit the linear model to the Olympics data, it is useful to provide a worked example on a smaller dataset. Assume we observe $N = 3$ data points, provided in Table 1.1. The final row also gives the various averages required to compute \widehat{w}_0 and \widehat{w}_1 : \bar{x} , \bar{t} , $\bar{x}\bar{t}$ and \bar{x}^2 . The three data points are plotted in Figure 1.4.

Substituting these values into Equation 1.10 gives:

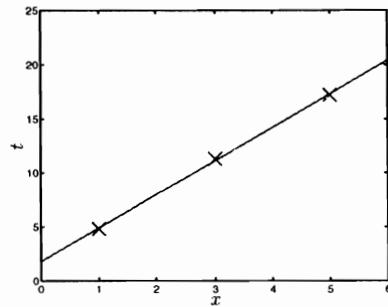
$$\begin{aligned} w_1 &= \frac{41.57 - 3 \times 11.1}{11.67 - 3 \times 3} \\ &= \frac{8.27}{2.67} \\ &= 3.1 \end{aligned}$$

TABLE 1.1: Synthetic dataset for linear regression example.

n	x_n	t_n	$x_n t_n$	x_n^2
1	1	4.8	4.8	1
2	3	11.3	33.9	9
3	5	17.2	86	25
$(1/N) \sum_{n=1}^N$	3	11.1	41.57	11.67



(a) The three synthetic data points described in Table 1.1



(b) The least squares fit defined by $f(x; w_0, w_1) = 1.8 + 3.1x$

FIGURE 1.4: Data and function for the worked example of Section 1.1.5.

and:

$$w_0 = 11.1 - 3.1 \times 3 = 1.8.$$

Our best linear function is therefore:

$$f(x; w_0, w_1) = 1.8 + 3.1x,$$

and it is shown in Figure 1.4(b).

1.1.6 Least squares fit to the Olympics data

The data for the Olympics 100 m dataset (shown in Figure 1.1) is summarised in Table 1.2.

Applying exactly the same methodology to this data, we obtain the following values for w_1 and w_0 (note that our final values were worked out in MATLAB – if you work through, you might get slightly different results due to rounding errors):

$$\begin{aligned} w_1 &= \frac{20268.1 - 1952.37 \times 10.39}{3.8130 \times 10^6 - 1952.37 \times 1952.37} \\ &= \frac{-16.3}{1225.5} \\ &= -0.0133 \\ w_0 &= 10.39 - (-0.0133) \times 1952.37 \\ &= 36.416. \end{aligned}$$

Therefore, our best linear function is:

$$f(x; w_0, w_1) = 36.416 - 0.013x. \quad (1.11)$$

The function is plotted in Figure 1.5 (see Exercise EX 1.2). Do these values agree with the approximations you made in Exercise EX 1.1? (MATLAB script: `fitolympic.m`)

TABLE 1.2: Olympics men's 100 m data.

n	x_n	t_n	$x_n t_n$	x_n^2
1	1896	12.00	22752.0	3.5948×10^6
2	1900	11.00	20900.0	3.6100×10^6
3	1904	11.00	20944.0	3.6252×10^6
4	1906	11.20	21347.2	3.6328×10^6
5	1908	10.80	20606.4	3.6405×10^6
6	1912	10.80	20649.6	3.6557×10^6
7	1920	10.80	20736.0	3.6864×10^6
8	1924	10.60	20394.4	3.7018×10^6
9	1928	10.80	20822.4	3.7172×10^6
10	1932	10.30	19899.6	3.7326×10^6
11	1936	10.30	19940.8	3.7481×10^6
12	1948	10.30	20064.4	3.7947×10^6
13	1952	10.40	20300.8	3.8103×10^6
14	1956	10.50	20538.0	3.8259×10^6
15	1960	10.20	19992.0	3.8416×10^6
16	1964	10.00	19640.0	3.8573×10^6
17	1968	9.95	19581.6	3.8730×10^6
18	1972	10.14	19996.1	3.8888×10^6
19	1976	10.06	19878.6	3.9046×10^6
20	1980	10.25	20295.0	3.9204×10^6
21	1984	9.99	19820.2	3.9363×10^6
22	1988	9.92	19721.0	3.9521×10^6
23	1992	9.96	19840.3	3.9681×10^6
24	1996	9.84	19640.6	3.9840×10^6
25	2000	9.87	19740.0	4.0000×10^6
26	2004	9.85	19739.4	4.0160×10^6
27	2008	9.69	19457.5	4.0321×10^6
$(1/N) \sum_{n=1}^N$		1952.37	10.39	3.8130×10^6

1.1.7 Summary

It is worth recapping the topics covered so far. We have introduced the idea of creating a model (in particular a linear one) that encapsulates the relationship between a set of attributes and a set of responses. To enable us to fit (or learn) this model from data, we defined a *loss* function as a way of objectively identifying how good a particular model was. Using the squared loss, we derived exact expressions for the values of the model parameters that minimised the loss and therefore corresponded to the best function. Finally, we applied this technique to two different data sets. We shall now see how we can use the model to make predictions.

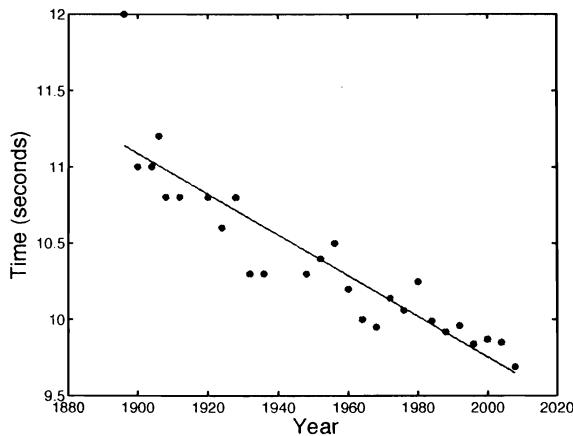


FIGURE 1.5: The least squares fit ($f(x; w_0, w_1) = 36.416 - 0.013x$) to the men's 100 m Olympics dataset.

1.2 Making predictions

Now that we have a model relating the Olympics year to the winning 100 m sprint time, we can use it to predict the winning time for a year that we have not yet observed. For example, to predict the winning times at the 2012 and 2016 Olympics, t^{2012} and t^{2016} , we plug $x = 2012$ and $x = 2016$ into our formula.

$$\begin{aligned} f(x; w_0 = 36.416, w_1 = -0.0133) &= 36.416 - 0.0133x \\ t^{2012} = f(2012; w_0, w_1) &= 36.416 - 0.0133 \times 2012 = 9.595 \\ t^{2016} = f(2016; w_0, w_1) &= 36.416 - 0.0133 \times 2016 = 9.541 \end{aligned}$$

These predictions can be seen in Figure 1.6 (MATLAB script: `olympred.m`). They tell us that based on our linear regression model we might expect a winning time of 9.595 s in London in 2012. This value is very precise. It seems unlikely that *any* model would be able to predict the outcome of such a complex event to such a high degree of accuracy, least of all one based on nothing more than a straight line. Our model is not even able to predict data that it has seen very precisely, as can be seen by the distance of some points to the line in Figure 1.5. Assuming that it will become more precise into the future seems particularly foolish.

Precise predictions are only of limited use in situations where our model is not perfect (almost all situations). In general, it is more useful to be able to express a range of values rather than any particular one. We shall see how to do this in Chapter 2 and beyond.

1.2.1 A second Olympics dataset

A second dataset, related to the first, is shown in Table 1.3 and is plotted, along with the linear model that minimises the squared loss, in Figure 1.7 (see Exercise EX

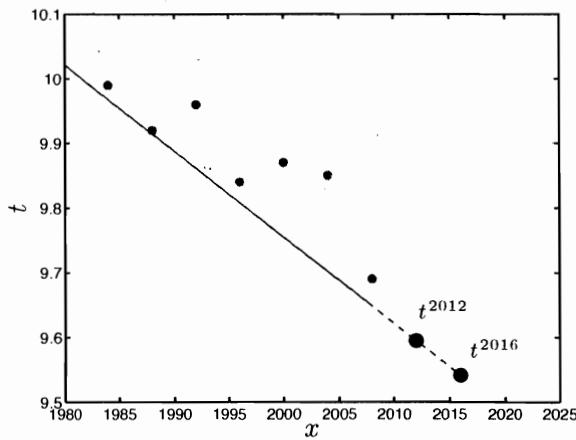


FIGURE 1.6: Zoomed-in plot of the winning time in the Olympics men's 100 m sprint from 1980 showing predictions for both the 2012 and 2016 Olympics.

TABLE 1.3: Olympics women's 100 m data.

n	x_n	t_n	$x_n t_n$	x_n^2
1	1928	12.20	23521.6	3.7172×10^6
2	1932	11.90	22990.8	3.7326×10^6
3	1936	11.50	22264.0	3.7481×10^6
4	1948	11.90	23181.2	3.7947×10^6
5	1952	11.50	22448.0	3.8103×10^6
6	1956	11.50	22494.0	3.8259×10^6
7	1960	11.00	21560.0	3.8416×10^6
8	1964	11.40	22389.6	3.8573×10^6
9	1968	11.00	21648.0	3.8730×10^6
10	1972	11.07	21830.0	3.8888×10^6
11	1976	11.08	21894.1	3.9046×10^6
12	1980	11.06	21898.8	3.9204×10^6
13	1984	10.97	21764.5	3.9363×10^6
14	1988	10.54	20953.5	3.9521×10^6
15	1992	10.82	21553.4	3.9681×10^6
16	1996	10.94	21836.2	3.9840×10^6
17	2000	11.12	22240.0	4.0000×10^6
18	2004	10.93	21903.7	4.0160×10^6
19	2008	10.78	21646.2	4.0321×10^6
$(1/N) \sum_{n=1}^N$		1970.74	11.22	3.8844×10^6

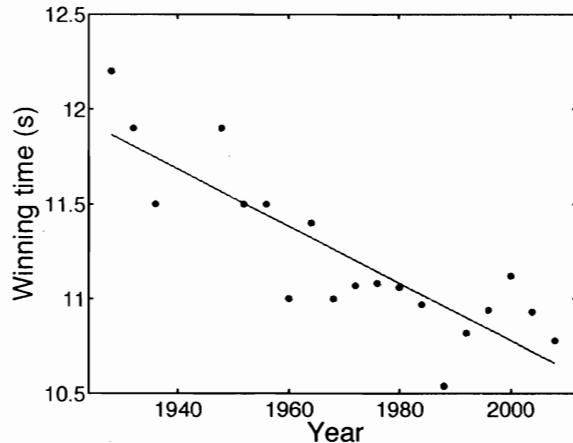


FIGURE 1.7: Women’s Olympics 100 m data with a linear model that minimises the squared loss.

1.6 and Exercise EX 1.7). The model for the women’s data is (remember that you might find slight differences to these values due to rounding errors):

$$f(x; w_0, w_1) = 40.92 - 0.015x.$$

It is interesting to compare this with the model obtained for the men’s data:

$$f(x; w_0, w_1) = 36.416 - 0.013x.$$

The women’s model has a higher intercept (w_0) and a steeper negative gradient (w_1). If we plot the two models together, as seen in Figure 1.8, we see that the higher intercept and larger negative gradient mean that at some point, the two lines

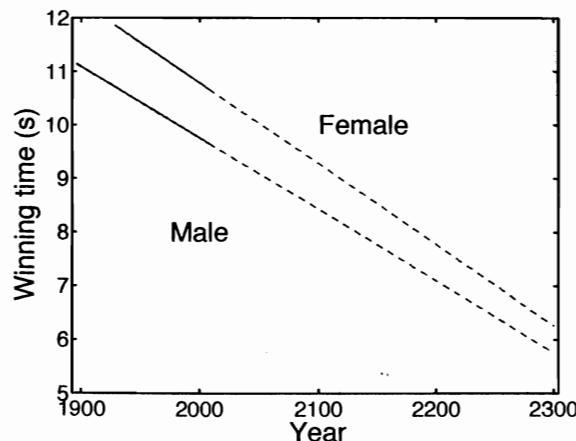


FIGURE 1.8: Male and female functions extrapolated into the future.

will intercept. Using our models we can predict the first Olympic games when the women's winning time will be faster than the men's. According to our models this will be in the 2592 Olympics (the actual answer has been rounded up to the nearest Olympics year and has been computed in MATLAB using the exact data so you might find slight differences due to rounding; (see Exercise EX 1.8).

As with the point predictions for individual models, we should not place too much confidence on this prediction coming about. Not only is the prediction incredibly precise, it is also a very long time from our last observed data point. Can we assume that the relationship between winning time and Olympics year will continue this far into the future? To assume that it can is also to assume that there will, eventually, be a winning time of 0 seconds and we know that this is impossible.

1.2.2 Summary

In the previous sections we have seen how we can fit a simple linear model to a small dataset and use the resulting model to make predictions. We have also described some of the limitations of making predictions in this way and we will introduce alternative techniques that overcome these limitations in later chapters. Up to this point, our attributes (x_n) have been individual numbers. We will now see how the linear model can be extended to larger sets of attributes, enabling us to model more complex relationships.

1.3 Vector/matrix notation

In many applications, we will be interested in problems where each data point is described by a set of several attributes. For example, we might decide that using only the Olympics year is unsuitable for a model of Olympics sprint data. A model that used the Olympics year and each athlete's personal best might be more accurate. Using s_1, s_2, \dots, s_8 to denote the personal best times for the athletes running in lanes 1 to 8, a possible linear model might consist of:

$$\begin{aligned} t = f(x, s_1, \dots, s_5; w_0, \dots, w_6) = & w_0 + w_1x + w_2s_1 + w_3s_2 + w_4s_3 \\ & + w_5s_4 + w_6s_5 + w_7s_6 + w_8s_7 + w_9s_8. \end{aligned}$$

We could go through the analysis of the previous sections to find $\widehat{w}_0, \dots, \widehat{w}_9$. After taking partial derivatives of the loss function, we would be left with 10 equations that would need to be re-arranged and substituted into one another. This would be a time consuming exercise and would rapidly become infeasible as the number of variables we wanted to include increased further – Machine Learning applications with thousands of variables are not uncommon. Fortunately there is an alternative – using vectors and matrices.

As this is an area that some readers will find unfamiliar, we shall now devote some time to describing vector and matrix notation and how to perform mathematical operations with quantities in vector and matrix form. Readers familiar with these concepts could jump straight to Section 1.4.

Comment 1.3 – Scalars, vectors and matrices: We will follow the standard convention of representing scalar values by letters (e.g. x), vectors by bold lowercase letters (e.g. \mathbf{x}) and matrices by bold uppercase letters (e.g. \mathbf{X}). Whilst we shall consistently stick to this notation, different communities have different ways of defining vectors. For example, \bar{x} is common for a vector x .

The 9 attributes for each data point (8 personal bests and Olympics year) can be combined into a single variable by stacking them together to form a vector. We will denote vectors with bold lower case letters, e.g. \mathbf{x}_n (see Comment 1.3). Often we will need to refer to individual elements within a particular vector or matrix and will use *indices* to make it clear which element we're referring to. For example, the first element of the vector \mathbf{x}_n would be denoted x_{n1} , the i th by x_{ni} .

If we want to show all of the elements in a vector, we write it out in a tabular fashion, surrounded by square brackets. Here are examples of vectors of length 2 and 4:

$$\mathbf{x}_n = \begin{bmatrix} x_{n1} \\ x_{n2} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}.$$

Comment 1.4 – Vector transpose: The transpose of a vector \mathbf{x} , denoted \mathbf{x}^\top , is obtained by rotating the vector such that rather than having one column and several rows, it has one row and several columns. For example:

$$\mathbf{x} = \begin{bmatrix} 4 \\ 7 \\ 11 \\ -2 \end{bmatrix}, \quad \mathbf{x}^\top = [4, 7, 11, -2].$$

It is often a bit clumsy to keep drawing vectors as columns so we will often draw them as rows, and use the transpose operator (see Comment 1.4) to show that they should be rotated. If we assume that we have D attributes, we would define \mathbf{x}_n as $\mathbf{x}_n = [x_{n1}, \dots, x_{nD}]^\top$. In the case of our Olympics data, $\mathbf{x} = [\text{Year}, s_1, s_2, \dots, s_8]^\top$.

Comment 1.5 – Matrix/vector dimensions and indexing: If we are quoting the size (or the dimension) of a matrix or vector, we give two numbers, starting with the number of rows. For example,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}$$

has dimension 3×2 . A vector is a special case of a matrix where the second dimension is 1. For example,

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

could be thought of as a matrix with dimension 4×1 .

When indexing elements within a vector, a single number is sufficient (e.g. y_3 for the third element in \mathbf{y} above). When indexing a matrix, we will use two subscripts, starting with the row. For example, a_{21} represents the item in the second row and first column of \mathbf{A} (above). Note that sometimes we will also have a subscript denoting the object index. For example, \mathbf{x}_n is the vector holding the n th set of attributes. This index, if present, will always come first. It should be obvious from the context whether or not this index is present.

Before we embark on adding additional variables, it is worthwhile to repeat the analysis of the original model ($t = w_0 + w_1x$) in vector form. This will allow us to compare the expressions we obtain for \widehat{w}_0 and \widehat{w}_1 in both cases. The first step is to combine w_0 and w_1 into a single parameter vector \mathbf{w} and create data vectors \mathbf{x}_n by augmenting each x_n with a 1, i.e.

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad \mathbf{x}_n = \begin{bmatrix} 1 \\ x_n \end{bmatrix}$$

The model can be expressed in terms of \mathbf{x}_n and \mathbf{w} as (matrix/vector multiplication is defined in Comment 1.7):

$$f(x_n; w_0, w_1) = \mathbf{w}^\top \mathbf{x}_n = w_0 + w_1 x_n.$$

We can replace any instance of $w_0 + w_1x$ by $\mathbf{w}^\top \mathbf{x}$. For example, our squared loss \mathcal{L} can be expressed as

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N (t_n - \mathbf{w}^\top \mathbf{x}_n)^2. \quad (1.12)$$

In actual fact, we can express this average loss as the following function of various vectors and matrices which will be easier to manipulate:

$$\mathcal{L} = \frac{1}{N} (\mathbf{t} - \mathbf{X}\mathbf{w})^\top (\mathbf{t} - \mathbf{X}\mathbf{w}).$$

To see how this is equivalent to Equation 1.12, we start by combining all \mathbf{x}_n into

one matrix \mathbf{X} , and all t_n into one vector \mathbf{t} :

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}.$$

Comment 1.6 – Matrix transpose: For a matrix, \mathbf{X} , the transpose, \mathbf{X}^\top is formed by turning each row into a column and each column into a row. For example, if $\mathbf{Y} = \mathbf{X}^\top$, then $Y_{ij} = X_{ji}$.

$$\mathbf{X} = \begin{bmatrix} 1 & 4 \\ 3 & 6 \\ -2 & 11 \end{bmatrix}, \quad \mathbf{X}^\top = \begin{bmatrix} 1 & 3 & -2 \\ 4 & 6 & 11 \end{bmatrix}.$$

Comment 1.7 – Matrix multiplication: To proceed, we must introduce the concept of matrix multiplication. Taking the product, \mathbf{AB} , of an $N \times M$ matrix \mathbf{A} and a $P \times Q$ matrix \mathbf{B} is only possible if $M = P$, i.e. the number of columns in \mathbf{A} is equal to the number of rows in \mathbf{B} . Assuming that this is the case, the product, $\mathbf{C} = \mathbf{AB}$, is the $N \times Q$ matrix defined such that

$$C_{ij} = \sum_k A_{ik} B_{kj}.$$

It is often helpful to draw the matrices, for example,

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \\ a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \end{bmatrix}$$

where we can think of computing elements of \mathbf{C} by working simultaneously across the relevant row of \mathbf{A} and column of \mathbf{B} .

A special case that we will meet regularly is the *inner product* between two column vectors, defined as $z = \mathbf{x}^\top \mathbf{y}$, the result of which is a scalar. Both vectors must be of the same length and the transpose ensures that the number of columns in \mathbf{x} is the same as the number of rows in \mathbf{y} . Applying the same technique as that for matrices, we see that

$$z = \sum_k x_k y_k.$$

Therefore, if we perform the matrix multiplication \mathbf{Xw} we will end up with a vector which looks like this:

$$\mathbf{Xw} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \times \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} w_0 + w_1 x_1 \\ w_0 + w_1 x_2 \\ \vdots \\ w_0 + w_1 x_N \end{bmatrix}.$$

Subtracting this from \mathbf{t} will give us:

$$\mathbf{t} - \mathbf{X}\mathbf{w} = \begin{bmatrix} t_1 - w_0 - w_1x_1 \\ t_2 - w_0 - w_1x_2 \\ \vdots \\ t_N - w_0 - w_1x_N \end{bmatrix}.$$

and we can use a single multiplication and transpose to neatly perform the squaring and summation and obtain our original loss function:

$$\begin{aligned} (\mathbf{X}\mathbf{w} - \mathbf{t})^\top(\mathbf{X}\mathbf{w} - \mathbf{t}) &= (w_0 + w_1x_1 - t_1)^2 + (w_0 + w_1x_2 - t_2)^2 + \dots \\ &\quad + (w_0 + w_1x_N - t_N)^2 \\ &= \sum_{n=1}^N (w_0 + w_1x_n - t_n)^2 \\ &= \sum_{n=1}^N (t_n - f(x_n; w_0, w_1))^2. \end{aligned}$$

Therefore, our loss can be written compactly as:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^\top(\mathbf{t} - \mathbf{X}\mathbf{w}), \quad (1.13)$$

and the following loss expressions are all equivalent:

$$\mathcal{L} = \frac{1}{N}(\mathbf{t} - \mathbf{X}\mathbf{w})^\top(\mathbf{t} - \mathbf{X}\mathbf{w}) = \frac{1}{N} \sum_{n=1}^N (t_n - \mathbf{w}^\top \mathbf{x}_n)^2 = \frac{1}{N} \sum_{n=1}^N (t_n - (w_0 + w_1x_n))^2$$

Comment 1.8 – Transpose of a product: The transpose of a matrix product, $(\mathbf{X}\mathbf{w})^\top$, can be expanded by reversing the order of multiplication and transposing the two individual matrices

$$(\mathbf{X}\mathbf{w})^\top = \mathbf{w}^\top \mathbf{X}^\top.$$

To deal with more complex forms, we can apply the same result several times. For example,

$$\begin{aligned} (\mathbf{ABCD})^\top &= ((\mathbf{AB})(\mathbf{CD}))^\top \\ &= (\mathbf{CD})^\top(\mathbf{AB})^\top \\ &= \mathbf{D}^\top \mathbf{C}^\top \mathbf{B}^\top \mathbf{A}^\top \end{aligned}$$

It will be easier to work with this matrix loss once we have multiplied out the brackets. Noting that order is important in matrix multiplication (this is implied by the restriction on sizes discussed in Comment 1.7) and the definition for the transpose of a product given in Comment 1.8:

$$\begin{aligned} \mathcal{L} &= \frac{1}{N}(\mathbf{X}\mathbf{w} - \mathbf{t})^\top(\mathbf{X}\mathbf{w} - \mathbf{t}) \\ &= ((\mathbf{X}\mathbf{w})^\top - \mathbf{t}^\top)(\mathbf{X}\mathbf{w} - \mathbf{t}) \\ &= \frac{1}{N}(\mathbf{X}\mathbf{w})^\top \mathbf{X}\mathbf{w} - \frac{1}{N} \mathbf{t}^\top \mathbf{X}\mathbf{w} - \frac{1}{N}(\mathbf{X}\mathbf{w})^\top \mathbf{t} + \frac{1}{N} \mathbf{t}^\top \mathbf{t} \\ &= \frac{1}{N} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} - \frac{2}{N} \mathbf{w}^\top \mathbf{X}^\top \mathbf{t} + \frac{1}{N} \mathbf{t}^\top \mathbf{t}. \end{aligned} \quad (1.14)$$

The two terms $\mathbf{t}^\top \mathbf{X} \mathbf{w}$ and $\mathbf{w}^\top \mathbf{X}^\top \mathbf{t}$ are the transpose of one another (using the identity for the transpose of a product) and also scalars (satisfy yourself that the result is a 1×1 matrix and hence a scalar). This implies that they must be the same and can therefore be combined.

Differentiating loss in vector/matrix form: We now require the value of the vector \mathbf{w} corresponding to a turning point (minimum) of \mathcal{L} . To do this, we must take the partial derivative of \mathcal{L} with respect to the *vector* \mathbf{w} . This involves taking partial derivatives of \mathcal{L} with respect to each element of \mathbf{w} in turn and then stacking the results into a vector. It is worth explicitly doing this in this instance, although we will see later that we can actually obtain $\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$ directly in vector form. In our two variable case, this vector is

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial w_0} \\ \frac{\partial \mathcal{L}}{\partial w_1} \end{bmatrix},$$

the vector containing the partial derivatives of \mathcal{L} with respect to w_0 and w_1 . The two elements of this vector should be the same as Equations 1.7 and 1.6, respectively. We can check that our loss is indeed correct by manually differentiating Equation 1.13 with respect to the two parameters. First, we need the multiplied out expression

$$\mathcal{L} = \frac{1}{N} (\mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{t} + \mathbf{t}^\top \mathbf{t}).$$

The last term doesn't include either w_0 or w_1 so we can ignore it. When multiplied out, the first term is (see Exercise EX 1.3)

$$w_0^2 \frac{1}{N} \left(\sum_{n=1}^N X_{n0}^2 \right) + 2w_0 w_1 \frac{1}{N} \left(\sum_{n=1}^N X_{n0} X_{n1} \right) + w_1^2 \frac{1}{N} \left(\sum_{n=1}^N X_{n1}^2 \right),$$

where X_{n0} is the first element of the n th row of \mathbf{X} , i.e. the first element of the n th data object and X_{n1} is the second (we've started numbering from zero to maintain the relationship with w_0). Similarly, the second term is equivalent to

$$2w_0 \frac{1}{N} \left(\sum_{n=1}^N X_{n0} t_n \right) + 2w_1 \frac{1}{N} \left(\sum_{n=1}^N X_{n1} t_n \right).$$

Combining these and noting that, in our previous notation, $X_{n0} = 1$ and $X_{n1} = x_n$ results in

$$w_0^2 + 2w_0 w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n \right) + w_1^2 \frac{1}{N} \left(\sum_{n=1}^N x_{n1}^2 \right) - 2w_0 \frac{1}{N} \left(\sum_{n=1}^N t_n \right) - 2w_1 \frac{1}{N} \left(\sum_{n=1}^N x_n t_n \right).$$

Recalling our shorthand for the various averages and differentiating with respect to w_0 and w_1 results in

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_0} &= 2w_0 + 2w_1 \bar{x} - 2\bar{t} \\ \frac{\partial \mathcal{L}}{\partial w_1} &= 2w_0 \bar{x} + 2w_1 \bar{x^2} - 2\bar{t}. \end{aligned}$$

It is left as an informal exercise to show that these are indeed equivalent to the derivates obtained from the non-vectorised loss function (Equations 1.7 and 1.6).

Fortunately, there are many standard identities that we can use that enable us

TABLE 1.4: Some useful identities when differentiating with respect to a vector.

$f(\mathbf{w})$	$\frac{\partial f}{\partial \mathbf{w}}$
$\mathbf{w}^T \mathbf{x}$	\mathbf{x}
$\mathbf{x}^T \mathbf{w}$	\mathbf{x}
$\mathbf{w}^T \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^T \mathbf{Cw}$	$2\mathbf{Cw}$

to differentiate the vectorised expression directly. Those that we will need are shown in Table 1.4.

From these identities, and equating the derivative to zero, we can directly obtain the following:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{w}} &= \frac{2}{N} \mathbf{X}^T \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{X}^T \mathbf{t} = 0 \\ \mathbf{X}^T \mathbf{X} \mathbf{w} &= \mathbf{X}^T \mathbf{t}.\end{aligned}\quad (1.15)$$

Comment 1.9 – Identity matrix: we will regularly come across the identity matrix \mathbf{I}_N . It is the $N \times N$ matrix with ones on the diagonal and zeros elsewhere.

$$\mathbf{I}_1 = 1, \quad \mathbf{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \mathbf{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Often, the size of the identity matrix will be obvious from the expression it's found in. In these cases, we will omit the size subscript.

A key property of the identity matrix is that any vector or matrix multiplied by a suitably sized identity matrix is equal to the original matrix or vector. For example, if $\mathbf{y} = [y_1, \dots, y_D]^T$ and \mathbf{I}_D is the $D \times D$ identity matrix,

$$\mathbf{y}^T \mathbf{I}_D = \mathbf{y}, \quad \mathbf{I} \mathbf{y} = \mathbf{y}.$$

Similarly, for an $N \times M$ matrix,

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1M} \\ a_{21} & a_{22} & \dots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NM} \end{bmatrix}$$

$$\mathbf{A} \mathbf{I}_M = \mathbf{A}, \quad \mathbf{I}_N \mathbf{A} = \mathbf{A}.$$

Multiplying a scalar by an identity results in a matrix with the scalar value on each diagonal element. An example that crops up a lot is:

$$\sigma^2 \mathbf{I}_M = \begin{bmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{bmatrix}$$

Comment 1.10 – Matrix inverse: The inverse of a matrix \mathbf{A} is defined as the matrix \mathbf{A}^{-1} that satisfies $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. We don't provide the general form for inverting a matrix here, but from school mathematics, a 2×2 matrix can be inverted with the following formula:

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad \mathbf{A}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

A special case that we will come across regularly is the inverse of a matrix that only has values on the diagonal (i.e. all off-diagonal elements are zero). The inverse of such a matrix is another diagonal matrix where each diagonal element is simply the inverse of the corresponding element in the original. For example,

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{DD} \end{bmatrix}, \quad \mathbf{A}^{-1} = \begin{bmatrix} a_{11}^{-1} & 0 & \dots & 0 \\ 0 & a_{22}^{-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{DD}^{-1} \end{bmatrix}$$

It is worth noting that this definition implies that the inverse of an identity matrix (see Comment 1.9) is simply another identity matrix:

$$\mathbf{I}^{-1} = \mathbf{I}.$$

The final step in deriving an expression $\hat{\mathbf{w}}$, the optimum value of \mathbf{w} , is re-arranging Equation 1.15. We cannot divide both sides by $\mathbf{X}^T\mathbf{X}$ (division isn't defined for matrices) but we can pre-multiply both sides by a matrix that will cancel the $\mathbf{X}^T\mathbf{X}$ from the left (leaving only an identity matrix; see Comment 1.9). This matrix is called the matrix inverse of $\mathbf{X}^T\mathbf{X}$ (see Comment 1.10) and is denoted by $(\mathbf{X}^T\mathbf{X})^{-1}$. Pre-multiplying both sides of (1.15) with $(\mathbf{X}^T\mathbf{X})^{-1}$, we obtain:

$$\mathbf{I}\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t}.$$

As $\mathbf{I}\mathbf{w} = \mathbf{w}$ (from the definition of the identity matrix), we are left with a matrix equation for $\hat{\mathbf{w}}$, the value of \mathbf{w} that minimises the loss:

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{t} \quad (1.16)$$

1.3.1 Example

We can check that our matrix equation is doing exactly the same as the scalar equations we got previously by multiplying it out. In two dimensions,

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} \sum_{n=1}^N x_{n0}^2 & \sum_{n=1}^N x_{n0}x_{n1} \\ \sum_{n=1}^N x_{n1}x_{n0} & \sum_{n=1}^N x_{n1}^2 \end{bmatrix}.$$

Using \bar{x} to denote averages, this can be re-written as

$$\mathbf{X}^T\mathbf{X} = N \begin{bmatrix} \bar{x}_0^2 & \bar{x}_0\bar{x}_1 \\ \bar{x}_1\bar{x}_0 & \bar{x}_1^2 \end{bmatrix}.$$

The identity for the inverse of a 2×2 matrix (see Comment 1.10) enables us to invert this

$$(\mathbf{X}^T \mathbf{X})^{-1} = \frac{1}{N} \frac{1}{\overline{x_0^2} \overline{x_1^2} - \overline{x_1 x_0} \overline{x_0 x_1}} \begin{bmatrix} \overline{x_1^2} & -\overline{x_0 x_1} \\ -\overline{x_1 x_0} & \overline{x_0^2} \end{bmatrix}.$$

We need to multiply this by $\mathbf{X}^T \mathbf{t}$, which is (jumping straight to the average notation):

$$N \begin{bmatrix} \overline{x_0 t} \\ \overline{x_1 t} \end{bmatrix}.$$

Now, we know that x_{n0} is 1 always and redefining x_{n1} as x_n (to be consistent with the scalar notation), we need to evaluate:

$$\hat{\mathbf{w}} = \frac{1}{N} \frac{1}{\overline{x^2} - \overline{x} \overline{x}} \begin{bmatrix} \overline{x^2} & -\overline{x} \\ -\overline{x} & 1 \end{bmatrix} \times N \begin{bmatrix} \overline{t} \\ \overline{x t} \end{bmatrix},$$

which is:

$$\hat{\mathbf{w}} = \begin{bmatrix} \widehat{w}_0 \\ \widehat{w}_1 \end{bmatrix} = \left(\frac{1}{\overline{x^2} - \overline{x} \overline{x}} \right) \begin{bmatrix} \overline{x^2} \overline{t} - \overline{x} \overline{x t} \\ -\overline{x} \overline{t} + \overline{x t} \end{bmatrix}. \quad (1.17)$$

Starting with \widehat{w}_1 (the second line)

$$\widehat{w}_1 = \frac{\overline{x t} - \overline{x} \overline{t}}{\overline{x^2} - \overline{x} \overline{x}},$$

exactly as before. \widehat{w}_0 requires a little more rearrangement and it is easier to work backwards. Starting from our original expression and substituting this new expression for \widehat{w}_1

$$\begin{aligned} \widehat{w}_0 &= \overline{t} - \widehat{w}_1 \overline{x} \\ &= \overline{t} - \overline{x} \frac{\overline{x t} - \overline{x} \overline{t}}{\overline{x^2} - \overline{x} \overline{x}} \\ &= \overline{t} \left(\frac{\overline{x^2} - \overline{x} \overline{x}}{\overline{x^2} - \overline{x} \overline{x}} \right) - \overline{x} \frac{\overline{x t} - \overline{x} \overline{t}}{\overline{x^2} - \overline{x} \overline{x}} \\ &= \frac{\overline{t} \overline{x^2} - \overline{t} \overline{x} \overline{x} - \overline{x} \overline{x t} + \overline{x} \overline{x} \overline{t}}{\overline{x^2} - \overline{x} \overline{x}} \\ &= \frac{\overline{t} \overline{x^2} - \overline{x} \overline{x t}}{\overline{x^2} - \overline{x} \overline{x}}, \end{aligned}$$

which is exactly the first line in Equation 1.17 as required.

1.3.2 Numerical example

To help those readers who are not familiar with working with vectors and matrices, we will now repeat the synthetic linear regression example we saw in the previous section. The data in matrix notation is:

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 5 \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} 4.8 \\ 11.3 \\ 17.2 \end{bmatrix}.$$

Examining Equation 1.16 we see that the first quantity we need to calculate is $\mathbf{X}^T \mathbf{X}$:

$$\mathbf{X}^T \mathbf{X} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 1 & 3 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 9 \\ 9 & 35 \end{bmatrix}.$$

Using the formula provided above, we compute the inverse as

$$(\mathbf{X}^T \mathbf{X})^{-1} = \frac{1}{24} \begin{bmatrix} 35 & -9 \\ -9 & 3 \end{bmatrix}.$$

Multiplying by \mathbf{X}^T ,

$$(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T = \frac{1}{24} \begin{bmatrix} 35 & -9 \\ -9 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 5 \end{bmatrix} = \frac{1}{24} \begin{bmatrix} 26 & 8 & -10 \\ -6 & 0 & 6 \end{bmatrix}.$$

Finally, we multiply this matrix by \mathbf{t} :

$$((\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T) \mathbf{t} = \frac{1}{24} \begin{bmatrix} 26 & 8 & -10 \\ -6 & 0 & 6 \end{bmatrix} \times \begin{bmatrix} 4.8 \\ 11.3 \\ 17.2 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 3.1 \end{bmatrix}.$$

Therefore, our formula is $f(x; w_0, w_1) = 1.8 + 3.1x$, exactly as before.

1.3.3 Making predictions

Given a new vector of attributes x_{new} , the prediction from the model t_{new} is computed as:

$$t_{\text{new}} = \hat{\mathbf{w}}^T \mathbf{x}_{\text{new}}.$$

1.3.4 Summary

In the previous sections, we have described our linear model in terms of vectors and matrices. The result is a very useful model – our expression for $\hat{\mathbf{w}}$ makes no assumptions as to the number of parameters included in $\hat{\mathbf{w}}$ (its length). We can therefore compute $\hat{\mathbf{w}}$ and make predictions for any linear model of the form:

$$t_n = w_1 x_{n1} + w_2 x_{n2} + w_3 x_{n3} + \dots$$

This is a powerful tool – many real datasets are described by more than one attribute and for many, a linear model of this type will be appropriate. We have also learnt that predictions from this model are very precise and that this is not always sensible. We shall see how to overcome this in later chapters.

The attributes that make up \mathbf{x}_n could be measurements of different properties (e.g. winning times and personal bests). Alternatively, they could be the result of applying a set of functions to an individual attribute like the Olympics year: x_n . This allows us to extend our armoury beyond models corresponding to straight lines and is the subject of the next section.

1.4 Nonlinear response from a linear model

At the start of this chapter, we made the assumption that we could model the relationship between time and Olympics 100 m sprint times using a linear function. In many real applications, this is too restrictive. Even for the 100 m data it could be argued that it is far too simplistic – the linear model predicts that in the year 3000, the time will be -3.5 seconds! Fortunately, we can use exactly the same framework we have already described to fit a family of more complex models through a transformation of the attributes.

The linear model we have seen thus far,

$$f(x; \mathbf{w}) = w_0 + w_1 x,$$

is linear in both the parameters (w) and the data (x) (see Comment 1.1). The linearity in the parameters is desirable from a computational point of view as the solution that minimises the squared loss can be found exactly via Equations 1.8 and 1.10. Consider augmenting our data matrix \mathbf{X} with an additional column, x_n^2 :

$$\mathbf{x}_n = \begin{bmatrix} 1 \\ x_n \\ x_n^2 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix}$$

and adding an extra parameter to \mathbf{w} :

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix},$$

resulting in:

$$f(x; \mathbf{w}) = \mathbf{w}^\top \mathbf{x} = w_0 + w_1 x + w_2 x^2.$$

As the model is still linear in the parameters, we can use Equation 1.16 to find \mathbf{w} but the function we are fitting is **quadratic** in the data. Figure 1.9 shows an example of using exactly this method to fit a function quadratic in the data to a suitable dataset (solid line) (MATLAB script: `synthquad.m`). Also shown is the function we get if we try and fit our original linear (in the data) model (dashed line, $t = w_0 + w_1 x$). It is clear from the quality of the fit to the data that the quadratic model is a more appropriate model for this data.

More generally, we can add as many powers of x as we like to get a **polynomial** function of any order. For a K th order polynomial, our augmented data matrix will be:

$$\mathbf{X} = \begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \cdots & x_1^K \\ x_2^0 & x_2^1 & x_2^2 & \cdots & x_2^K \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ x_N^0 & x_N^1 & x_N^2 & \cdots & x_N^K \end{bmatrix} \quad (1.18)$$

(where $x^0 = 1$ and our function can be written in the more general form:

$$f(x; \mathbf{w}) = \sum_{k=0}^K w_k x^k.$$

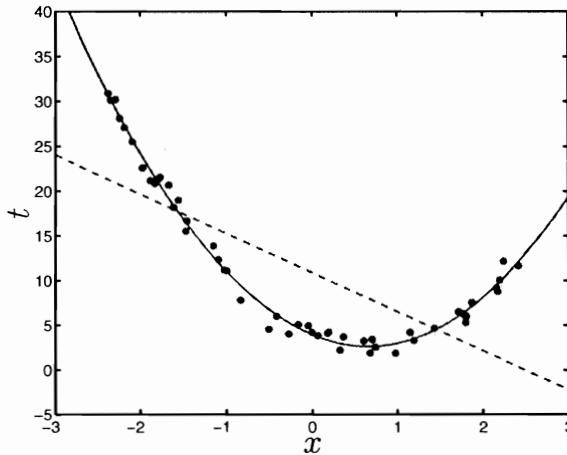


FIGURE 1.9: Example of linear and quadratic models fitted to a dataset generated from a quadratic function.

Figure 1.10 shows the effect of fitting an 8th order polynomial function to the 100 m sprint data that we have seen previously (MATLAB script: `olympoly.m`). Comparing with Figures 1.5 and 1.6, does the 8th order model look better than the 1st order model? To answer this question, we need to be more precise about what we mean by better. For models built to make predictions, the best model is arguably the one that produces the best predictions. We shall return to the issue of model selection in more detail in Section 1.5. However, two things are immediately apparent and warrant description. First, the 8th order polynomial gets closer to the observed data than the 1st order polynomial (original model). This is reflected in a lower value of the loss function: $\mathcal{L}^8 = 0.459$, $\mathcal{L}^1 = 1.358$ (where \mathcal{L}^k is the loss achieved with a k th order polynomial). In fact, increasing the polynomial order will always result in a model that gets closer to the training data. Second, the predictions (shown by the dashed line) do not look sensible, particularly outside the range of the observed data.

We are not restricted to polynomial functions. We are free to define any set of K functions of x , $h_k(x)$:

$$\mathbf{X} = \begin{bmatrix} h_1(x_1) & h_2(x_1) & \cdots & h_K(x_1) \\ h_1(x_2) & h_2(x_2) & \cdots & h_K(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(x_N) & h_2(x_N) & \cdots & h_K(x_N) \end{bmatrix}$$

which can be anything that we feel may be appropriate for the data available. For example, there appears to be a slight periodic trend in the 100 m data. A suitable

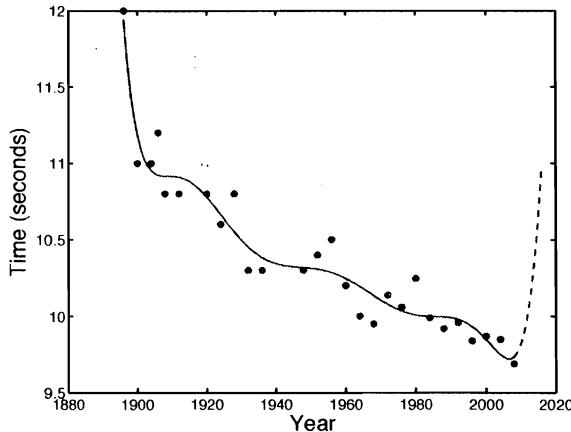


FIGURE 1.10: 8th order polynomial fitted to the Olympics 100 m men's sprint data.

set of functions might be:

$$\begin{aligned} h_1(x) &= 1 \\ h_2(x) &= x \\ h_3(x) &= \sin\left(\frac{x-a}{b}\right) \\ f(x; \mathbf{w}) &= w_0 + w_1x + w_2 \sin\left(\frac{x-a}{b}\right). \end{aligned}$$

This model has 5 parameters – w_0, w_1, w_2, a, b . Unfortunately, only the first three can be inferred using the procedures that we have developed. The last two, a and b , appear inside a nonlinear (sine) function. As such, taking partial derivatives with respect to these parameters and equating to zero will not result in a set of equations that can be solved analytically. There are many ways of overcoming this problem, the simplest being a search over all values of a and b in some sensible range. However, we will ignore this problem for now and assume that we know of suitable values. If a and b are fixed, we can set the remaining parameters (w_0, w_1, w_2) using the expressions we derived previously. Assuming a and b are fixed ($a = 2660, b = 4.3$), Figure 1.11 shows a least squares fit using this model. In this case $\mathcal{L} = 1.1037$ so it is fitting the observed data better than the 1st order polynomial but not as well as the 8th order polynomial. The various model components are clearly visible in Figure 1.11: the constant term ($w_0 = 36.610$), the downward linear trend ($w_1 = -0.013$) and the nonlinear sinusoidal term ($w_3 = -0.133$) causing oscillations. Notice also how the values for w_0 and w_1 are very similar to those for the 1st order polynomial model (c.f. Figure 1.5) – we have added an oscillating component around our original linear model.

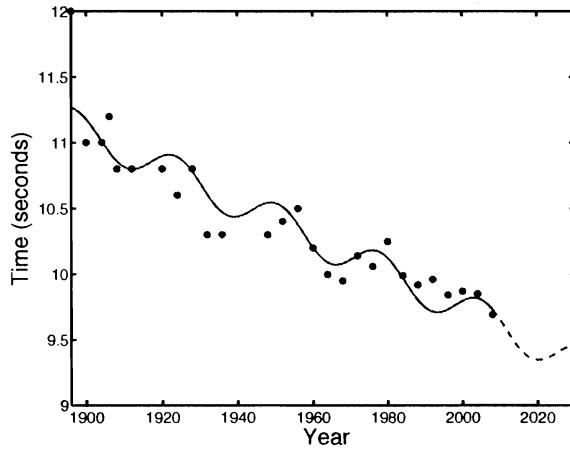


FIGURE 1.11: Least squares fit of $f(x; \mathbf{w}) = w_0 + w_1x + w_2 \sin\left(\frac{x-a}{b}\right)$ to the 100 m sprint data ($a = 2660, b = 4.3$).

1.5 Generalisation and over-fitting

In Section 1.4, we posed the question of which was better, the 1st or 8th order polynomial. Given that our original aim in building these models was to make predictions, it makes sense that the best model is the one which is able to make the most accurate predictions. Such a model will be one that can *generalise* beyond the examples we have for *training* (our Olympics data up to 2008, for example). Ideally, we would like to choose the model that performs best (i.e. minimises the loss) on this unseen data but, by the very nature of the problem, this data is unavailable.

Figure 1.10 gave an early indication that we should be very suspicious of using the loss on the training data to choose a model that will be used to make predictions. The plot shows an 8th order polynomial fit to the men's 100 m data which has a much lower loss on the training data than a 1st order polynomial. At the same time, the predictions for future Olympics are very poor. For this data, a model based on an 8th order polynomial pays too much attention to the training data (it *over-fits*) and as a result does not generalise well to new data. As we make models more and more complex, they will be able to get closer and closer to the data that we have already seen. Unfortunately, beyond a certain point, the quality of the predictions can deteriorate rapidly. Determining the optimal model complexity such that it is able to generalise well without **over-fitting** is very challenging. This tradeoff is often referred to as the bias-variance tradeoff and we will briefly mention this in Section 2.8.

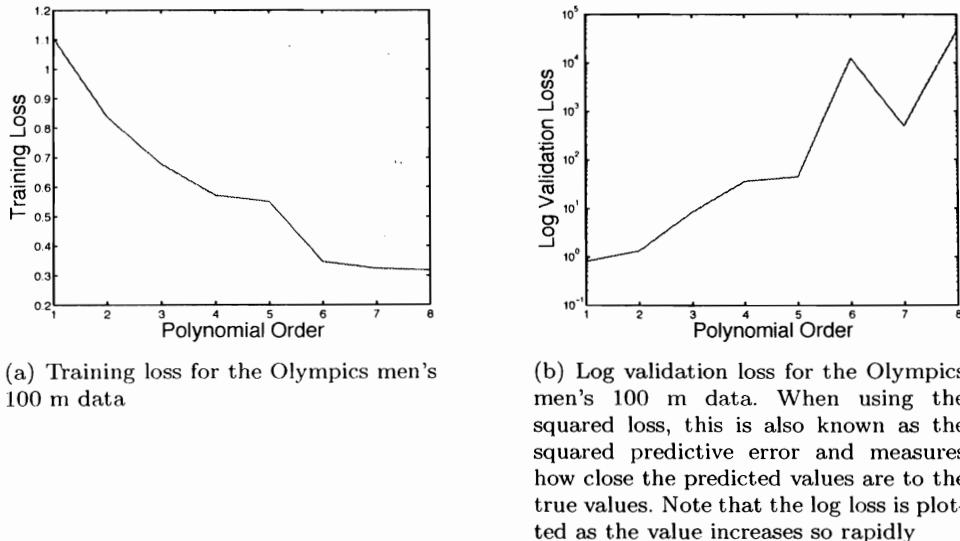


FIGURE 1.12: Training and validation loss for Olympics men's 100 m data.

1.5.1 Validation data

One common way to overcome this problem is to use a second dataset, often referred to as a **validation** set. It is so called as it is used to validate the predictive performance of our model. The validation data could be provided separately or we could create it by removing some data from the original *training* set. For example, in our 100 m data, we could remove all Olympics since 1980 from the training set and make these the validation set. To choose between a set of models, we train each one on the reduced training set and then compute their loss on the validation set. Plots of the training and (log) validation losses can be seen in Figure 1.12(a) and Figure 1.12(b), respectively. The training loss decreases **monotonically** as the polynomial order (and hence **model complexity**) increases. However, the validation loss increases rapidly as the polynomial order increases, suggesting that a 1st order polynomial has the best **generalisation** ability and will produce the most reliable predictions. This hypothesis is easily tested. In Figure 1.13 we can see the data (labeled as training and validation) and 1st, 4th and 8th order polynomial functions (MATLAB script: `olympval.m`). It is clear to see that for this data, had we been performing this task in 1979, a 1st order model would indeed have given the best predictions.

1.5.2 Cross-validation

The loss that we calculate from validation data will be sensitive to the choice of data in our validation set. This is particularly problematic if our dataset (and hence our validation set) is small. **Cross-validation** is a technique that allows us to make more efficient use of the data we have.

K -fold cross-validation splits the data into K equally (or as close to equal as possible) sized blocks, illustrated in Figure 1.14. Each block takes its turn as a

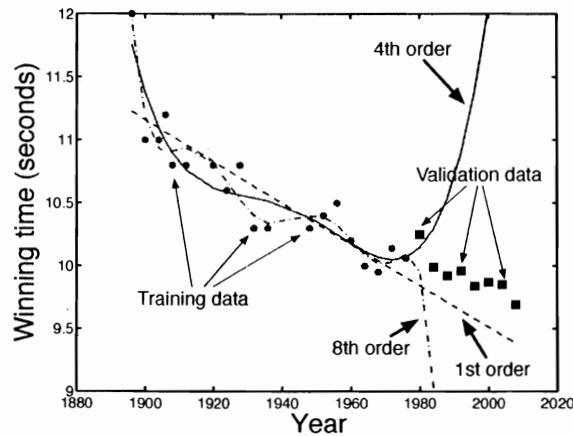


FIGURE 1.13: Generalisation ability of 1st, 4th and 8th order polynomials on Olympics men's 100 m data.

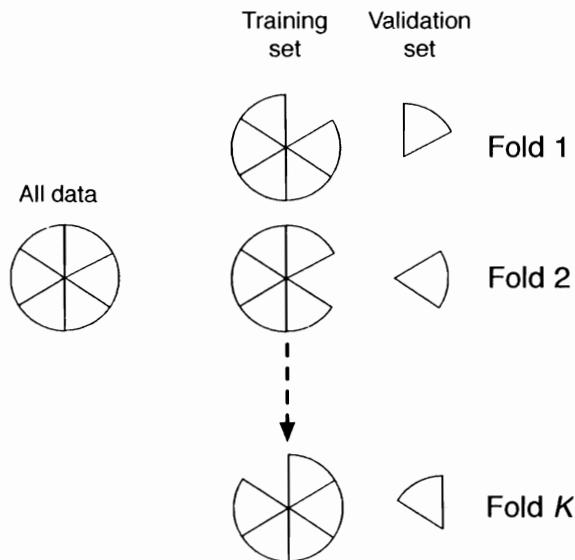


FIGURE 1.14: Cross-validation. The dataset is depicted on the left as a pie chart. In each of the K folds, one set of data points is removed from the training set and used to validate or test the model.

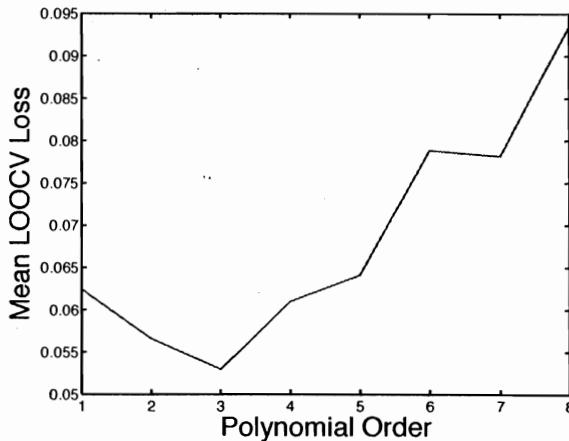


FIGURE 1.15: Mean LOOCV loss as polynomials of increasing order are fitted to the Olympics men's 100 m data.

validation set for a training set comprised of the other $K - 1$ blocks. Averaging over the resulting K loss values gives us our final loss value. An extreme case of K -fold cross-validation is where $K = N$, the number of observations in our dataset: each data observation is held out in turn and used to test a model trained on the other $N - 1$ objects. This particular form of cross-validation is given the name Leave-One-Out Cross Validation (LOOCV). The average squared validation loss for LOOCV is:

$$\mathcal{L}^{CV} = \frac{1}{N} \sum_{n=1}^N (t_n - \hat{\mathbf{w}}_{-n}^\top \mathbf{x}_n)^2, \quad (1.19)$$

where $\hat{\mathbf{w}}_{-n}$ is the estimate of the parameters without the n th training example.

The mean LOOCV error for the Olympics men's 100 m data can be seen in Figure 1.15. This plot suggests that a 3rd order polynomial would be best. This is in disagreement with the value obtained from using the last few data points as a validation set. Disagreement like this is not uncommon – **model selection** is a very difficult problem. However, the two methods do agree on one thing – the model certainly shouldn't be 6th order or above.

One drawback of illustrating model selection on a real dataset is that we don't know what the 'true' model is and therefore don't know if our selection techniques are working. We can overcome this by generating a synthetic dataset. Fifty input-target pairs were generated from a noisy 3rd-order polynomial function and used to learn polynomial functions of increasing order (from 1st to 7th). Ideally, we hope to see minimum validation loss for the true polynomial order of 3. A further 1000 input-target pairs were generated from the true function and are used as an independent test set with which to compute an additional, independent loss. This very large dataset will give us a good approximation to the true expected loss against which we can compare the LOOCV loss.

The results can be seen in Figure 1.16 (MATLAB script: `cv_demo.m`). As we have already discovered, the training loss keeps decreasing as the order increases.

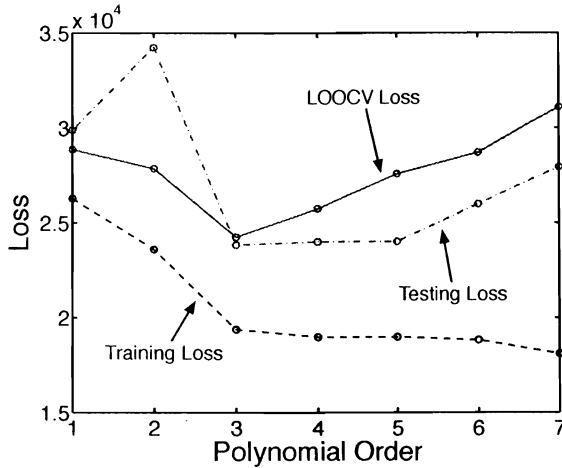


FIGURE 1.16: The training, testing and leave-one-out loss curves obtained for a noisy cubic function where a sample size of 50 is available for training and LOOCV estimation. The test error is computed using 1000 independent samples.

The LOOCV loss and the test loss decrease as the order is increased to 3 and then increase as the order is increased further. Either of these validation methods would have predicted the correct model order. Unfortunately, we will rarely be able to call upon 1000 independent points from outside our training set and will heavily rely on a cross-validation scheme, often LOOCV.

1.5.3 Computational scaling of K -fold cross-validation

LOO cross-validation appears to be a good means of estimating our expected loss from the training data, allowing us to explore and assess various alternative models. However, consider implementing LOOCV. We need to train our model N times, which will take roughly N times longer than training it once on all of the data (it is not exactly the same, as we will be training it on one less data point). For some models, particularly if we have a lot of data, this might not be feasible.

This simplest way to alleviate this problem is to use $K \ll N$. For example, in 10-fold cross-validation, we would leave out 10% of the data for validation and use the remaining 90% for training. This reduces the number of training loops from N to 10 – a considerable saving if $N \gg 10$. A popular choice is to use N -fold cross-validation and repeat it several times with the data partitioned differently into the N groups, allowing averages to be taken across both folds and repetitions.

1.6 Regularised least squares

In the previous section, we discussed how predictions on data that was not part of the training set could be used to ensure good predictive performance (good generalisation) and prevent the model from over-fitting. In essence, this stops our model from becoming too complex. However, there is another way that this can be done, known as **regularisation**.

Consider a trivial model, defined by $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$ where $\mathbf{w} = [0, 0, \dots, 0]^\top$ – the model always predicts a value of 0. This is the simplest model possible. Any change we make to the elements of \mathbf{w} increases their absolute value and makes the model more complex. Specifically, consider the 5th order polynomial model

$$f(x; \mathbf{w}) = w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4 + w_5x^5.$$

If we start with all of the elements of \mathbf{w} being zero, the function always predicts a value of zero. Now imagine we set w_0 to some non-zero value. The model now predicts a constant (w_0). Leaving w_0 at its new value, we can set w_1 to some value. The model has become more complex and as each additional parameter is given a non-zero value, the model becomes more complex still. In general, we could consider that the higher the sum of the absolute values in \mathbf{w} , the more complex the model (note that it is the absolute value – we don't want the positive values to cancel with the negative ones). Alternatively, because absolute values tend to make the maths a bit harder, we could define the complexity of our model as

$$\sum_i w_i^2$$

or, in vector form,

$$\mathbf{w}^\top \mathbf{w}.$$

As we don't want our model to become too complex, it makes sense to try and keep this value low. So, rather than just minimising the average squared loss \mathcal{L} , we could minimise a regularised loss \mathcal{L}' made by adding together our previous loss and a term penalising over-complexity:

$$\mathcal{L}' = \mathcal{L} + \lambda \mathbf{w}^\top \mathbf{w} \quad (1.20)$$

The parameter λ controls the trade-off between penalising not fitting the data well (\mathcal{L}) and penalising overly complex models ($\mathbf{w}^\top \mathbf{w}$). We can find the optimal value of \mathbf{w} in exactly the same way as before. Adding the regularisation term to our original squared loss (Equation 1.14) gives:

$$\mathcal{L}' = \frac{1}{N} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{w}^\top \mathbf{X}^\top \mathbf{t} + \frac{1}{N} \mathbf{t}^\top \mathbf{t} + \lambda \mathbf{w}^\top \mathbf{w}.$$

Taking partial derivatives with respect to \mathbf{w}

$$\frac{\partial \mathcal{L}'}{\partial \mathbf{w}} = \frac{2}{N} \mathbf{X}^\top \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{X}^\top \mathbf{t} + 2\lambda \mathbf{w}.$$

Setting this expression to zero and solving for \mathbf{w} gives

$$\begin{aligned}\frac{2}{N} \mathbf{X}^T \mathbf{X} \mathbf{w} - \frac{2}{N} \mathbf{X}^T \mathbf{t} + 2\lambda \mathbf{w} &= 0 \\ (\mathbf{X}^T \mathbf{X} + N\lambda \mathbf{I})\mathbf{w} &= \mathbf{X}^T \mathbf{t}.\end{aligned}$$

Hence, the regularised least squares solution is given by:

$$\hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + N\lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{t}. \quad (1.21)$$

Clearly, if $\lambda = 0$ we retrieve the original solution. We can see the effect of increasing λ with a synthetic example. Figure 1.17 shows 6 synthetic data points. A 5th order polynomial function can fit the 6 data points exactly and we can see this if we set $\lambda = 0$ (in general, N data-points can be perfectly fitted by a $(N - 1)$ th order polynomial). If we increase λ we begin to see the regularisation taking effect. $\lambda = 1e - 06$ follows the general shape of the exact 5th order polynomial but without as much variability and subsequently is further from the data points. $\lambda = 0.01$ and $\lambda = 0.1$ continue this trend – the function becomes less complex (MATLAB script: `regls.m`).

Choosing the value of λ presents us with the same over-fitting/generalisation trade-off we had when choosing the polynomial order. If it is too small, our function is likely to be too complex. Too large, and we will not capture any useful trends in the data. Fortunately, we can use exactly the validation techniques introduced in the previous section to determine the best value of λ . In particular, it is common to use cross-validation to choose the value of λ that gives the best predictive performance (see Exercise EX 1.12).

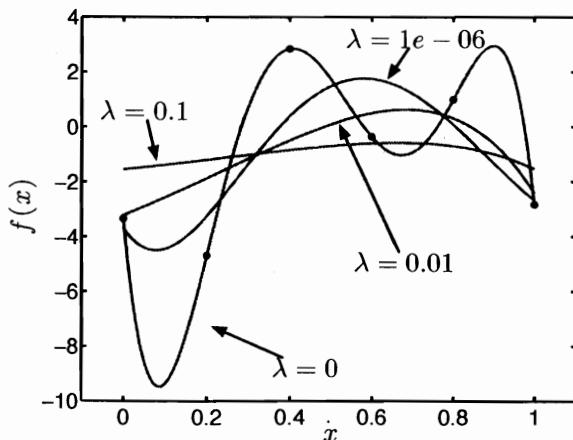


FIGURE 1.17: Effect of varying the regularisation parameter λ for a 5th order polynomial function.

1.7 Exercises

- EX 1.1. By examining Figure 1.1 estimate the kind of values we should expect for w_0 and w_1 (e.g. High? Low? Positive? Negative?).
- EX 1.2. Write a Matlab script that can find w_0 and w_1 for an arbitrary dataset of x_n, t_n pairs.
- EX 1.3. Show that:

$$\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} = w_0^2 \left(\sum_{n=1}^N x_{n1}^2 \right) + 2w_0 w_1 \left(\sum_{n=1}^N x_{n1} x_{n2} \right) + w_1^2 \left(\sum_{n=1}^N x_{n2}^2 \right),$$

where

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \\ \vdots & \vdots \\ x_{N1} & x_{N2} \end{bmatrix}.$$

(Hint – it's probably easiest to do the $\mathbf{X}^T \mathbf{X}$ first!)

- EX 1.4. Using \mathbf{w} and \mathbf{X} as defined in the previous exercise, show that $(\mathbf{X}\mathbf{w})^T = \mathbf{w}^T \mathbf{X}^T$ by multiplying out both sides.
- EX 1.5. When multiplying a scalar by a vector (or matrix), we multiply each element of the vector (or matrix) by that scalar. For $\mathbf{x}_n = [x_{n1}, x_{n2}]^T$, $\mathbf{t} = [t_1, \dots, t_N]^T$, $\mathbf{w} = [w_0, w_1]^T$ and

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}$$

show that

$$\sum_n \mathbf{x}_n t_n = \mathbf{X}^T \mathbf{t}$$

and

$$\sum_n \mathbf{x}_n \mathbf{x}_n^T \mathbf{w} = \mathbf{X}^T \mathbf{X} \mathbf{w}.$$

- EX 1.6. Using the data provided in Table 1.3, find the linear model that minimises the squared loss.
- EX 1.7. Using the model obtained in the previous exercise, predict the women's winning time at the 2012 and 2016 Olympic games.
- EX 1.8. Using the models for the men's and women's 100 m, find the Olympic games when it is predicted for women to run a faster winning time than men. What are the predicted winning times? Do they seem realistic?

- EX 1.9. Load the data stored in the file `synthdata.mat`. Fit a 4th order polynomial function – $f(x; \mathbf{w}) = w_0 + w_1x + w_2x^2 + w_3x^3 + w_4x^4$ – to this data. What do you notice about w_2 and w_4 ? Use 10-fold cross-validation to choose the polynomial order (between 1 and 4).
- EX 1.10. Derive the optimal least squares parameter value, $\hat{\mathbf{w}}$, for the total training loss:

$$\mathcal{L} = \sum_{n=1}^N (t_n - \mathbf{w}^\top \mathbf{x}_n)^2.$$

How does the expression compare with that derived from the average loss?

- EX 1.11. The following expression is known as the *weighted* average loss:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^N \alpha_n (t_n - \mathbf{w}^\top \mathbf{x}_n)^2$$

where the influence of each data point is controlled by its associated α parameter. Assuming that each α_n is fixed, derive the optimal least squares parameter value $\hat{\mathbf{w}}$.

- EX 1.12. Using K -fold cross-validation, find the value of λ that gives the best predictive performance on the Olympic men's 100 m data for (a) a 1st order polynomial (i.e. the standard linear model) and (b) a 4th order polynomial.

Further reading

- [1] F. Galton. Regression towards mediocrity in hereditary stature. *Anthropological Miscellanea*, 15:246–263, 1886.

The term ‘regression’ was first used in the context of genetics by Francis Galton. This is one of Galton’s original genetics papers on regression from 1886.

- [2] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, second edition, 2009.

This book includes a detailed chapter on least squares techniques which would be a good starting point to explore this area further.

- [3] K. B. Petersen and M. S. Pedersen. The matrix cookbook.

<http://www2.imm.dtu.dk/pubdb/p.php?3274>, October 2008.

An excellent free resource that provides many useful matrix identities. Particularly useful for manipulating, and taking expectations with respect to, multi-variate Gaussian densities.

Chapter 2

Linear Modelling: A Maximum Likelihood Approach

In the previous chapter, we introduced the idea of learning the parameters of a model by defining and minimising a loss function. By the end of this chapter, we will have derived exactly the same equation for the optimal parameter values from a different starting point. In particular, we will explicitly model the **noise** (the errors between the model and the observations) in the data by incorporating a **random variable**. We will demonstrate the considerable advantages of incorporating a noise term into our model. A large section of this chapter (Sections 2.2 to 2.5) is an introduction to random variables and **probability** which can be skipped by readers already familiar with these concepts.

2.1 Errors as noise

In Figure 1.5 we saw the result of minimising the squared loss function to model the Olympics 100 m data with a linear model. The linear model appears to capture an interesting downward trend but is unable to explain each data point perfectly — there are errors between the model and the true values. These errors are highlighted in Figure 2.1.

When building our model, we assumed that there was a linear relationship between years and winning times. This model appeared to capture the general trend in the data whilst ignoring the sometimes large deviation between the model and the observed data. From a modelling perspective, ignoring these errors is hard to defend. If we know they are going to be present we should make an effort to build them into our model.

In this chapter we will see the benefits of explicitly modelling these errors. In particular, it allows us to express the level of uncertainty in our estimate of the model parameters, w — if we change w a bit, do we still have a *good* model? This in turn allows us to express a degree of uncertainty in our predictions — ‘we believe the winning time will be between a and b ’ rather than ‘we believe the winning time will be exactly c ’.

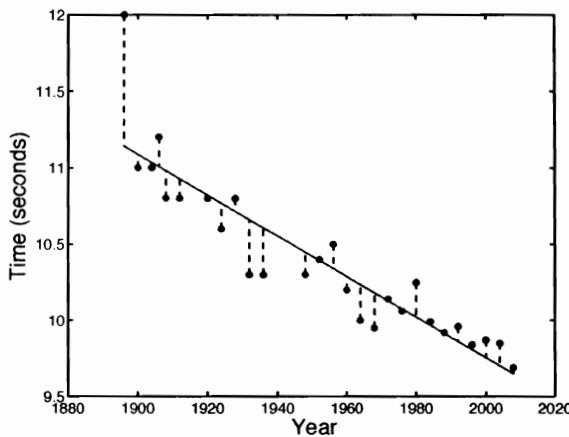


FIGURE 2.1: Linear fit to the Olympics men’s 100 m data with errors highlighted.

2.1.1 Thinking generatively

The process that generated this particular dataset is very complex – we couldn’t even begin to make a near-perfect model of one sprinter and the events surrounding his preparation and performance, let alone several of them *and* all of the other factors. However, it is still useful to think of our modelling problem as a **generative** one: can we build a model that could be used to create (or generate) a dataset that *looks* like ours? Although we are happy to accept that this isn’t in fact how the data were generated, we shall see that this is a useful strategy.

How might we go about generating data from our current model? We have an equation: $f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}$ that, if we plug in the values for \mathbf{w} that we found in the previous chapter, could be used to *generate* a winning time for any particular year. Figure 2.2 shows winning times generated in this way for a number of years between 1920 and 2000. It doesn’t look much like the data in Figure 2.1. To make it more realistic, we need to add some errors. Examining Figure 2.1, we notice a couple of important features of the errors:

1. They are different at each year. Some are positive, some negative and they all have different magnitudes.
2. There does not seem to be any obvious relationship between the size (or direction) of the error and the year. The error does not appear to be a function of x , the Olympics year.

If we had a method for generating a random amount of time (in seconds) that could be either positive or negative and was, on average, roughly the same size as the errors in Figure 2.1, we could generate one such value for each data point we wished to generate and add it to $\mathbf{w}^\top \mathbf{x}$. The tools that we will need to incorporate this variability into our model come from **Statistics**. In the next section we will introduce random variables and some of the ways in which they can be manipulated. Readers already familiar with this can jump straight to Section 2.6.

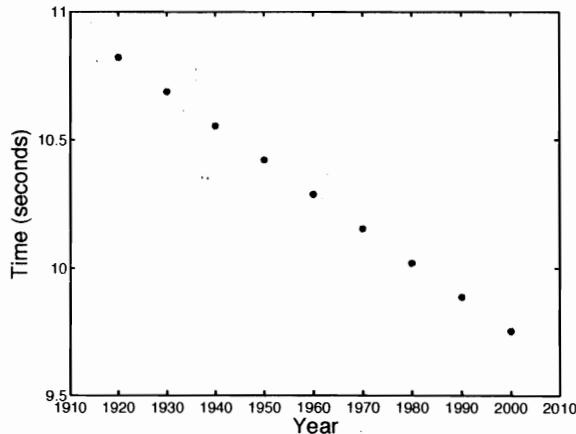


FIGURE 2.2: Dataset generated from a linear model.

2.2 Random variables and probability

Any model we build will be a simplification of the real system that generated the data we observe. This will lead to a discrepancy between the model and reality which the tools presented in this section will help us to model and understand. As we must start with the basics, it may at first seem slightly disconnected from the particular problem of adding errors to our generated 100 m data and being able to express uncertainties in our predictions, but the connection will become clearer as we progress.

2.2.1 Random variables

The equation

$$y = 5x - 2$$

has two variables, x and y . If we were given a value for one (say $y = 8$) we could solve for the other ($x = 2$). *Random* variables are very different. They allow us to assign numerical values to **random events**. For example, I would like to model the outcome of a coin toss. As a starting point, I create a variable called X to which I will assign the value 1 if the coin lands heads and 0 if it lands tails. X is a random variable – the ‘variable’ part describes the fact that it can take a number of different values (in this case, 0 and 1) and the ‘random’ part is so called because we don’t know what value X will take before the coin toss takes place – we couldn’t express the outcome as a function of standard variables (e.g. $y = 5x - 2$). It is a common convention to use upper-case letters to describe random variables and lower-case ones for possible values that the random variable can take.

There are two types of random variable and they must be treated slightly differently. **Discrete** random variables are the easiest to conceptualise as they are used for random events for which we can systematically list (or count) all possible out-

comes. A discrete random variable could be used to, for example, describe a coin toss (possible outcomes are 0 and 1) or the rolling of a die (1, 2, 3, 4, 5 or 6). The collection of possible outcomes is known as the **sample space**.

It might seem that being able to systematically write all of the possible events in order should be true of almost anything. In fact, there are many possible events for which this is not the case. Taking our Olympics 100 m example and assuming that the winning time is going to be between 9 and 10 seconds, we could attempt to systematically write down all possibilities:

$$9, 9.1, 9.2, \dots$$

at which point, we realise that we've missed some out (all the ones between 9 and 9.1 for example) so we start again:

$$9, 9.01, 9.02, \dots, 9.1, \dots$$

But what about all the ones between 9 and 9.01? Starting a third time

$$9, 9.001, 9.002, 9.003, \dots, 9.01, \dots \text{etc.}$$

The possible outcomes of this event cannot be systematically listed (after writing down any two values, someone could point out the missing ones in between). For events like this, we must use **continuous** random variables.

Table 2.1 gives examples of events or quantities that we might wish to model with random variables and whether or not they are discrete or continuous. We will now introduce some important concepts through discrete random variables before extending the ideas to the continuous case.

2.2.2 Probability and distributions

Let Y be a random variable that represents the toss of a coin. If the coin lands heads, $Y = 1$ and if tails, $Y = 0$. To model this event (the coin toss), we need to be able to quantify how likely either outcome is. For discrete random variables, we do this by defining the probabilities of the different outcomes. One intuitive way of thinking about the probability of a particular outcome is to imagine that it represents the proportion of times this outcome would happen if the event were to be repeated

TABLE 2.1: Events we might want to model with random variables.

Process	Discrete or continuous
Toss of a coin	Discrete
Roll of a die	Discrete
Outcome of a 100 m race	Continuous
Failure of a node in a computer network	Discrete
Outcome of a court case	Discrete
Height of a human	Continuous
Mass of a pebble	Continuous
Score in a football match	Discrete
Errors in our 100 m linear regression model	See Exercise EX 2.1.

many times. If a fair (i.e. not biased to land either way) coin were tossed 1000 times, we might expect to see heads roughly half of the time (and tails the rest of time). It would seem sensible to define the probability of seeing a head, which we will denote $P(Y = 1)$, as a half, or 0.5. If the coin doesn't land as a head, it lands as a tail (there are only two options in our sample space) and so the proportion of tails must be one minus the proportion of heads. Therefore, $P(Y = 0) = 1 - P(Y = 1) = 0.5$.

Conceptualising probabilities as the proportion of times a particular outcome would occur if an event were repeated many times is not the only way they can be thought of. It is not always the most natural analogy, particularly for events that can only occur once. It will be sufficient for our needs, but the reader is encouraged to investigate this interesting area further.

From our short discussion on proportions, we can write down two important rules governing probabilities:

- Probabilities must be greater than or equal to 0 (a proportion cannot be negative) and less than or equal to 1.
- The sum of the probabilities of each possible individual outcome must be equal to 1.

$$\text{e.g. for a coin: } P(Y = 1) + P(Y = 0) = 1$$

$$\text{for a die: } P(Y = 1) + P(Y = 2) + \dots + P(Y = 6) = 1$$

The mathematical equivalents of these statements are:

$$0 \leq P(Y = y) \leq 1 \quad (2.1)$$

$$\sum_y P(Y = y) = 1 \quad (2.2)$$

where the lower case y is used, by convention, to represent values that the random variable Y can take. Note that we will often need to write summations over the values that a random variable can take – to keep notation concise, \sum_y will be used to denote a sum over all of the possible values that can be taken by a random variable Y .

$P(Y = y)$ is a scalar value – the probability that the random variable Y has outcome y . This notation can sometimes become unwieldy and so we will sometimes use the following shorthand:

$$P(Y = y) = P(y).$$

The set of all of the possible outcomes (all of the y s) and their probabilities, $P(y)$, is known as a probability **distribution**. It tells us how the total probability (1) is distributed (or shared out) over all possible outcomes.

Often, we can use Equations 2.1 and 2.2 to define probabilities based on some fundamental assumptions. For example, in the coin example, we might assume that the two outcomes are equally likely: $P(Y = 1) = P(Y = 0) = r$. Plugging this into Equation 2.2 and remembering that r must lie between 0 and 1 (Equation 2.1), we can use some algebra to work out the value of r (see Exercise EX 2.2):

$$P(Y = 0) + P(Y = 1) = 1$$

$$2r = 1$$

$$r = \frac{1}{2}.$$

2.2.3 Adding probabilities

Let Y be a random variable for modelling the outcome of rolling a fair die. If we encode our assumption that the die is fair by assuming that all outcomes are equally likely, we know enough from the previous section to compute the probabilities of each possible outcome – 1, 2, 3, 4, 5 or 6. The die is rolled and the result is a 4. If it is rolled again, what is the probability of the result being lower than 4? Maybe we are playing a betting game and want to know whether the odds on offer are acceptable. The outcomes that are lower than 4 are 1, 2 and 3, suggesting that we need to be able to calculate the probability that the die lands 1 *or* 2 *or* 3. If the die were to be rolled many times, we could compute the proportion of times that this was the case. The proportion of times the die lands 1 *plus* the proportion of times the die lands 2 *plus* the proportion of times the die lands 3. This leads us to the following additive law of probability:

$$P(Y < 4) = P(Y = 1) + P(Y = 2) + P(Y = 3).$$

Exactly the same result applies if the outcomes in which I'm interested are not in order. For example, the probability that I roll a 1 or a 6 would be $P(Y = 1) + P(Y = 6)$. It is also not just restricted to individual outcomes. For example, the probability that I don't roll a 4 could be computed as

$$\begin{aligned} P(Y \neq 4) &= P(Y < 4) + P(Y > 4) \\ &= P(Y = 1) + P(Y = 2) + P(Y = 3) + P(Y = 5) + P(Y = 6). \end{aligned}$$

As an aside, it is worth remembering that there is generally more than one way to compute any probability. In this example, it would in fact be easier to make use of Equation 2.2 and compute:

$$\begin{aligned} P(Y \neq 4) + P(Y = 4) &= 1 \\ P(Y \neq 4) &= 1 - P(Y = 4). \end{aligned}$$

2.2.4 Conditional probabilities

Often one event will affect the outcome of another. For example, I toss a coin and then tell you what the result was (you cannot see the coin). There are two events – the first is tossing the coin, the second is me communicating the outcome of the coin toss to you. Let's assume that these two events are represented by two random variables. X is 1 if the coin lands heads and 0 if tails. Y is 1 if I tell you heads, and 0 if I tell you tails. Unless I'm behaving very strangely, the outcome of Y will depend on the outcome of X . We can use **conditional probabilities** to express the probability that Y takes a particular value given that X has taken a particular value. We express this as

$$P(Y = y|X = x); \quad (2.3)$$

which reads as the probability that Y has the outcome y given that X has the outcome x . As for unconditional probabilities, we will also make use of the following shorthand:

$$P(Y = y|X = x) = P(y|x).$$

In our example, if we assume that I always tell the truth, the probability that I say heads if the coin lands heads is 1 (it will always happen)

$$P(Y = 1|X = 1) = 1.$$

Similarly for tails

$$P(Y = 0|X = 0) = 1.$$

Using Equation 2.2 and these probabilities, we can deduce $P(Y = 0|X = 1)$ and $P(Y = 1|X = 0)$:

$$\begin{aligned} P(Y = 0|X = 1) + P(Y = 1|X = 1) &= 1 \\ P(Y = 0|X = 1) = 1 - P(Y = 1|X = 1) &= 0. \\ P(Y = 1|X = 0) + P(Y = 0|X = 0) &= 1 \\ P(Y = 1|X = 0) = 1 - P(Y = 1|X = 1) &= 0. \end{aligned}$$

Things get a bit more interesting if I'm not so truthful. Let's assume that if the coin lands tails, I always tell the truth but the proportion of times I tell the truth if it lands heads is 0.8. This implies that if the coin lands heads I'll say heads with probability 0.8 and tails with probability 0.2. The full list of conditional probabilities under this assumption is

$$\begin{aligned} P(Y = 1|X = 1) &= 0.8 \\ P(Y = 0|X = 1) &= 0.2 \\ P(Y = 1|X = 0) &= 0 \\ P(Y = 0|X = 0) &= 1. \end{aligned}$$

Just as in non-conditional probabilities, Equation 2.2 must be satisfied, i.e. $\sum_y P(Y = y|X = x) = 1$. We can check this for the values just computed:

$$\begin{aligned} \sum_y P(Y = y|X = 1) &= P(Y = 1|X = 1) + P(Y = 0|X = 1) = 0.8 + 0.2 = 1 \\ \sum_y P(Y = y|X = 0) &= P(Y = 1|X = 0) + P(Y = 0|X = 0) = 0 + 1 = 1 \end{aligned}$$

Armed with the conditional probabilities and assuming that $P(X = 1) = P(X = 0) = 0.5$ (i.e. our coin is fair), we might ask 'what is the probability that the coin lands heads *and* I say heads?' This is different from $P(Y = 1|X = 1)$; the conditional distribution assumes that $X = 1$ has already happened and the only uncertainty that remains is what will happen with Y , whereas my question concerns both events. If neither has happened, what is the probability that they will both have a particular outcome? Other interesting quantities that we may want to evaluate are $P(Y = 1)$ and $P(Y = 0)$, the probability that I say heads or tails. To compute any of these, we need to understand probabilities and distributions of more than one variable.

2.2.5 Joint probabilities

Given two (or more) random variables, we may wish to know the probability that they each take a particular value. Continuing our previous coin tossing example, we

might want to know the probability that the coin shows heads *and* I say heads or the probability that the coin shows heads *and* I say tails. These are joint probabilities and are denoted as

$$P(Y = y, X = x) \quad (2.4)$$

(or, in functional form, $p(y, x)$). How we deal with these joint distributions depends on whether or not the random variables are *dependent*. In our example, Y (what I say) depends on X (how the coin lands). This is the case even when I'm not always being truthful – how the coin lands determines how I *decide* what to say. If there is no dependence between the variables (e.g. if two random variables represent different coin tosses, the outcome of one is unlikely to affect the outcome of the other), the **joint probability** can be computed by multiplying the individual probabilities together

$$P(Y = y, X = x) = P(Y = y) \times P(X = x).$$

The probability that Y takes value y *and* X takes value x is equal to the probability that Y takes value y multiplied by the probability that X takes value x . More generally (and here we switch to the functional form $p(y_1, \dots, y_J)$ rather than $P(Y_1 = y_1, \dots, Y_J = y_J)$ for convenience), for a family of J random variables Y_1, \dots, Y_J ,

$$P(y_1, y_2, \dots, y_J) = P(y_1) \times p(y_2) \times \cdots \times P(y_J) = \prod_{j=1}^J P(y_j). \quad (2.5)$$

If the events are dependent, we cannot decompose the joint probability in this manner. However, if we can create conditional distributions, we can decompose the joint probability using the following definitions:

$$P(Y = y, X = x) = P(Y = y|X = x) \times P(X = x) \quad (2.6)$$

or as

$$P(Y = y, X = x) = P(X = x|Y = y) \times P(Y = y). \quad (2.7)$$

So, the probability that the coin lands heads *and* I say heads is

$$P(Y = 1, X = 1) = P(Y = 1|X = 1) \times P(X = 1) = 0.8 \times 0.5 = 0.4$$

or, in other words, if we repeated this many times, the proportion of times that the coin landed heads and I said heads is 0.4. The fact that I occasionally lie when the coin shows heads has reduced the probability that you will hear heads from 0.5 (if I were always honest) to 0.4.

There are four possible combinations of X and Y and hence four possible outcomes of the event. Equation 2.2 tells us that if we sum the probabilities of all four of these events we should get 1:

$$\sum_{x,y} P(X = x, Y = y) = 1. \quad (2.8)$$

(Note that $\sum_{x,y}$ corresponds to a summation over all possible combinations of x and y). We can test this by working them all out from Equation 2.6. We already know $P(X = 1, Y = 1) = 0.4$. The others are

$$\begin{aligned} P(Y = 0, X = 1) &= P(Y = 0|X = 1)P(X = 1) = 0.2 \times 0.5 = 0.1 \\ P(Y = 1, X = 0) &= P(Y = 1|X = 0)P(X = 0) = 0 \times 0.5 = 0 \\ P(Y = 0, X = 0) &= P(Y = 0|X = 0)P(X = 0) = 1 \times 0.5 = 0.5. \end{aligned}$$

Adding these together gives $0.4 + 0.1 + 0 + 0.5 = 1$ as required.

Before we move on, we will quickly consider these three values. The first (0.1) gives the probability that I say tails and the coin lands heads. This has increased from the truthful case (it would be zero if I always told the truth) because I sometimes lie if the coin is heads. The second (0) is the probability that I say heads when the coin is actually tails. This is 0 because I never lie if the coin is tails. The final value is the probability that I say tails and the coin lands tails. This is 0.5 – the coin lands tails half the time and if it does, I always tell the truth.

2.2.6 Marginalisation

If you recorded the proportion of times I said heads or tails, you would in effect be computing $P(Y = 1)$ and $P(Y = 0)$. These expressions do not involve X – they just refer to what I say. $P(Y = y)$ can be obtained by **marginalising** out X from the joint distribution $P(Y = y, X = x)$. This is done by summing the joint probabilities over all possible values of X :

$$P(Y = y) = \sum_x P(Y = y, X = x). \quad (2.9)$$

In our coin example X can take one of two values, so this summation would become

$$P(Y = y) = P(Y = y, X = 0) + P(Y = y, X = 1).$$

In general, for joint probabilities of J random variables, to get $P(Y_j = y_j)$ the marginal distribution of one of them is given by:

$$P(Y_j = y_j) = P(y_j) = \sum_{y_1, \dots, y_{j-1}, y_{j+1}, \dots, y_J} P(y_1, \dots, y_J). \quad (2.10)$$

The summation in this expression looks a bit strange. It is summing over all combinations of the remaining $J - 1$ variables (y_j is missing). For example, if $J = 3$ and each variable can take only the values 0 or 1, to compute $P(Y_1 = y_1) = p(y_1)$ would require summation over four different combinations of y_2 and y_3 ,

y_2	y_3
0	0
0	1
1	0
1	1

If $J = 4$, this increases to 8

y_2	y_3	y_4
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

In general, for binary variables, the number of combinations will be 2^{J-1} , which rapidly increases with J . If our random variables have more than two outcomes it gets even worse (e.g. 6^{J-1} for a die). Marginalisation is important in some probabilistic areas of Machine Learning and can be very challenging, inspiring approximation methods such as those that we shall see in Chapter 4.

Returning to our coin example, $P(Y = 1)$ is

$$\begin{aligned} P(Y = 1) &= \sum_x P(Y = 1, X = x) \\ &= P(Y = 1, X = 0) + P(Y = 1, X = 1) \\ &= 0 + 0.4 = 0.4 \end{aligned}$$

and $P(Y = 0)$ is

$$\begin{aligned} P(Y = 0) &= \sum_x P(Y = 0, X = x) \\ &= P(Y = 0, X = 0) + P(Y = 0, X = 1) \\ &= 0.5 + 0.1 = 0.6. \end{aligned}$$

We could also have computed $P(Y = 0)$ by using the value for $P(Y = 1)$ and Equation 2.2. These probabilities tell us the proportion of times I say heads and tails. They are different from the proportion of times that the coin lands heads or tails ($P(X = 1) = P(X = 0) = 0.5$). This discrepancy is due to the uncertainty in my communication of the results – in the context of this chapter, I am effectively a source of noise or errors. A further example of conditional probabilities and marginalisation is provided in Comment 2.1.

Comment 2.1 – Conditional probabilities and marginalisation – an example: Let's assume that we have a fair coin and two dice (one of which is a little unusual). We will generate a coin toss (X) and a dice roll (Y) using the following procedure. First, toss the coin. If it gives heads, roll die 1. If it gives tails, roll die 2. Die 1 and die 2 are different, with probabilities defined in the following table:

$$\begin{array}{ccccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \text{Die 1} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ \text{Die 2} & \frac{1}{12} & \frac{1}{6} & \frac{1}{4} & \frac{1}{4} & \frac{1}{6} & \frac{1}{12} \end{array} = P(y|X = H) \\ = P(y|X = T)$$

So, the probability of rolling, say, a 3 is $1/6$ with die 1 and $1/4$ with die 2. As we roll die 1 if our coin showed heads and die 2 if tails, we have the following conditional distributions:

$$P(y|X = H), \quad P(y|X = T),$$

i.e. the distribution over Y depends on the outcome of X . The joint distribution is given as (Equation 2.6):

$$p(y, x) = p(y|x)p(x).$$

We can use this to compute the probability of rolling a 3 *and* a head:

$$P(Y = 3, x = H) = P(Y = 3|X = H)P(X = H) = \frac{1}{6} \times \frac{1}{2} = \frac{1}{12}.$$

Alternatively, a 3 *and* a tail:

$$P(Y = 3, X = T) = P(Y = 3|X = T)P(X = T) = \frac{1}{4} \times \frac{1}{2} = \frac{1}{8}.$$

Perhaps more interestingly, we can compute the marginal distribution for Y . From our definition (Equation 2.9):

$$P(y) = \sum_x P(y, x) = \sum_x P(y|x)p(x).$$

Therefore, the probability of rolling a 3 is:

$$\begin{aligned} P(Y = 3) &= \sum_x P(Y = 3|x)p(x) \\ &= P(Y = 3|X = H)P(X = H) + P(Y = 3|X = T)P(X = T) \\ &= \frac{1}{6} \times \frac{1}{2} + \frac{1}{4} \times \frac{1}{2} = \frac{5}{24}. \end{aligned}$$

2.2.7 Aside – Bayes' rule

Although we won't need it in this chapter, it is worth introducing Bayes'¹ rule as it will feature heavily from Chapter 3 onwards. The left hand sides of Equations 2.6

¹Named after the Reverend Thomas Bayes, a British mathematician and Presbyterian minister, who first proposed this reversing of conditional probabilities.

and 2.7 are identical, so we can also equate the right hand sides

$$P(Y = y|X = x)P(X = x) = P(X = x|Y = y)P(Y = y).$$

Rearranging, we can get an expression for the probability of X conditioned on a particular value of Y ($P(X = x|Y = y)$) that depends on the probability of Y conditioned on a particular value of X ($P(Y = y|X = x)$), which is known as Bayes' rule:

$$P(X = x|Y = y) = \frac{P(Y = y|X = x)P(X = x)}{P(Y = y)}. \quad (2.11)$$

In our example, this is the probability that the coin landed in a particular way given (or conditioned on) what I said. This is likely to be of interest to you if you want to make predictions about how the coin actually landed. Substituting our numerical values, we can work out $P(X = 1|Y = 1)$

$$P(X = 1|Y = 1) = \frac{P(Y = 1|X = 1)P(X = 1)}{P(Y = 1)} = \frac{0.8 \times 0.5}{0.4} = 1,$$

from which we can also deduce that $P(X = 0|Y = 1) = 0$ (Equation 2.2 again). Similarly, we can compute $P(X = 0|Y = 0)$

$$P(X = 0|Y = 0) = \frac{P(Y = 0|X = 0)P(X = 0)}{P(Y = 0)} = \frac{1 \times 0.5}{0.6} = 0.83,$$

from which we can deduce that $P(X = 1|Y = 0) = 0.17$.

The first two values give the probabilities of the true coin toss if I say heads (i.e. $Y = 1$) and the second two the true probabilities if I say tails ($Y = 0$). $P(X = 1|Y = 1) = 1$ tells us that my saying heads must mean that heads was the true outcome of the coin toss. $P(X = 0|Y = 0) = 0.83$ tells us that if tails is heard, it is more likely that the coin was tails (probability of 0.83) than heads (probability of 0.17). Reversing the conditioning in this way is very useful when building models and is something that we shall return to in Chapter 3 and beyond.

2.2.8 Expectations

When dealing with random variables, it is useful to summarise a distribution with a value or values that encapsulate its characteristics. An obvious example is the mean value – the average value that we expect the random variable to take. The mean is an example of an **expectation**. An expectation tells us what value we would expect some function $f(X)$ of a random variable X to take and is defined (for discrete random variables) as:

$$\mathbf{E}_{P(x)} \{f(X)\} = \sum_x f(x)P(x). \quad (2.12)$$

For example, if we're interested in the expected value of X (the mean), $f(X) = X$ and the expression becomes:

$$\mathbf{E}_{P(x)} \{X\} = \sum_x xP(x).$$

For a fair die ($P(x) = 1/6$), the expected value of X would be

$$\mathbf{E}_{P(x)}\{X\} = \sum_x x \frac{1}{6} = \frac{1}{6} + \frac{2}{6} + \dots + \frac{6}{6} = \frac{21}{6} = 3.5.$$

Notice from this example that the expected value doesn't have to be one of the values that the random variable can take (we can never roll 3.5).

Expected values of other functions are computed in exactly the same manner. For example, the expected value of $f(X) = X^2$ is:

$$\mathbf{E}_{P(x)}\{X^2\} = \sum_x x^2 \frac{1}{6} = \frac{1}{6} + \frac{4}{6} + \dots + \frac{36}{6} = \frac{91}{6}.$$

It is important to realise that the expected value of a function of X is not in general the function evaluated at the expected value of X . Mathematically, $\mathbf{E}_{P(x)}\{f(X)\}$ does not necessarily equal $f(\mathbf{E}_{P(x)}\{X\})$. As an example, we've just computed $\mathbf{E}_{P(x)}\{X^2\} = 91/6$, which is not equal to $(\mathbf{E}_{P(x)}\{X\})^2 = (21/6)^2$. One situation where the two are equal is when the function is just a constant multiplied by X . In this case, doing a little algebra allows us to show that the two are equivalent:

$$\begin{aligned} f(X) &= aX \\ \mathbf{E}_{P(x)}\{f(X)\} &= \sum_x axP(x) \\ &= a \sum_x xP(x) \\ &= a\mathbf{E}_{P(x)}\{X\} \\ &= f(\mathbf{E}_{P(x)}\{X\}). \end{aligned}$$

Another important case is when the function is simply a constant. In this case, the expectation disappears due to the fact that the distribution has to sum to 1 over all possible outcomes:

$$\begin{aligned} f(X) &= a \\ \mathbf{E}_{P(x)}\{f(X)\} &= \sum_x aP(x) \\ &= a \sum_x P(x) \\ &= a. \end{aligned}$$

A final special case that will prove useful is that the expectation of a sum of different functions is equal to a sum of the individual expectations:

$$\begin{aligned} \mathbf{E}_{P(x)}\{f(X) + g(X)\} &= \sum_x (f(x) + g(x))P(x) \\ &= \sum_x f(x)P(x) + \sum_x g(x)P(x) \\ &= \mathbf{E}_{P(x)}\{f(X)\} + \mathbf{E}_{P(x)}\{g(X)\}. \end{aligned}$$

The two most common expectations that we will come across are the mean ($\mathbf{E}_{P(x)}\{X\}$ as defined above) and the **variance**. Variance is a measure of how variable the random variable is and is defined as the expected squared deviation from the mean:

$$\text{var}\{X\} = \mathbf{E}_{P(x)} \{(X - \mathbf{E}_{P(x)}\{x\})^2\} \quad (2.13)$$

Multiplying out the bracket gives us the following convenient expression for the variance of a random variable:

$$\begin{aligned} \text{var}\{X\} &= \mathbf{E}_{P(x)} \{(X - \mathbf{E}_{P(x)}\{x\})^2\} \\ &= \mathbf{E}_{P(x)} \{X^2 - 2X\mathbf{E}_{P(x)}\{X\} + \mathbf{E}_{P(x)}\{x\}^2\} \\ &= \mathbf{E}_{P(x)} \{X^2\} - 2\mathbf{E}_{P(x)}\{X\}\mathbf{E}_{P(x)}\{X\} + \mathbf{E}_{P(x)}\{X\}^2. \end{aligned}$$

To get from the second to the third line, we have used the fact that $\mathbf{E}_{P(x)}\{\mathbf{E}_{P(x)}\{f(X)\}\} = \mathbf{E}_{P(x)}\{f(X)\}$. The result of $\mathbf{E}_{P(x)}\{f(X)\}$ is a constant (all X terms are removed by the expectation). The outer expectation is the expected value of a constant, which we have already shown is equal to the constant. Collecting together the $\mathbf{E}_{P(x)}\{X\}^2$ terms gives:

$$\text{var}\{X\} = \mathbf{E}_{P(x)}\{X^2\} - \mathbf{E}_{P(x)}\{X\}^2 \quad (2.14)$$

Random variables with high variance would, on average, take values further away from their mean than random variables with low variance.

Comment 2.2 – Vector random variables: It will often be necessary to define probability distributions over vectors. This is nothing more than a shorthand way of defining large joint distributions. For example, the values that could be taken on by random variables X_1, X_2, \dots, X_N can be expressed as the vector $\mathbf{x} = [x_1, x_2, \dots, x_N]^\top$. Using this shorthand:

$$p(\mathbf{x}) = p(x_1, x_2, \dots, x_N) = P(X_1 = x_1, X_2 = x_2, \dots, X_N = x_N).$$

Even though \mathbf{x} is a vector, $p(\mathbf{x})$ is a scalar quantity, just as $P(X_1 = x_1, X_2 = x_2, \dots, X_N = x_N)$ is.

Expectations are computed for vector random variables (see Comment 2.2) in exactly the same way. For a random variable X that can take vector values \mathbf{x} , expectations are defined as

$$\mathbf{E}_{P(\mathbf{x})}\{f(\mathbf{x})\} = \sum_{\mathbf{x}} f(\mathbf{x})P(\mathbf{x})$$

where the sum is over all possible values of the vector \mathbf{x} . Therefore, the mean vector is defined as

$$\mathbf{E}_{P(\mathbf{x})}\{\mathbf{x}\} = \sum_{\mathbf{x}} \mathbf{x}P(\mathbf{x}).$$

When dealing with vectors, the concept of variance is generalised to a **covariance** matrix. This is defined as

$$\text{cov}\{\mathbf{x}\} = \mathbf{E}_{P(\mathbf{x})} \left\{ (\mathbf{x} - \mathbf{E}_{P(\mathbf{x})}\{\mathbf{x}\})(\mathbf{x} - \mathbf{E}_{P(\mathbf{x})}\{\mathbf{x}\})^\top \right\} \quad (2.15)$$

If \mathbf{x} is a vector of length D then $\text{cov}\{\mathbf{x}\}$ is a $D \times D$ matrix. The diagonal elements correspond to the variance of the individual elements of \mathbf{x} whilst the off-diagonal elements tell us to what extent different elements of \mathbf{x} co-vary, that is, how dependent are they on one another. A high positive value between, say, elements x_d and x_e suggests that if x_d increases, so does x_e . A high negative value suggests that they are related but move in opposite directions (x_d increases whilst x_e decreases) and a value of (or close to) zero suggests that there is no relationship between them (they are independent). We give some examples of covariance matrices and the associated densities in Section 2.5.4. Just as for variance, the covariance expression can be manipulated into a more convenient form as follows:

$$\begin{aligned}\text{cov}\{\mathbf{x}\} &= \mathbf{E}_{P(\mathbf{x})} \left\{ (\mathbf{x} - \mathbf{E}_{P(\mathbf{x})} \{\mathbf{x}\}) (\mathbf{x} - \mathbf{E}_{P(\mathbf{x})} \{\mathbf{x}\})^\top \right\} \\ &= \mathbf{E}_{P(\mathbf{x})} \left\{ \mathbf{x}\mathbf{x}^\top - 2\mathbf{x}\mathbf{E}_{P(\mathbf{x})} \{\mathbf{x}\}^\top + \mathbf{E}_{P(\mathbf{x})} \{\mathbf{x}\} \mathbf{E}_{P(\mathbf{x})} \{\mathbf{x}\}^\top \right\}\end{aligned}$$

Re-arranging this expression results in:

$$\text{cov}\{\mathbf{x}\} = \mathbf{E}_{P(\mathbf{x})} \left\{ \mathbf{x}\mathbf{x}^\top \right\} - \mathbf{E}_{P(\mathbf{x})} \{\mathbf{x}\} \mathbf{E}_{P(\mathbf{x})} \{\mathbf{x}\}^\top \quad (2.16)$$

2.3 Popular discrete distributions

In all of our examples thus far, we have worked with random variables for which we can list the probabilities of each possible outcome. This is useful for explanatory purposes but rapidly becomes infeasible as the number of possible outcomes increases. In reality, we will often work with well known families of distributions. Each family is suitable for particular types of events and in general these distributions have parameters that can be tuned to change their characteristics. In this section we will describe some common discrete distributions that you are likely to come across in machine learning.

2.3.1 Bernoulli distribution

We have already come across the Bernoulli distribution several times without realising it. It is used for events like a coin toss that have two possible outcomes. For a random variable X that can take two values, 0 or 1 (a binary random variable), where the probability that it takes the value 1 is defined as q , the Bernoulli distribution is:

$$P(X = x) = q^x(1 - q)^{1-x}. \quad (2.17)$$

The Bernoulli distribution is also a special case of the **binomial** distribution (see below) when $N = 1$.

2.3.2 Binomial distribution

The binomial distribution extends the Bernoulli distribution to define the probability of observing a certain number of heads in a total of N tosses. More generally,

we might think of events that have two outcomes (success or failure). If we have N such events, the binomial random variable Y can take values from 0 (no successes) to N (N successes). The probability of observing a particular number of successes is given by:

$$P(Y = y) = P(y) = \binom{N}{y} q^y (1 - q)^{N-y}. \quad (2.18)$$

The second part of this expression looks very similar to the Bernoulli expression we have already seen. In fact, if we define the N binary outcomes as x_1, \dots, x_N , the second part of the binomial expression is the product of the N binomial probabilities:

$$\begin{aligned} \prod_{n=1}^N q^{x_n} (1 - q)^{1-x_n} &= q^{\sum_n x_n} (1 - q)^{N - \sum_n x_n} \\ &= q^y (1 - q)^{N-y}, \end{aligned}$$

where $y = \sum_n x_n$: the number of successes (a success corresponds to $x_n = 1$). The first part of the binomial expression is required because there is potentially more than one set of x_1, x_2, \dots, x_N that corresponds to, say, $y = 3$. $q^y (1 - q)^{N-y}$ gives us the probability of just one of these sets. Summing over all possible sets is equivalent to multiplying by the number of such sets, given by the combinations function, $\binom{N}{y}$ (read as N choose y – see Comment 2.3 for details). Figure 2.3 shows an example of the distribution function when $N = 50$ and $q = 0.7$.

2.3.3 Multinomial distribution

Our previous two examples have been distributions over scalar random variables – we will now look at a distribution that assigns probabilities to vectors of discrete

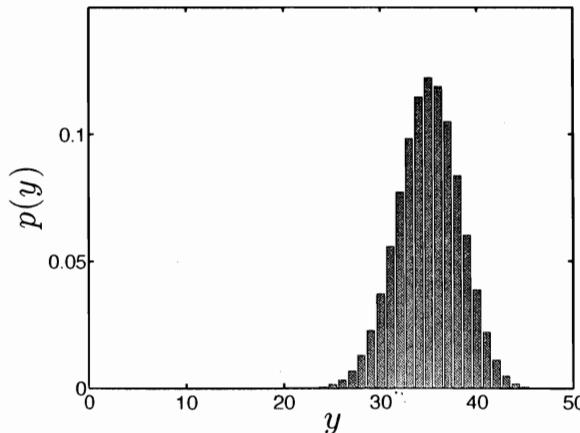


FIGURE 2.3: An example of the probability distribution function for a binomial random variable when $N = 50$ and $q = 0.7$ (see Equation 2.18).

values. The fundamental ideas are exactly the same – the distribution assigns a probability to every possible vector and the sum of these probabilities must equal one. As a motivation for vector random variables, imagine you were building a machine that would produce random documents of N words and you wanted to define a distribution over these documents. This isn't as foolish as it might sound – Machine Learning techniques are often used to analyse text data by defining distributions over documents in just this manner. One way of representing a document would be with a vector of word counts. Assuming J possible words in our vocabulary, the vector would be of length J and the j th element would hold the number of times the j th word appears in the document. The **multinomial** distribution allows us to define a distribution over such vectors. Let Y be a random variable that represents a document. An instance of this random variable is a vector of word counts $\mathbf{y} = [y_1, \dots, y_J]^\top$; the multinomial distribution defines the probability of \mathbf{y} as

$$P(Y = \mathbf{y}) = P(\mathbf{y}) = \frac{N!}{\prod_j y_j!} \prod_j q_j^{y_j} \quad (2.19)$$

where q_j are the parameters of the multinomial distribution and represent the probabilities of the individual words ($\sum_j q_j = 1$).

Comment 2.3 – Combinations: N choose y , written as

$$\binom{N}{y}$$

is mathematical shorthand for the number of ways in which y distinct objects can be chosen from a set of N objects. For example, $\binom{4}{1}$ would be 4 – there are 4 ways I can choose one object from four objects – object 1 on its own, object 2 on its own, object 3 on its own or object 4 on its own. $\binom{4}{2}$ is 6 – the possible choices are 1 and 2, 1 and 3, 1 and 4, 2 and 3, 2 and 4 or 3 and 4. In general,

$$\binom{N}{y} = \frac{N!}{y!(N-y)!}$$

where $N!$ (read N factorial) is

$$\prod_{i=1}^N i = N \times (N-1) \times (N-2) \times \dots \times 1.$$

2.4 Continuous random variables – density functions

We saw at the start of this section that we are unable to systematically write down all possible outcomes of a continuous random variable. Unfortunately, this precludes us from assigning probabilities to particular values. To overcome this we work with the probabilities of the outcome falling within some range or interval. For

example, given a continuous random variable X that can take on any value between minus infinity and infinity it makes sense to try and work out

$$P(x_1 \leq X \leq x_2)$$

but not

$$P(X = x).$$

When working with continuous random variables, we need a continuous analogue to the probability distribution (recall that this, for a discrete random variable, was the set of outcomes (x) and the probabilities of each outcome, expressed as a function of x , $p(x)$). This is provided by a **probability density function** (pdf), also denoted $p(x)$. To compute the probability that X lies in a particular range, we compute the definite integral (see Comment 2.4) of $p(x)$ with respect to x over this range

$$P(x_1 \leq X \leq x_2) = \int_{x_1}^{x_2} p(x)dx.$$

If our random variable may only take values in the range $x_1 \leq X \leq x_2$, it stands to reason that the probability that it lies in this range must be 1. This leads us to the continuous equivalent of Equation 2.2:

$$\int_{x_1}^{x_2} p(x)dx = 1 \text{ where } x_1 \leq X \leq x_2. \quad (2.20)$$

Equation 2.1 also has a continuous equivalent,

$$p(x) \geq 0 \quad (2.21)$$

that tells us that a pdf can never be negative. Note that there is no upper bound on the value of the pdf – it is not a probability and so can (and often will) be higher than 1 for a particular value of x .

Comment 2.4 – Definite Integrals: When differentiating a function including a constant term, the term disappears, e.g.

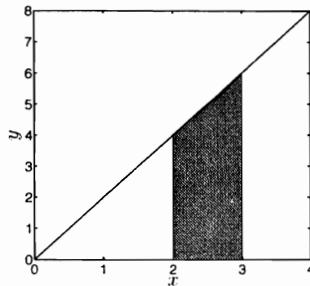
$$\frac{d}{dx}(x^2 + 3) = 2x.$$

Hence, when we are integrating a function, we have to admit the possibility that there might be a constant term

$$\int 2x \, dx = x^2 + C.$$

This is called an indefinite integral as we don't know the value of C .

Often we will be interested in using integration to compute the area under a curve. For example, here we are interested in computing the area under the curve $y = 2x$ between $x = 2$ and $x = 3$, as shown in the plot on the right. This is calculated as



$$\int_2^3 2x \, dx = [x^2 + C]_2^3$$

where the $[\cdot]_a^b$ means take the value of the object inside the brackets when $x = a$ away from the value when $x = b$. In this case, this suggests

$$(3^2 + C) - (2^2 + C) = 9 - 4 + C - C = 5.$$

This is a definite integral – the constants cancel out and the answer is exact.

Joint and conditional continuous densities: Just as with the discrete case, we can define joint probability density functions over several continuous random variables. For example, $p(x, y)$ is the joint density of two random variables X and Y and $p(\mathbf{w})$ is the density of a vector, \mathbf{w} , which could be thought of as the joint density of $p(w_0, w_1, \dots)$ – random variables representing each element in the vector. Although we cannot compute $P(X = x, Y = y)$, we can compute

$$P(x_1 \leq X \leq x_2, y_1 \leq Y \leq y_2) = \int_{x=x_1}^{x_2} \int_{y=y_1}^{y_2} p(x, y) \, dx \, dy.$$

The same applies for conditional distributions, although the conditioning is done on an exact value (as this event is assumed to have happened). For example, we would compute

$$P(x_1 \leq X \leq x_2 | Y = y) = \int_{x=x_1}^{x_2} p(x|Y=y) \, dx.$$

Often we will use the shorthand $p(x|y)$ to describe the density function of X given that $Y = y$.

Marginalisation: You may have already guessed that to marginalise over a continuous random variable, we replace the summation from the discrete case with an integral. For example, the pdf $p(y)$ can be computed from $p(y, x)$ as follows:

$$p(y) = \int_{x=x_1}^{x_2} p(y, x) dx$$

where $x_1 \leq X \leq x_2$ describes the sample space of X .

Expectations: Expectations with respect to continuous random variables are performed by integrating over the range of values that the random variable can take:

$$\mathbf{E}_{p(x)} \{f(x)\} = \int f(x)p(x)dx. \quad (2.22)$$

All of the expressions derived in Section 2.2.8 are identical in the continuous case.

In many practical scenarios, we will not be able to perform this integral – we may not know the exact form of $p(x)$ or it might simply be impossible to integrate. However, if we can generate samples from $p(x)$, it can be approximated by

$$\mathbf{E}_{p(x)} \{f(x)\} \approx \frac{1}{S} \sum_{s=1}^S f(x_s) \quad (2.23)$$

where x_s is one of the S samples from $p(x)$. This is an example of a **Monte Carlo** approximation to an integral which we will see a lot more of in subsequent chapters.

2.5 Popular continuous density functions

Just as for the discrete case, there are several common families of continuous density functions that we will often come across. In this section, we will describe three of them.

2.5.1 The uniform density function

The simplest continuous density function is the uniform density function. The uniform density function, $p(y) = \mathcal{U}(a, b)$, is constant between a and b and zero elsewhere

$$p(y) = \begin{cases} r & \text{for } a \leq y \leq b \\ 0 & \text{otherwise} \end{cases} \quad (2.24)$$

An example where $a = 3$ and $b = 8$ can be seen in Figure 2.4. We can compute the value of r for any values of a and b by remembering that the integral of the pdf over the sample space must be equal to 1 by definition. In this case,

$$\begin{aligned} P(a \leq Y \leq b) = 1 &= \int_{y=a}^b p(y) dy = \int_{y=a}^b r dy \\ &= [yr]_a^b = rb - ra = r(b - a) \\ r &= \frac{1}{b - a}. \end{aligned}$$

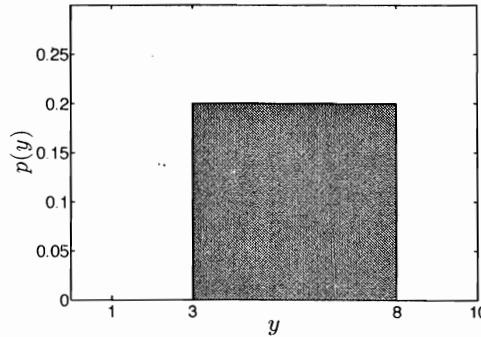


FIGURE 2.4: An example of the uniform pdf.

This is quite intuitive – it is the total probability available – 1 – divided by the length of the interval in which the variables must lie ($b - a$). We can also easily define multi-dimensional uniform random variables. For example, if $\mathbf{y} = [y_1, y_2]^\top$,

$$p(\mathbf{y}) = \begin{cases} r & \text{for } a \leq y_1 \leq b \text{ and } c \leq y_2 \leq d \\ 0 & \text{otherwise} \end{cases}$$

and we can compute r in just the same way:

$$\begin{aligned} P(a \leq y_1 \leq b, c \leq y_2 \leq d) &= 1 = \int_{y_1=a}^b \int_{y_2=c}^d r \, dy_1 \, dy_2 \\ &= \int_{y_1=a}^b [ry_2]_c^d \, dy_1 = \int_{y_1=a}^b r(d - c) \, dy_1 \\ &= [r(d - c)y_1]_a^b = r(d - c)(b - a) \\ r &= \frac{1}{(d - c)(b - a)}. \end{aligned}$$

Again, this is intuitive – it is the total probability – 1 – divided by the area of the interval in which the variables must lie $(d - c)(b - a)$.

As an aside, Equation 2.23 shows how we can approximate expectations by taking samples (realisations of the random variable) from the appropriate distribution. We will demonstrate this approach by computing the expected value of y^2 analytically and via sampling. The analytical result is given by:

$$\begin{aligned} \mathbf{E}_{p(y)} \{y^2\} &= \int_{y=a}^b y^2 p(y) \, dy = \int_{y=a}^b \frac{y^2}{b-a} \, dy \\ &= \left[\frac{y^3}{3(b-a)} \right]_a^b = \frac{b^3 - a^3}{3(b-a)}. \end{aligned}$$

Substituting $a = 0, b = 1$ gives:

$$\mathbf{E}_{p(y)} \{y^2\} = \frac{1}{3}.$$

To compute the sample based approximation, we need to be able to draw samples

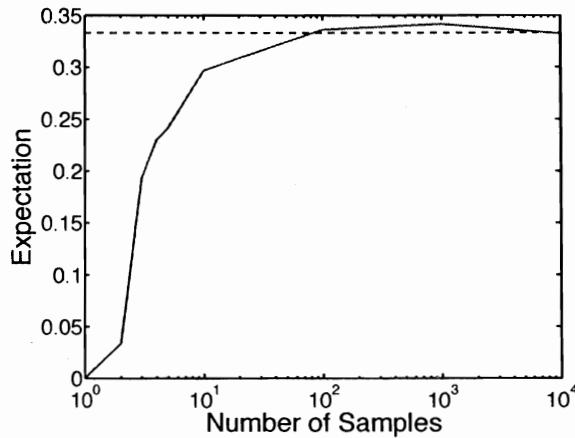


FIGURE 2.5: Effect of increasing the number of samples on the approximation to the expectation given in Equation 2.25 where $p(y) = \mathcal{U}(0, 1)$. The dashed line is the true value of $1/3$. Note the log scale on the x -axis.

from $\mathcal{U}(0, 1)$. In Matlab, the command `rand` generates samples from this distribution. If we generate S samples, y_s , we can approximate the expectation as:

$$\mathbf{E}_{p(y)} \{y^2\} = \frac{1}{S} \sum_{s=1}^S y_s^2. \quad (2.25)$$

Figure 2.5 shows how this approximation improves as we increase the number of samples from 1 to 10^4 . The true value, $\frac{1}{3}$, is shown as the dashed line (MATLAB script: `approx_expected_value.m`). After only 100 samples, the approximation is reasonably good. Approximating expectations with samples will be used extensively in later chapters (see Exercise EX 2.4).

2.5.2 The beta density function

The beta density function can be used for continuous random variables that are restricted to between 0 and 1. The beta density function is defined as:

$$p(r) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1} (1-r)^{\beta-1} \quad (2.26)$$

where α and β are parameters that control the shape of the density function; both must be positive. $\Gamma(z)$ is known as the gamma function and we will omit a discussion here except to say that it can be computed in MATLAB using the inbuilt function `gamma`. Figure 2.6 shows the beta pdfs corresponding to three different sets of parameters. We will use the beta density function considerably in Chapter 3 and so will leave more discussion until then.

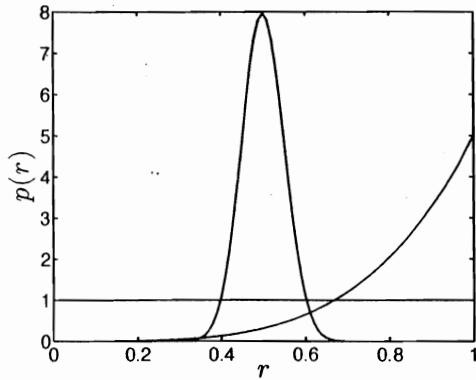


FIGURE 2.6: Examples of beta pdfs with three different pairs of parameters.

2.5.3 The Gaussian density function

Gaussian random variables are used in many continuous applications. One reason is the ease with which the Gaussian pdf can be manipulated in certain useful situations. The Gaussian distribution is defined over a sample space that includes all real numbers (i.e. all numbers between $-\infty$ and ∞) and has a pdf for a random variable Y defined as:

$$p(y|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2\sigma^2}(y - \mu)^2\right\} \quad (2.27)$$

and is characterised by two variables – the mean (μ) and variance (σ^2). Figure 2.7 shows three Gaussian pdfs with different μ, σ^2 values. The highest value of the pdf is obtained when $y = \mu$ and the density is symmetric about this point. The width of the density is controlled by σ^2 – the higher the value, the wider the density. If we used the leftmost Gaussian in Figure 2.7 to generate instances of a random variable, we would only expect values from a small range around -2 . For the rightmost Gaussian,

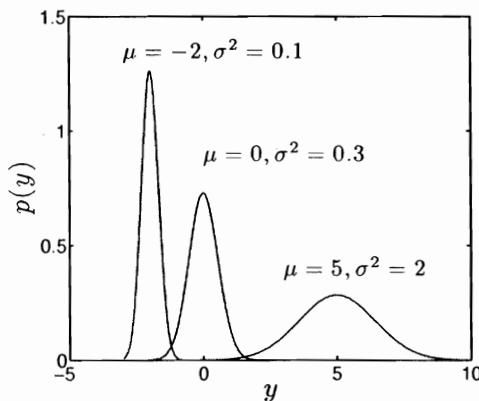


FIGURE 2.7: Three Gaussian pdfs with different means and variances.

we would anticipate values from quite a large range around 5. A common shorthand for the Gaussian pdf is $\mathcal{N}(\mu, \sigma^2)$. Therefore, if Y has a Gaussian pdf, we could write

$$p(y|\mu, \sigma^2) = \mathcal{N}(y|\mu, \sigma^2)$$

which reads as ‘the density function for the random variable Y is normal with mean μ and variance σ^2 ’ (Gaussian and normal are used interchangeably).

2.5.4 Multivariate Gaussian

The Gaussian distribution can also be generalised to define a density function over continuous vectors. This multivariate Gaussian density for a vector $\mathbf{x} = [x_1, \dots, x_D]^\top$ is something we will use a great deal in subsequent chapters. The density function is defined as

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\} \quad (2.28)$$

where the mean $\boldsymbol{\mu}$ is now a vector (of the same size as \mathbf{x}), the d th element of which tells us the mean value of x_d and the variance has become a $D \times D$ covariance matrix. A graphical example is perhaps the best way of getting a feel for this density and the effects of the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. The first example is shown in the top line of Figure 2.8. In this example, the parameters are:

$$\boldsymbol{\mu} = [2, 1]^\top, \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

This is a special case of the multi-variate Gaussian where the two variables (say x_1 and x_2) are independent. To show this, we note that $\boldsymbol{\Sigma} = \mathbf{I}$. So,

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{D/2} |\mathbf{I}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{I}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \right\}.$$

Now, $\mathbf{I}^{-1} = \mathbf{I}$ (see Comment 1.10) allowing us to manipulate this expression to obtain a product over univariate Gaussian pdfs. Starting with the expression above (having swapped the \mathbf{I}^{-1} for \mathbf{I}), we can convert the matrix product inside the exponential into a sum over the D different elements (see Exercise EX 2.5):

$$\begin{aligned} p(\mathbf{x}) &= \frac{1}{(2\pi)^{D/2} |\mathbf{I}|^{1/2}} \exp \left\{ -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^\top \mathbf{I} (\mathbf{x} - \boldsymbol{\mu}) \right\} \\ &= \frac{1}{(2\pi)^{D/2} |\mathbf{I}|^{1/2}} \exp \left\{ -\frac{1}{2} \sum_{d=1}^D (x_d - \mu_d)^2 \right\} \end{aligned}$$

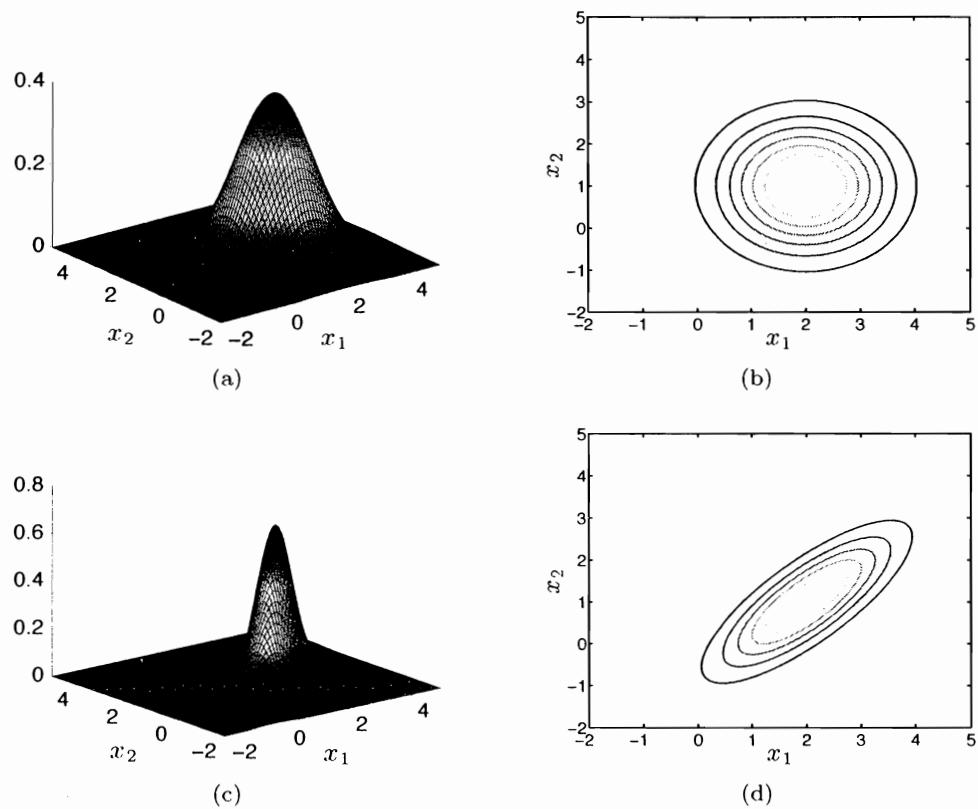


FIGURE 2.8: Example surface (left) and contour (right) plots for two different two-dimensional Gaussian pdfs.

Comment 2.5 – Matrix determinant: The determinant of a square matrix, denoted $|\mathbf{A}|$, for matrix \mathbf{A} is a useful quantity, especially when dealing with multivariate Gaussians. For large matrices, it is too cumbersome to calculate by hand but it can be done for small matrices. For example, for a 2×2 matrix

$$\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad |\mathbf{A}| = ad - bc,$$

but for anything bigger than this it is safest to resort to a computer unless the matrix has a special structure. One special matrix that we will see a lot of is a square matrix that only has diagonal elements (all off-diagonal elements are zero). In this case, the determinant is simply the product of these elements. For example,

$$\mathbf{A} = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{DD} \end{bmatrix}, \quad |\mathbf{A}| = \prod_{d=1}^D a_{dd}.$$

It is not easy to gain an intuition into what the determinant represents. Its role in the normalisation constant of the multivariate Gaussian leads us to think of it as related to the volume of the Gaussian unnormalised Gaussian (remember that the normalised volume must be equal to 1) and it may be useful to think of it in this way.

The exponential of a sum is a product of exponentials, allowing us to rewrite the expression as follows:

$$= \frac{1}{(2\pi)^{D/2} |\mathbf{I}|^{1/2}} \prod_{d=1}^D \exp \left\{ -\frac{1}{2} (x_d - \mu_d)^2 \right\}$$

$|\mathbf{I}|$ is the determinant of \mathbf{I} , which, from the discussion of diagonal matrices in Comment 2.5, is equal to 1. The other constant term, $(2\pi)^{D/2}$, could be written as $\prod_{d=1}^D (2\pi)^{1/2}$ and so our expression can be rewritten as:

$$p(\mathbf{x}) = \prod_{d=1}^D \frac{1}{(2\pi)^{1/2}} \exp \left\{ -\frac{1}{2} (x_d - \mu_d)^2 \right\}$$

Each term in the product is a univariate Gaussian (with mean μ_d and variance 1) and therefore, by the definition of independence, the elements of \mathbf{x} are independent. This result doesn't just hold for $\Sigma = \mathbf{I}$; it holds for any covariance matrix that has non-zero elements only in the diagonal positions. These diagonal elements will be the variances of the individual, univariate Gaussians (see Exercises EX 2.5 and EX 2.6 for further exercises and practice at this kind of Gaussian manipulation).

The second row in Figure 2.8 gives another example, with parameters:

$$\boldsymbol{\mu} = [2, 1]^\top, \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

In this example, we could not write the pdf as a product of univariate Gaussians, suggesting that the elements of \mathbf{x} are not independent. We can also see the dependence between them in the contour plot (bottom right of Figure 2.8). If x_1 and x_2 are independent, $p(x_2|x_1)$ should not vary with different values of x_1 . Imagine that $x_1 = 3$. It looks from the figure that when $x_1 = 3$, values for x_2 are grouped around 2. If $x_1 = 1$, the values are grouped around 0. Clearly we expect different values of x_2 in both cases and, intuitively, x_1 and x_2 are dependent (MATLAB script: `gauss_surf.m`). Experiment with the values in the covariance matrix to see the effect this has on the surface and contour plots.

A nice feature of the multivariate Gaussian is that the conditional density function $p(x_2|x_1)$ is another Gaussian for which we can easily obtain the mean and variance. We will omit the details here but it is something that we will use often.

2.5.5 Summary

This completes our brief introduction to random variables and probability. Although we have only skimmed the surface of an enormous subject, the material presented in the previous few sections is sufficient for us to extend our model to explicitly measure the discrepancy between predictions and measurements. In the remainder of this chapter, we will add a random variable to our model that will model the error between the linear model and our data. Assuming that the random variable follows a Gaussian density, we will end up with exactly the same equation for $\hat{\mathbf{w}}$ (the optimum parameter value) as in Chapter 1. However, the inclusion of the noise term allows us to obtain degrees of confidence in both our parameter values and predictions.

2.6 Thinking generatively...continued

We now have a sufficient grounding in random variables to be able to handle the errors in our linear model (as shown in Figure 2.1). In Section 2.1.1 we began thinking about how we could generate data that looks like the data that we have observed. In particular, we considered generating the n th winning time from a function of the form $\mathbf{w}^\top \mathbf{x}_n$ and then adding a random quantity that we shall call ϵ_n – a random variable.

Our model now takes the following form:

$$t_n = \mathbf{w}^\top \mathbf{x}_n + \epsilon_n. \quad (2.29)$$

To complete the definition of this model, we need to decide on a distribution for ϵ_n . First, it should be clear that the difference between the model and the actual winning times is a continuous quantity. Therefore, ϵ_n is a continuous random variable. We also do not just have one random variable, but one for each observed Olympics year.

It seems reasonable to assume that these values are independent:

$$p(\epsilon_1, \dots, \epsilon_N) = \prod_{n=1}^N p(\epsilon_n).$$

The final assumption is the form of $p(\epsilon_n)$. We will assume that this is a Gaussian (or normal) distribution with zero mean and variance σ^2 . We will not make much effort to justify this assumption here except to say that this allows ϵ_n to be both positive and negative (allows data to lie both above and below the line $\mathbf{w}^\top \mathbf{x}$) and has interesting modelling properties that link it to the squared loss that we used in Chapter 1. As for the choice of loss functions discussed in Section 1.1.3, in a real modelling situation one should be much more careful to properly justify this choice.

Using a normal density for ϵ , i.e. $p(\epsilon) = \mathcal{N}(\mu, \sigma^2)$ (see Section 2.5.3) with a mean (μ) of zero and a variance of $\sigma^2 = 0.05$ (don't worry about the particular value here for now) we obtain a much more realistic looking dataset, shown in Figure 2.9 (MATLAB script: `genolymp.m`).

Our model now consists of two components:

1. A **deterministic** component ($\mathbf{w}^\top \mathbf{x}_n$), sometimes referred to as a *trend* or *drift*.
2. A random component (ϵ_n), sometimes referred to as *noise*.

We have already pointed out that we are not restricted to noise from a Gaussian distribution. We are also not restricted to *additive* noise. For some applications, a multiplicative term might be more appropriate (in which case, $t = f(\mathbf{x}; \mathbf{w})\epsilon$). For example, degradation of image pixels is often modelled with multiplicative noise. However, as we shall see in the following sections, choosing additive Gaussian noise allows us to obtain exact expressions for the optimal parameter value $\hat{\mathbf{w}}$.

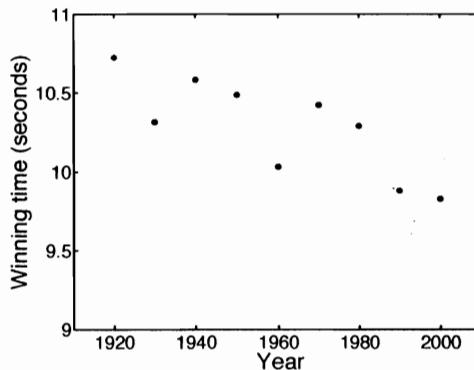


FIGURE 2.9: Dataset generated from a linear model with Gaussian errors.

2.7 Likelihood

Our model is of the following form:

$$t_n = f(\mathbf{x}_n; \mathbf{w}) + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, \sigma^2).$$

As in Chapter 1, we need to find the optimal value of \mathbf{w} , $\hat{\mathbf{w}}$. We also have an additional parameter σ^2 that needs to be set. In Chapter 1 we found the value of \mathbf{w} that minimised the loss. The loss measured the difference between the observed values of t and those predicted by the model. The effect of adding a random variable to the model is that the output of the model, t , is now itself a random variable. In other words, there is no single value of t_n for a particular \mathbf{x}_n . As such, we cannot use the loss as a means of optimising \mathbf{w} and σ^2 .

Adding a constant ($\mathbf{w}^\top \mathbf{x}_n$) to a Gaussian random variable is equivalent to another Gaussian random variable with the mean shifted by the same constant:

$$\begin{aligned} y &= a + z \\ p(z) &= \mathcal{N}(m, s) \\ p(y) &= \mathcal{N}(m + a, s) \end{aligned}$$

Therefore, the random variable t_n has the following density function:

$$p(t_n | \mathbf{x}_n, \mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{w}^\top \mathbf{x}_n, \sigma^2)$$

Note the conditioning on the left hand side – the density of t_n depends on particular values of \mathbf{x}_n and \mathbf{w} (they determine the mean) and σ^2 (the variance).

To see how we can use this to find optimal values of \mathbf{w} and σ^2 , consider one of the years from our dataset – 1980. Based on the model (w_0, w_1) found in the previous chapter and assuming again that $\sigma^2 = 0.05$, we can plot $p(t_n | \mathbf{x}_n = 1980, \mathbf{w}, \sigma^2)$ as a function of t_n , as shown Figure 2.10. The solid line shows:

$$p(t_n | \mathbf{x}_n = [1, 1980]^\top, \mathbf{w} = [36.416, -0.0133]^\top, \sigma^2 = 0.05),$$

which is a Gaussian density with mean $\mu = 36.416 - 0.0133 \times 1980 = 10.02$ and variance $\sigma^2 = 0.05$. Recall that for a continuous random variable, t , $p(t)$ cannot be interpreted as a probability. The height of the curve at a particular value of t can be interpreted as how *likely* it is that we would observe that particular t for $x = 1980$. The most *likely* winning time in 1980 would be 10.02 seconds (for a Gaussian, the most likely (highest) point corresponds to the mean). Also shown on the plot are three example times – A, B and C. Of these, B is the most likely and C the least likely.

The actual winning time in the 1980 Olympics is C (10.25 seconds). The density $p(t_n | \mathbf{x}_n, \mathbf{w}, \sigma^2)$ evaluated at $t_n = 10.25$ is an important quantity, known as the **likelihood** of the n th data point. We cannot change $t_n = 10.25$ (this is our data) but we can change \mathbf{w} and σ^2 to try and move the density so as to make it as high as possible at $t = 10.25$. The idea of finding parameters that maximise the likelihood in this way is a key concept in machine learning.

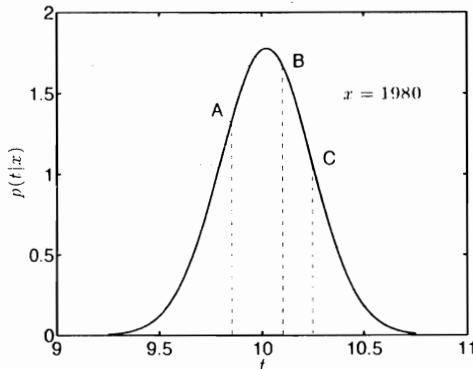


FIGURE 2.10: Likelihood function for the year 1980.

2.7.1 Dataset likelihood

In general, we are not interested in the likelihood of a single data point but that of all of the data. If we have N data points, we are interested in the joint conditional density:

$$p(t_1, \dots, t_N | \mathbf{x}_1, \dots, \mathbf{x}_N, \mathbf{w}, \sigma^2).$$

This is a joint density over all of the responses in our dataset (see Section 2.2.5). We will write this compactly (using vector notation and \mathbf{X} as defined in Chapter 1) as $p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)$. Evaluating this density at the observed data points gives a single likelihood value for the whole dataset which we can optimise by varying \mathbf{w} and σ^2 .

The assumption that the noise at each data point is independent ($p(\epsilon_1, \dots, \epsilon_N) = \prod_n p(\epsilon_n)$) enables us to factorise this density into something more manageable. In particular, this joint conditional density can be factorised into N separate terms, one for each data object:

$$L = p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{n=1}^N p(t_n | \mathbf{x}_n, \mathbf{w}, \sigma^2) = \prod_{n=1}^N \mathcal{N}(\mathbf{w}^\top \mathbf{x}_n, \sigma^2) \quad (2.30)$$

Note that we haven't gone as far as saying that the t_n values are themselves completely independent. This is not the case – the t_n values are, on average, decreasing over time, suggesting a clear statistical dependence between them. If they were completely independent, it would not be worthwhile actually trying to model the data at all. In fact, they are **conditionally independent** – given a value for \mathbf{w} (the deterministic part of the model), the t_n are independent; without them they are not. If this sounds a bit strange, think of it in the following way: Imagine that we had values for all of the Olympics years and winning times except one of the ones in the middle – say 1960. For simplicity, we shall use \mathbf{X}, \mathbf{t} to denote all Olympics years and winning times excluding 1960. If we want to use \mathbf{X} and \mathbf{t} to learn something about t_{1960} , we are interested in the following conditional distribution:

$$p(t_{1960} | \mathbf{x}_{1960}, \mathbf{X}, \mathbf{t}).$$

From the definition of conditional distributions, this is given by:

$$p(t_{1960} | \mathbf{x}_{1960}, \mathbf{X}, \mathbf{t}) = \frac{p(t_{1960}, \mathbf{t} | \mathbf{x}_{1960}, \mathbf{X})}{p(\mathbf{t} | \mathbf{X})}$$

Assuming that the elements of \mathbf{t} are independent results in t_{1960} only depending on \mathbf{x}_{1960} :

$$p(t_{1960} | \mathbf{x}_{1960}, \mathbf{X}, \mathbf{t}) = \frac{p(t_{1960} | \mathbf{x}_{1960}) \prod_n p(t_n | \mathbf{x}_n)}{\prod_n p(t_n | \mathbf{x}_n)} = p(t_{1960} | \mathbf{x}_{1960}).$$

However, for our model to be of any use, t_{1960} must, in some sense, be dependent on the other data. This dependence is encapsulated in the parameter \mathbf{w} . The deterministic part of our model captures this dependence. If we know \mathbf{w} , all that remains is the errors between the observed data and $\mathbf{w}^\top \mathbf{x}_n$. These errors are assumed to be independent. Hence, conditioned on \mathbf{w} , the observations are independent. Without a model (and therefore a \mathbf{w}), the observations are not independent.

We will now show how we can find the values of \mathbf{w} and σ^2 that maximise the likelihood.

2.7.2 Maximum likelihood

Equation 2.30 gives us a single value that tells us how likely our dataset is, given the current model (by model, we mean choice of \mathbf{w} and σ^2). As our dataset is fixed, varying the model will result in different likelihood values. A sensible choice of model would be that which maximised the likelihood. In other words, we will select the model parameters that will make our observations most likely.

For analytical reasons, we will maximise the **natural logarithm** of the likelihood (we will follow the Machine Learning convention of using $\log(y)$ to denote the natural logarithm of y , often denoted elsewhere as $\ln(y)$). We can do this because the estimated arguments $\hat{\mathbf{w}}$ and $\hat{\sigma}^2$ that maximise the log likelihood will also maximise the likelihood.

Substituting the expression for the Gaussian density function (Equation 2.27) and separating the various terms gives us an expression that will be easier to deal with:

$$\begin{aligned} \log L &= \sum_{n=1}^N \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{1}{2\sigma^2} (t_n - f(\mathbf{x}_n; \mathbf{w}))^2 \right\} \right) \\ &= \sum_{n=1}^N \left(-\frac{1}{2} \log(2\pi) - \log \sigma - \frac{1}{2\sigma^2} (t_n - f(\mathbf{x}_n, \mathbf{w}))^2 \right) \\ &= -\frac{N}{2} \log 2\pi - N \log \sigma - \frac{1}{2\sigma^2} \sum_{n=1}^N (t_n - f(\mathbf{x}_n, \mathbf{w}))^2. \end{aligned}$$

Substituting our particular deterministic component $f(\mathbf{x}_n; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}_n$ gives us the log likelihood expression that we will work with:

$$\log L = -\frac{N}{2} \log 2\pi - N \log \sigma - \frac{1}{2\sigma^2} \sum_{n=1}^N (t_n - \mathbf{w}^\top \mathbf{x}_n)^2. \quad (2.31)$$

As for the least squares solution derived in Chapter 1, we can find the optimal parameters by taking derivatives, equating them to zero and solving for turning points, in a manner similar to that in Section 1.1.4. For \mathbf{w} (noting that $\mathbf{w}^\top \mathbf{x}_n = \mathbf{x}_n^\top \mathbf{w}$),

$$\begin{aligned}\frac{\partial \log L}{\partial \mathbf{w}} &= \frac{1}{\sigma^2} \sum_{n=1}^N \mathbf{x}_n(t_n - \mathbf{x}_n^\top \mathbf{w}) \\ &= \frac{1}{\sigma^2} \sum_{n=1}^N \mathbf{x}_n t_n - \mathbf{x}_n \mathbf{x}_n^\top \mathbf{w} = \mathbf{0}.\end{aligned}$$

Note that $\frac{\partial \log L}{\partial \mathbf{w}}$ is a vector and so we equate it to $\mathbf{0}$, a vector of zeros of the same size. Recall the shorthand matrix/vector forms we used in Chapter 1:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}.$$

In this notation, $\sum_{n=1}^N \mathbf{x}_n t_n$ can be written as $\mathbf{X}^\top \mathbf{t}$ and similarly $\sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top \mathbf{w}$ as $\mathbf{X}^\top \mathbf{X} \mathbf{w}$ (see Exercise EX 1.5). This allows us to write the derivative in the more convenient vector/matrix form:

$$\frac{\partial \log L}{\partial \mathbf{w}} = \frac{1}{\sigma^2} (\mathbf{X}^\top \mathbf{t} - \mathbf{X}^\top \mathbf{X} \mathbf{w}) = \mathbf{0}. \quad (2.32)$$

Solving this expression for \mathbf{w} will lead to an expression for the optimal value:

$$\begin{aligned}\frac{1}{\sigma^2} (\mathbf{X}^\top \mathbf{t} - \mathbf{X}^\top \mathbf{X} \mathbf{w}) &= 0 \\ \mathbf{X}^\top \mathbf{t} - \mathbf{X}^\top \mathbf{X} \mathbf{w} &= 0 \\ \mathbf{X}^\top \mathbf{X} \mathbf{w} &= \mathbf{X}^\top \mathbf{t} \\ \mathbf{w} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}.\end{aligned}$$

This is the **maximum likelihood** solution for \mathbf{w} :

$$\widehat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} \quad (2.33)$$

Remarkably, this solution is *exactly* that which we have already derived for the least squares case in Chapter 1 (Equation 1.16). Minimising the squared loss is equivalent to the maximum likelihood solution if the noise is assumed to be Gaussian. Also, the noise variance, σ^2 , does not appear in this expression at all – it scales the likelihood but doesn't affect the value of $\widehat{\mathbf{w}}$ corresponding to its maximum.

To obtain an expression for σ^2 (assuming $\mathbf{w} = \widehat{\mathbf{w}}$) we can follow the same procedure. Taking partial derivatives and equating to zero results in

$$\frac{\partial L}{\partial \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{n=1}^N (t_n - \mathbf{x}^\top \widehat{\mathbf{w}})^2 = 0. \quad (2.34)$$

Re-arranging gives $\widehat{\sigma^2}$, the maximum likelihood estimate for σ^2 :

$$\widehat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^N (t_n - \mathbf{x}^\top \widehat{\mathbf{w}})^2. \quad (2.35)$$

This expression makes perfect sense – the variance is simply the average squared error. We would prefer this in matrix notation so, using the fact that $\sum_{n=1}^N (t_n - \mathbf{x}^\top \widehat{\mathbf{w}})^2$ is equivalent to $(\mathbf{t} - \mathbf{X}\widehat{\mathbf{w}})^\top (\mathbf{t} - \mathbf{X}\widehat{\mathbf{w}})$,

$$\begin{aligned} \sigma^2 &= \frac{1}{N} (\mathbf{t} - \mathbf{X}\widehat{\mathbf{w}})^\top (\mathbf{t} - \mathbf{X}\widehat{\mathbf{w}}) \\ &= \frac{1}{N} (\mathbf{t}^\top \mathbf{t} - 2\mathbf{t}^\top \mathbf{X}\widehat{\mathbf{w}} + \widehat{\mathbf{w}}^\top \mathbf{X}^\top \mathbf{X}\widehat{\mathbf{w}}). \end{aligned} \quad (2.36)$$

This can be further simplified by substituting $\widehat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}$ (note that $\widehat{\mathbf{w}}^\top = \mathbf{t}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}$ because $(\mathbf{X}^\top \mathbf{X})^{-1}$ is **symmetric** and is therefore equal to its own transpose):

$$\begin{aligned} \widehat{\sigma}^2 &= \frac{1}{N} (\mathbf{t}^\top \mathbf{t} - 2\mathbf{t}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} + \mathbf{t}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}) \\ &= \frac{1}{N} (\mathbf{t}^\top \mathbf{t} - 2\mathbf{t}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} + \mathbf{t}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}) \\ &= \frac{1}{N} (\mathbf{t}^\top \mathbf{t} - \mathbf{t}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}) \\ \widehat{\sigma}^2 &= \frac{1}{N} (\mathbf{t}^\top \mathbf{t} - \mathbf{t}^\top \mathbf{X}\widehat{\mathbf{w}}). \end{aligned} \quad (2.37)$$

Using the Olympics 100 m data, our optimal parameter values (for a 1st order (linear) polynomial) are:

$$\widehat{\mathbf{w}} = [36.4165, -0.0133]^\top, \widehat{\sigma}^2 = 0.0503.$$

$\widehat{\mathbf{w}}$ is the same as the least squares solution provided in the previous chapter (they are both computed using the same expression). $\widehat{\sigma}^2$ tells us the variance of the Gaussian noise that we have assumed is used to corrupt our data. Later in this chapter we will see that modelling the noise in this way provides several benefits over loss minimisation. Before we do, we shall first look at some of the characteristics of the solution.

2.7.3 Characteristics of the maximum likelihood solution

In Chapter 1, we used the second derivatives of the loss function to ensure that we had found a minimum. We will now do a similar thing with the second derivatives of the likelihood to ensure that we have found a maximum. Our derivatives are now with respect to a vector and to examine the second derivatives, we construct the **Hessian matrix** (see Comment 2.6). Each entry in this matrix is the second derivative with respect to a pair of elements of \mathbf{w} . To be sure that we have found a maximum, we must show that the Hessian matrix is *negative definite* (see Comment 2.7).

Comment 2.6 – Hessian matrix: A Hessian matrix is square matrix containing all of the second order partial derivatives of a function. For example, the Hessian matrix for a function $f(\mathbf{x}; \mathbf{w})$ with parameters $\mathbf{w} = [w_1, \dots, w_K]^T$ would be

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial w_1^2} & \frac{\partial^2 f}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_1 \partial w_K} \\ \frac{\partial^2 f}{\partial w_2 \partial w_1} & \frac{\partial^2 f}{\partial w_2^2} & \cdots & \frac{\partial^2 f}{\partial w_2 \partial w_K} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial w_K \partial w_1} & \frac{\partial^2 f}{\partial w_K \partial w_2} & \cdots & \frac{\partial^2 f}{\partial w_K^2} \end{bmatrix}.$$

We can use the Hessian to tell us something about turning points in $f(\mathbf{x}; \mathbf{w})$. For example, if the Hessian is *negative definite* (see Comment 2.7) at some turning point $\hat{\mathbf{w}}$, then we know that that turning point corresponds to a maximum.

The Hessian matrix of second order partial derivatives can be computed by differentiating Equation 2.32 with respect to \mathbf{w}^\top :

$$\frac{\partial^2 \log L}{\partial \mathbf{w} \partial \mathbf{w}^\top} = -\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X}. \quad (2.38)$$

If we substitute $\mathbf{x}_n = [1, x_n]^\top$, the diagonal elements of this matrix are equivalent (they differ by multiplication by a constant) to the second derivatives obtained in Equation 1.9 (see Exercise EX 2.7).

Comment 2.7 – Negative definite matrices: A real-valued matrix \mathbf{H} is negative definite if

$$\mathbf{x}^\top \mathbf{H} \mathbf{x} < 0$$

for all vectors of real values \mathbf{x} .

To be sure this is a maximum, we need to determine whether or not this matrix is negative definite. We can do this by showing that

$$-\frac{1}{\sigma^2} \mathbf{z}^\top \mathbf{X}^\top \mathbf{X} \mathbf{z} < 0$$

for any vector \mathbf{z} or equivalently (because σ^2 must be positive) that

$$\mathbf{z}^\top \mathbf{X}^\top \mathbf{X} \mathbf{z} > 0$$

for any vector \mathbf{z} . At this stage, it is probably worth showing how this can be done. We will assume that each \mathbf{x}_n is two dimensional so that we can explicitly multiply out the various terms. To be more general, we will define \mathbf{X} slightly differently than before as:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ \vdots & \ddots \\ x_{N1} & x_{N2} \end{bmatrix},$$

Thus, $\mathbf{X}^\top \mathbf{X}$ is

$$\mathbf{X}^\top \mathbf{X} = \begin{bmatrix} \sum_{i=1}^N x_{i1}^2 & \sum_{i=1}^N x_{i1} x_{i2} \\ \sum_{i=1}^N x_{i2} x_{i1} & \sum_{i=1}^N x_{i2}^2 \end{bmatrix}.$$

Pre- and post-multiplying by an arbitrary real vector $\mathbf{z} = [z_1, z_2]^\top$,

$$\begin{aligned}\mathbf{z}^\top \mathbf{X}^\top \mathbf{X} \mathbf{z} &= \left[z_1 \sum_{i=1}^N x_{i1}^2 + z_2 \sum_{i=1}^N x_{i2} x_{i1}, z_1 \sum_{i=1}^N x_{i1} x_{i2} + z_2 \sum_{i=1}^N x_{i2}^2 \right] \mathbf{z} \\ &= z_1^2 \sum_{i=1}^N x_{i1}^2 + 2z_1 z_2 \sum_{i=1}^N x_{i1} x_{i2} + z_2^2 \sum_{i=1}^N x_{i2}^2.\end{aligned}$$

Because the first and last terms must be positive, proving that this expression is greater than zero is equivalent to proving that their combined value is larger than the middle term:

$$z_1^2 \sum_{i=1}^N x_{i1}^2 + z_2^2 \sum_{i=1}^N x_{i2}^2 > 2z_1 z_2 \sum_{i=1}^N x_{i1} x_{i2}.$$

Defining $y_{i1} = z_1 x_{i1}$ and $y_{i2} = z_2 x_{i2}$ and substituting into our expression gives

$$\sum_{i=1}^N (y_{i1}^2 + y_{i2}^2) > 2 \sum_{i=1}^N y_{i1} y_{i2}.$$

Now, considering some arbitrary i ,

$$\begin{aligned}y_{i1}^2 + y_{i2}^2 &> 2y_{i1} y_{i2} \\ y_{i1}^2 - 2y_{i1} y_{i2} + y_{i2}^2 &> 0 \\ (y_{i1} - y_{i2})^2 &> 0\end{aligned}$$

which will only not be the case if $y_{i1} = y_{i2}$ and therefore $x_{i1} = x_{i2}$ – something unlikely to happen in practice. So, if $y_{i1}^2 + y_{i2}^2 > 2y_{i1} y_{i2}$ holds for any i , the summation of any number of these terms must also satisfy the inequality. Hence, $\mathbf{z}^\top \mathbf{X}^\top \mathbf{X} \mathbf{z}$ is always positive, our Hessian is negative definite and the solution corresponds to a maximum of the likelihood.

To ensure that our expression for $\widehat{\sigma^2}$ corresponds to a maximum of the likelihood, we differentiate Equation 2.34 again with respect to σ :

$$\frac{\partial^2 \log L}{\partial \sigma^2} = \frac{N}{\sigma^2} - \frac{3}{\sigma^4} (\mathbf{t} - \mathbf{X} \widehat{\mathbf{w}})^\top (\mathbf{t} - \mathbf{X} \widehat{\mathbf{w}}).$$

We can simplify this by substituting the value for $\widehat{\sigma^2}$ given in Equation 2.36, resulting in:

$$\begin{aligned}\frac{\partial^2 \log L}{\partial \sigma^2} &= \frac{N}{\widehat{\sigma^2}} - \frac{3}{(\widehat{\sigma^2})^2} N \widehat{\sigma^2} \\ &= -\frac{2N}{\widehat{\sigma^2}},\end{aligned}$$

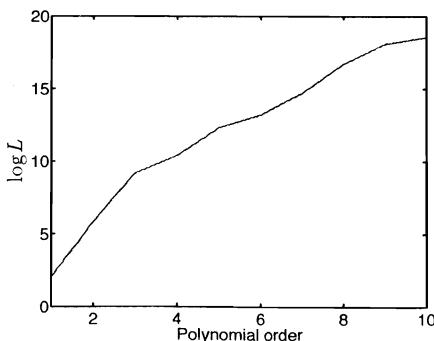
which is always negative and hence $\widehat{\sigma^2}$ corresponds to a maximum.

2.7.4 Maximum likelihood favours complex models

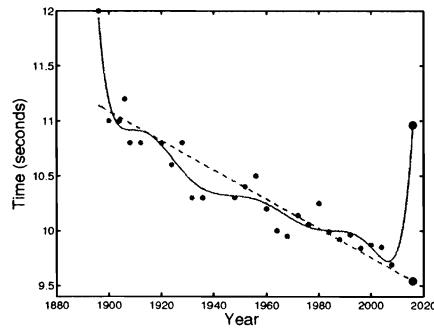
Plugging the expression for $\widehat{\sigma^2}$ (Equation 2.35) into the log likelihood expression (Equation 2.31) gives us the value of the log likelihood at the maximum:

$$\begin{aligned}\log L &= -\frac{N}{2} \log 2\pi - \frac{N}{2} \log \widehat{\sigma^2} - \frac{1}{2\widehat{\sigma^2}} N \widehat{\sigma^2} \\ &= -\frac{N}{2}(1 + \log 2\pi) - \frac{N}{2} \log \widehat{\sigma^2}.\end{aligned}$$

This tells us that the maximum value of L will keep increasing as we decrease $\widehat{\sigma^2}$. Recall that σ^2 is the variance of the noise incorporated into the model to capture effects that the deterministic part of our model (i.e. $f(\mathbf{x}; \mathbf{w})$) cannot. One way to decrease σ^2 is to modify $f(\mathbf{x}; \mathbf{w})$ so that it can capture more of the variability in the data – i.e. make it more flexible. For example, revisiting the Olympics men’s 100 m data we can investigate the increase in likelihood as model flexibility (or complexity) increases by fitting increasingly higher order polynomial functions. Figure 2.11(a) shows that $\log L$ increases as polynomials of increasing order are fitted to the Olympics men’s 100 m data (MATLAB script: `olymplike.m`). If we were to use $\log L$ to help choose which particular model to use, it will always point us to models of increasing complexity. This might seem like a sensible strategy – as $\widehat{\sigma^2}$ decreases, the deterministic part of our model must be capturing more of the variability in our data. However, consider the task of predicting the winning time for a year that we have not yet observed (e.g. 2016). Figure 2.11(b) shows 1st (dashed line) and 8th (solid line) order polynomial fits as well as their predictions for 2016 (shown as large dark circles). The more complex model makes a prediction of a winning time of close to 11 seconds (it would be one of the slowest ever) whereas the simpler model makes a much more realistic prediction. To the human eye, it looks like the simpler model has captured the important relationship in the data (the general downward trend) whilst the more complex model has not. This is a nice example of the trade-off between generalisation and over-fitting that we saw in Section 1.5. The



(a) Increase in log likelihood as the polynomial order increases



(b) 1st and 8th order polynomial functions fitted to the Olympics men’s 100 m data. Large dark circles correspond to predictions for the 2016 Olympics

FIGURE 2.11: Model complexity example with Olympics men’s 100 m data.

simpler model is better able to generalise than the more complex one. The more complex model is over-fitting – we have given the model too much freedom and it is attempting to make sense out of what is essentially noise. In Section 1.6 we showed how regularisation could be used to penalise overly complex parameter values. The same can be done with probabilistic models through the use of **prior distributions** on the parameter values. This will be introduced in the next chapter.

2.8 The bias-variance tradeoff

The tradeoff between generalisation and over-fitting discussed in Section 1.5 is also sometimes described as the bias-variance tradeoff. Imagine that we had access to the distribution from which the data were sampled, $p(\mathbf{x}, t)$. Using this distribution we could, in theory, compute the expected value of the squared error between estimated parameter values and the true values. We would like this value, $\bar{\mathcal{M}}$, to be as low as possible. It can be decomposed into two terms called the bias, \mathcal{B} , and the variance, \mathcal{V} :

$$\bar{\mathcal{M}} = \mathcal{B}^2 + \mathcal{V}.$$

The bias describes the systematic mismatch between our model and the process that generated the data. A model that is too simple will have a high bias (under-fitting). We can therefore decrease the bias and its contribution to the $\bar{\mathcal{M}}$ by making the model more complex. Unfortunately, more complex models have higher variance, thus increasing the \mathcal{V} component of $\bar{\mathcal{M}}$. Finding the correct balance between generalisation and over/under-fitting can thus also be thought of as finding the correct balance between bias and variance.

We omit further details here, but more details can be found in the suggested reading at the end of this chapter.

2.8.1 Summary

In the previous sections we have introduced a number of new concepts. First, we made a case for explicitly modelling the noise (or errors) in our dataset. Making the assumption that these errors could be adequately modelled by a Gaussian random variable, we showed that we could compute a quantity called the *likelihood* that describes how likely our data is as a function of our model parameters. This is a reasonable quantity to maximise when choosing our parameters and maximising the likelihood and minimising the squared loss give identical expressions for the optimal parameter values when we assume that the noise is Gaussian. In the remainder of the chapter we will look at two important benefits of explicitly modelling the noise: the ability to quantify the uncertainty in our parameters and the ability to express uncertainties in our predictions.

2.9 Effect of noise on parameter estimates

In this section we shall derive expressions for how much confidence we should place in our parameter estimates – how much could we change the straight line and still have a *good* model. If there is a lot of noise (σ^2 is high) it is likely that we could tolerate reasonably large changes in $\hat{\mathbf{w}}$. If there is very little noise, the quality of the fit will deteriorate rapidly. Before we derive these expressions, it is useful to explore the variability in $\hat{\mathbf{w}}$ by generating synthetic data. In particular, we shall generate lots of datasets with the same true \mathbf{w} and σ^2 and see how our maximum likelihood estimate $\hat{\mathbf{w}}$ varies. Consider the following model:

$$t_n = w_0 + w_1 x_n + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, \sigma^2). \quad (2.39)$$

Assuming that the true parameter values are $w_0 = -2$, $w_1 = 3$ and the noise variance is $\sigma^2 = 0.5^2$, we can generate as many sets of responses (t_1, \dots, t_N) as we like for a particular set of attributes (x_1, \dots, x_N) and compute $\hat{\mathbf{w}}$ for each set. An example of one such dataset and the *true* function can be seen in Figure 2.12, where the set of attributes consists of 20 values drawn from a uniform distribution between 0 and 1, i.e. $p(x) = \mathcal{U}(0, 1)$. Figure 2.13 shows the results of generating 10,000 datasets and fitting $\hat{\mathbf{w}}$ in each case. The left panel shows a histogram where the height of each bar represents the number of datasets that resulted in parameter values within a particular range and the right panel shows the same information as a contour

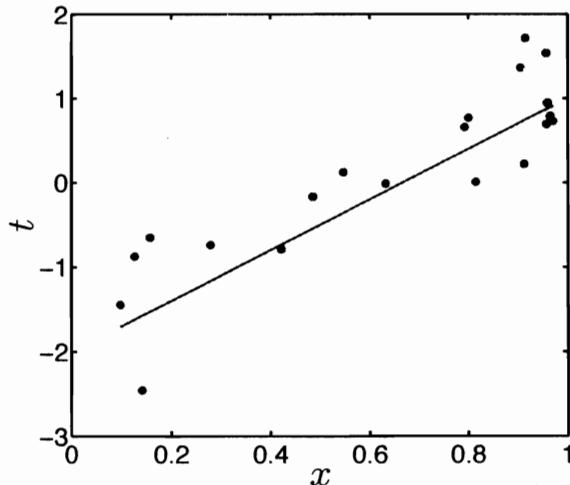


FIGURE 2.12: Data generated from the model given in Equation 2.39 and the true function.

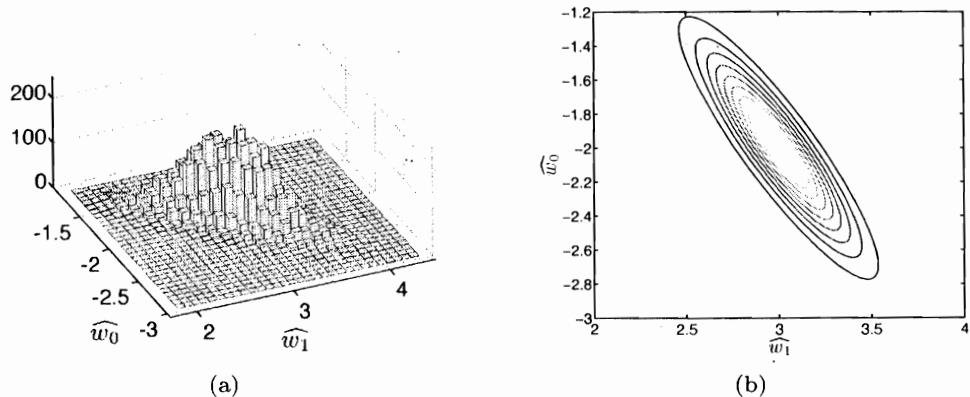


FIGURE 2.13: Variability in $\widehat{\mathbf{w}}$ for 10,000 datasets generated from the model described in Equation 2.39.

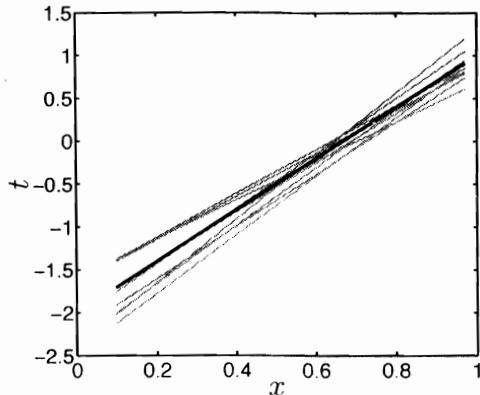


FIGURE 2.14: Functions inferred from 10 datasets generated from the model given in Equation 2.39 as well as the true function (wider, darker line).

plot. We can see a wide variability around the true values in both \widehat{w}_0 and \widehat{w}_1 . It is hard from these values to get a feel for how much variability this implies in the model, so examples of $\widehat{\mathbf{w}}$ from 10 datasets as well as the true function are plotted in Figure 2.14.

If we assume our real data to have been generated by such a process, it is useful to be able to quantify how variable our resulting estimates are. Unfortunately, we don't have access to many datasets from which we can compare values of $\widehat{\mathbf{w}}$. In the next section we will show how we can quantify this uncertainty using just the data that are available.

2.9.1 Uncertainty in estimates

We showed in the last section that the value we obtain for $\hat{\mathbf{w}}$ is strongly influenced by the particular noise values in the data. In light of this, it would be useful to know how much uncertainty there was in $\hat{\mathbf{w}}$. In other words, is this $\hat{\mathbf{w}}$ unique in explaining the data well or are there many that could do almost as well?

To progress, we must be very clear about what \mathbf{w} and $\hat{\mathbf{w}}$ mean. We have hypothesised a model which was responsible for the data. This model is

$$t_n = \mathbf{w}^\top \mathbf{x}_n + \epsilon_n$$

where \mathbf{w} represents the *true* value of the parameters and ϵ_n is a random variable that we have defined to be normally distributed. This assumption means that the *generating distribution* (or likelihood), $p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)$, is a product of normal densities:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{n=1}^N p(t_n|\mathbf{x}_n \mathbf{w}, \sigma^2) = \prod_{n=1}^N \mathcal{N}(\mathbf{w}^\top \mathbf{x}_n, \sigma^2).$$

In Section 2.5.4, we showed how a product of univariate Gaussian densities could be written as a multivariate Gaussian density with a diagonal covariance. It will be neater to work with a single multivariate Gaussian than a product over univariate ones. In this case, the multivariate Gaussian is:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I}).$$

Satisfy yourself that the mean and covariance terms are correct. Now, $\hat{\mathbf{w}}$ is an estimate of the true parameter value \mathbf{w} . Computing the expectation (Section 2.2.8) of $\hat{\mathbf{w}}$ with respect to the generating distribution will tell us what we expect $\hat{\mathbf{w}}$ to be, on average:

$$\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \{ \hat{\mathbf{w}} \} = \int \hat{\mathbf{w}} p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) d\mathbf{t}$$

Substituting $\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}$ into this expression allows us to evaluate the integral:

$$\begin{aligned} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \{ \hat{\mathbf{w}} \} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \int \mathbf{t} p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) d\mathbf{t} \\ \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \{ \hat{\mathbf{w}} \} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \{ \mathbf{t} \} \\ \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \{ \hat{\mathbf{w}} \} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X}\mathbf{w} \\ \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \{ \hat{\mathbf{w}} \} &= \mathbf{w} \end{aligned} \tag{2.40}$$

where we have used the fact that the expected value of a normally distributed random variable is equal to its mean ($\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \{ \mathbf{t} \} = \mathbf{X}\mathbf{w}$ because $p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I})$).

This result tells us that the expected value of our approximation $\hat{\mathbf{w}}$ is the true parameter value. We will consider this in more detail later in the chapter, but it means that our estimator is **unbiased** – it is not, on average, too big or too small.

This potential variability in the estimate of $\hat{\mathbf{w}}$ is encapsulated in its *covariance matrix*. For our purposes, this covariance matrix provides us with two useful pieces of information. The diagonal elements (the variances of the individual elements in

$\widehat{\mathbf{w}}$) tell us how much variability we might expect in the individual parameters – i.e. how well they are defined by the data. In our experiment above, the parameters appeared to vary quite a lot, suggesting that they were not defined very well by the data. The off-diagonal elements tell us how the parameters co-vary – if the values are high and positive, it tells us that increasing one will require an increase in the other to maintain a *good* model. Large negative values tell us the opposite – increasing one will cause a decrease in the other. Values close to zero tell us that the parameters are not dependent on one another. For the example described above, it looks (see Figure 2.13) like increasing w_1 causes a decrease in w_0 , so we might expect the off-diagonal elements in the covariance matrix to be negative.

In Section 2.2.8, we derived a general expression for the covariance matrix (Equation 2.16). Substituting \mathbf{t} and $p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)$ into this expression, and using the previous result, $\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\widehat{\mathbf{w}}\} = \mathbf{w}$, gives us:

$$\begin{aligned}\text{cov}\{\widehat{\mathbf{w}}\} &= \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\widehat{\mathbf{w}}\widehat{\mathbf{w}}^\top\} - \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\widehat{\mathbf{w}}\}\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\widehat{\mathbf{w}}\}^\top \\ &= \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\widehat{\mathbf{w}}\widehat{\mathbf{w}}^\top\} - \mathbf{w}\mathbf{w}^\top\end{aligned}\quad (2.41)$$

where we have used the expectation of $\widehat{\mathbf{w}}$ that we derived above. To compute this quantity, we will start with the first term. It can be expanded by substituting $\widehat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}$ and all of the terms that do not involve \mathbf{t} can be removed from the expectation:

$$\begin{aligned}\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\widehat{\mathbf{w}}\widehat{\mathbf{w}}^\top\} &= \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\left\{((\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t})((\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t})^\top\right\} \\ &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{t}\mathbf{t}^\top\} \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1}.\end{aligned}\quad (2.42)$$

Now, $p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I})$. Therefore the covariance of \mathbf{t} is, by definition, $\sigma^2\mathbf{I}$ and its mean is $\mathbf{X}\mathbf{w}$. By the same line of derivation that allowed us to reach Equation 2.41, we have:

$$\text{cov}\{\mathbf{t}\} = \sigma^2\mathbf{I} = \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{t}\mathbf{t}^\top\} - \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{t}\}\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{t}\}^\top. \quad (2.43)$$

Therefore, we can re-arrange this expression to obtain an expression for $\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{t}\mathbf{t}^\top\}$:

$$\begin{aligned}\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{t}\mathbf{t}^\top\} &= \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{t}\}\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{t}\}^\top + \sigma^2\mathbf{I} \\ &= \mathbf{X}\mathbf{w}(\mathbf{X}\mathbf{w})^\top + \sigma^2\mathbf{I} \\ &= \mathbf{X}\mathbf{w}\mathbf{w}^\top \mathbf{X}^\top + \sigma^2\mathbf{I}.\end{aligned}$$

Substituting this into Equation 2.42 gives:

$$\begin{aligned}\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)}\{\mathbf{w}\mathbf{w}^\top\} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X}\mathbf{w}\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \\ &\quad + \sigma^2(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \\ &= \mathbf{w}\mathbf{w}^\top + \sigma^2(\mathbf{X}^\top \mathbf{X})^{-1}.\end{aligned}\quad (2.44)$$

Finally, substituting this into Equation 2.41 gives the expression for the covariance of $\widehat{\mathbf{w}}$:

$$\begin{aligned}\text{cov}\{\widehat{\mathbf{w}}\} &= \mathbf{w}\mathbf{w}^\top + \sigma^2(\mathbf{X}^\top \mathbf{X})^{-1} - \mathbf{w}\mathbf{w}^\top \\ &= \sigma^2(\mathbf{X}^\top \mathbf{X})^{-1}\end{aligned}\quad (2.45)$$

which is the negative of the inverse of the Hessian matrix of second derivatives derived previously (Equation 2.38), i.e.

$$\text{cov}\{\hat{\mathbf{w}}\} = \sigma^2(\mathbf{X}^\top \mathbf{X})^{-1} = - \left(\frac{\partial^2 \log L}{\partial \mathbf{w} \partial \mathbf{w}^\top} \right)^{-1} \quad (2.46)$$

This result tells us that the certainty/uncertainty in the parameters (as described by $\text{cov}\{\hat{\mathbf{w}}\}$) is directly linked to the second derivative of the log likelihood. The second derivative of the log likelihood tells us about the curvature of the likelihood function. Therefore, low curvature corresponds to a high level of uncertainty in parameters and high curvature to a low level. In other words, we have an expression that tells us how much **information** our data gives us regarding our parameter estimates. In fact, our matrix $\sigma^2(\mathbf{X}^\top \mathbf{X})^{-1}$ is the negative inverse of something called the **Fisher Information Matrix** (\mathcal{I}). The Fisher Information Matrix is computed as the expected value of the matrix of second derivatives of the log likelihood:

$$\mathcal{I} = \mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ -\frac{\partial^2 \log p(t|\mathbf{X}, \mathbf{w}, \sigma^2)}{\partial \mathbf{w} \partial \mathbf{w}^\top} \right\}.$$

We already know what the bit in the brackets is – it is the Hessian matrix we calculated earlier, so

$$\mathcal{I} = \mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} \right\}$$

which, because the argument of the expectation is a constant, is just

$$\mathcal{I} = \frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X}. \quad (2.47)$$

The elements of \mathcal{I} tell us how much information (the more negative the information value is the more information is present) the data provides about a particular parameter (diagonal elements) or pairs of parameters (off-diagonal elements). Intuitively, if our data is very noisy, the information content is lower. In general, if the information content is high, the data can inform a very accurate parameter estimate and the covariance of $\hat{\mathbf{w}}$ will be low ($\text{cov}\{\hat{\mathbf{w}}\} = \mathcal{I}^{-1}$). If the information content is low, the covariance will be high (see Exercises EX 2.13 and EX 2.14).

As an example, look at the top line in Figure 2.15. The left-hand plot shows the data and the true function ($t = 3x - 2$) and the right hand plot shows the likelihood as a function of the two parameters. We can see that the likelihood function has a low curvature (contour lines are reasonably far apart) because of the large noise level and, as such, many sets of parameters will result in a reasonable model. A low curvature should, from Equation 2.46, correspond to high covariance in $\hat{\mathbf{w}}$. The Fisher information and covariance matrices are

$$\mathcal{I} = \begin{bmatrix} 50.0000 & 24.3311 \\ 24.3311 & 15.8953 \end{bmatrix}, \text{cov}\{\hat{\mathbf{w}}\} = \begin{bmatrix} 0.0784 & -0.1200 \\ -0.1200 & 0.2466 \end{bmatrix}.$$

It is difficult to know if these correspond to high or low information and covariance without context. This can be provided by comparing them with those obtained from the second dataset (second row in Figure 2.15). This dataset has much less noise and

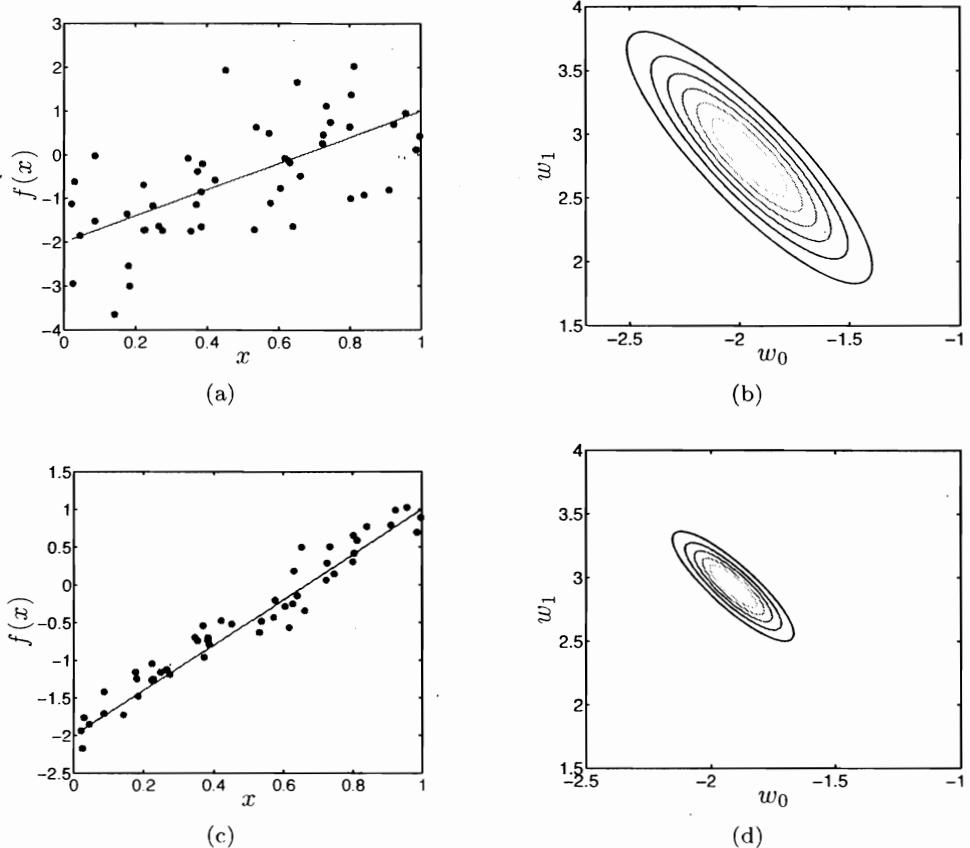


FIGURE 2.15: Two example datasets with different noise levels and the corresponding likelihood function.

the corresponding likelihood curvature is much higher (the contour lines are closer together). In this case, the information and covariance matrices are:

$$\mathcal{I} = \begin{bmatrix} 1.2500 \times 10^3 & 0.6083 \times 10^3 \\ 0.6083 \times 10^3 & 0.3974 \times 10^3 \end{bmatrix}, \quad \text{cov}\{\hat{\mathbf{w}}\} = \begin{bmatrix} 0.0031 & -0.0048 \\ -0.0048 & 0.0099 \end{bmatrix}$$

which have significantly higher (in \$\mathcal{I}\$) and lower (in \$\text{cov}\{\hat{\mathbf{w}}\}\$) values.

2.9.2 Comparison with empirical values

At the start of Section 2.9 we generated many sets of responses for a set of attributes using the model (and associated noise distribution) given in Equation 2.39. If we use \$\hat{\mathbf{w}}_s\$ to describe the parameters obtained from the \$s\$th dataset, the empirical covariance matrix can be computed as

$$\widehat{\text{cov}\{\hat{\mathbf{w}}\}} = \frac{1}{S} \sum_{s=1}^S (\hat{\mathbf{w}}_s - \hat{\mu})(\hat{\mathbf{w}}_s - \hat{\mu})^\top$$

where

$$\hat{\mu} = \frac{1}{S} \sum_{s=1}^S \hat{\mathbf{w}}_s.$$

Using the values shown in Figure 2.13, the empirical covariance matrix is

$$\widehat{\text{cov}\{\hat{\mathbf{w}}\}} = \begin{bmatrix} 0.0627 & -0.0809 \\ -0.0809 & 0.1301 \end{bmatrix}$$

Using Equation 2.45 and the true value of $\sigma^2 = 0.5^2$, the theoretical covariance matrix is

$$\text{cov}\{\hat{\mathbf{w}}\} = \begin{bmatrix} 0.0638 & -0.0821 \\ -0.0821 & 0.1317 \end{bmatrix}$$

which is very close to our empirical value. Normally, we do not have access to a bottomless supply of data and so we can use the theoretical covariance matrix to help understand the variability present in our data. The off-diagonal elements are negative – increasing one of the parameters forces the other to decrease.

To compute the theoretical covariance matrix, we have used the true noise variance. If we take one arbitrary dataset, we can estimate the variance (using Equation 2.35) as $\sigma^2 = 0.2080$ (the true value is $\sigma^2 = 0.25$). The covariance matrix using the estimated variance is:

$$\text{cov}\{\hat{\mathbf{w}}\} = \begin{bmatrix} 0.0530 & -0.0683 \\ -0.0683 & 0.1095 \end{bmatrix}$$

Because the estimated value of σ^2 is lower than the true value, the values in this matrix are lower than those when the true noise value is used. This suggests that the uncertainty is under-estimated and our predictions will be over-confident. The systematic under-estimation of noise variance in maximum likelihood is discussed more thoroughly in Section 2.10.2.

At the start of Section 2.9 we saw that changes in the exact values of the noise changed the parameter estimates. In reality we cannot generate many datasets with which to estimate this uncertainty in parameter values. However, we have derived an expression for the covariance of $\hat{\mathbf{w}}$ that can be used to approximate the uncertainty in parameters. Before we move on to variability in predictions, we will look at the uncertainty present in the maximum likelihood estimations from the Olympics data.

2.9.3 Variability in model parameters – Olympics data

Using the now familiar men's Olympics 100 m data and the standard linear function

$$f(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T \mathbf{x},$$

we know that the maximum likelihood value of \mathbf{w} , $\hat{\mathbf{w}}$, will be $[36.4165, -0.0133]^T$ (from Equation 2.33). The maximum likelihood variance value, $\widehat{\sigma^2}$, can be computed using Equation 2.37 and is $\widehat{\sigma^2} = 0.0503$. Using Equation 2.45, and using $\widehat{\sigma^2}$ as an estimate of σ^2 , we can compute the covariance matrix of the estimate:

$$\text{cov}\{\hat{\mathbf{w}}\} = \begin{bmatrix} 5.7972 & -0.0030 \\ -0.0030 & 1.5204e-06 \end{bmatrix}.$$

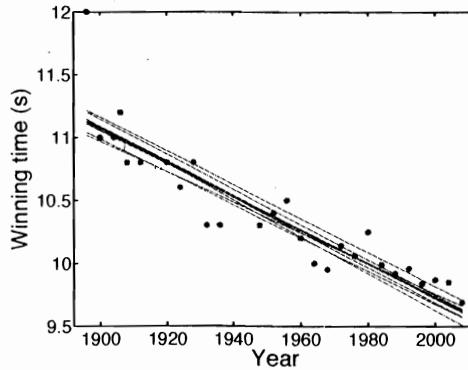


FIGURE 2.16: Ten samples of \mathbf{w} using the distribution given in Equation 2.48.

Taking the diagonal elements, we can see that the variance of \widehat{w}_0 (5.7972) is much higher than the variance in \widehat{w}_1 ($1.5204e - 06$), suggesting that we could tolerate bigger changes in \widehat{w}_0 than \widehat{w}_1 and still be left with a reasonably good model. In part, this can be explained by the fact that \widehat{w}_0 has a much higher absolute value. The negativity of the off-diagonal elements tells us that if we were to slightly increase either \widehat{w}_0 or \widehat{w}_1 , we would have to slightly decrease the other. This is relatively intuitive – if we were to slightly increase \widehat{w}_0 , the whole line would move up and the best value of \widehat{w}_1 would have to be decreased slightly (thereby producing a steeper negative gradient) to pass as close as possible to all of the data points.

Another way to get a feeling for the meaning of $\text{cov}\{\widehat{\mathbf{w}}\}$ is to look at the variability in models that it suggests. To do this, we can assume that $\widehat{\mathbf{w}}$ is a random variable with a Gaussian distribution

$$\mathbf{w} \sim \mathcal{N}(\widehat{\mathbf{w}}, \text{cov}\{\widehat{\mathbf{w}}\}). \quad (2.48)$$

From this density, we can sample several instances of \mathbf{w} and plot the resulting models. An example of 10 instances is shown in Figure 2.16. We can see that there is very little change in gradient (w_1) across the 10 samples but that this small gradient change would, if we extrapolated back to year zero, result in quite a large change of w_0 . This is reflected by the values in $\text{cov}\{\widehat{\mathbf{w}}\}$, as already discussed. The idea of having a distribution over model parameters rather than a single *best* value is very important in Machine Learning and is introduced in the next chapter.

2.10 Variability in predictions

In Chapter 1 we made some predictions about 100 m winning times in future Olympics. We argued that these predictions were not very useful as they took the form of exact values. It would seem more sensible to predict a range of values in which we think the winning time might fall. If we are quite certain about our prediction,

this range might be small; if we are less certain, it might be large. So, as well as obtaining an indication of the variability of our parameter estimate, $\hat{\mathbf{w}}$, it makes sense to provide indications of any variability or uncertainty in our predictions. Suppose we observe a new set of attributes, \mathbf{x}_{new} . We would like to predict the output t_{new} and in addition, the variability associated with this output, σ_{new}^2 .

To predict t_{new} , we multiply \mathbf{x}_{new} by the best set of model parameters, $\hat{\mathbf{w}}$:

$$t_{\text{new}} = \hat{\mathbf{w}}^\top \mathbf{x}_{\text{new}}. \quad (2.49)$$

To check that this is sensible, we can compute its expectation:

$$\begin{aligned} \mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \{t_{\text{new}}\} &= \mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \{\hat{\mathbf{w}}\}^\top \mathbf{x}_{\text{new}} \\ &= \mathbf{w}^\top \mathbf{x}_{\text{new}} \end{aligned}$$

where we have used Equation 2.40. The expected value of our prediction is the new data attribute multiplied by the *true* \mathbf{w} . In Section 2.2.8 we derived a general expression for variance. In our case, this is:

$$\sigma_{\text{new}}^2 = \text{var}\{t_{\text{new}}\} = \mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \{t_{\text{new}}^2\} - (\mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \{t_{\text{new}}\})^2$$

To evaluate this expression, we need to first substitute $t_{\text{new}} = \hat{\mathbf{w}}^\top \mathbf{x}_{\text{new}}$:

$$\begin{aligned} \text{var}\{t_{\text{new}}\} &= \mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ (\hat{\mathbf{w}}^\top \mathbf{x}_{\text{new}})^2 \right\} - (\mathbf{w}^\top \mathbf{x}_{\text{new}})^2 \\ &= \mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \mathbf{x}_{\text{new}}^\top \hat{\mathbf{w}} \hat{\mathbf{w}}^\top \mathbf{x}_{\text{new}} \right\} - \mathbf{x}_{\text{new}}^\top \mathbf{w} \mathbf{w}^\top \mathbf{x}_{\text{new}}. \end{aligned}$$

Substituting our now familiar expression for $\hat{\mathbf{w}}$:

$$\text{var}\{t_{\text{new}}\} = \mathbf{x}_{\text{new}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{E}_{p(t|\mathbf{X}, \mathbf{w}, \sigma^2)} \{t t^\top\} \mathbf{X} (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_{\text{new}} - \mathbf{x}_{\text{new}}^\top \mathbf{w} \mathbf{w}^\top \mathbf{x}_{\text{new}}.$$

Using the expression for the $\text{cov}\{t\}$ (Equation 2.43) allows us to compute the expectation and simplify the expression:

$$\begin{aligned} \text{var}\{t_{\text{new}}\} &= \mathbf{x}_{\text{new}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top (\sigma^2 \mathbf{I} + \mathbf{X} \mathbf{w} \mathbf{w}^\top \mathbf{X}^\top) \mathbf{X} (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_{\text{new}} - \mathbf{x}_{\text{new}}^\top \mathbf{w} \mathbf{w}^\top \mathbf{x}_{\text{new}} \\ &= \sigma^2 \mathbf{x}_{\text{new}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_{\text{new}} + \mathbf{x}_{\text{new}}^\top \mathbf{w} \mathbf{w}^\top \mathbf{x}_{\text{new}} - \mathbf{x}_{\text{new}}^\top \mathbf{w} \mathbf{w}^\top \mathbf{x}_{\text{new}} \\ &= \sigma^2 \mathbf{x}_{\text{new}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_{\text{new}}. \end{aligned}$$

Note that by substituting our expression for $\text{cov}\{\hat{\mathbf{w}}\}$ (Equation 2.41), this expression can be re-written as

$$\sigma_{\text{new}}^2 = \mathbf{x}_{\text{new}}^\top \text{cov}\{\hat{\mathbf{w}}\} \mathbf{x}_{\text{new}}.$$

To summarise, our prediction and associated variance are given as:

$$t_{\text{new}} = \mathbf{x}_{\text{new}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} = \mathbf{x}_{\text{new}}^\top \hat{\mathbf{w}} \quad (2.50)$$

$$\sigma_{\text{new}}^2 = \sigma^2 \mathbf{x}_{\text{new}}^\top (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}_{\text{new}}. \quad (2.51)$$

σ^2 is the *true* variance of the dataset noise. In its place, we can use our estimate, $\widehat{\sigma^2}$.

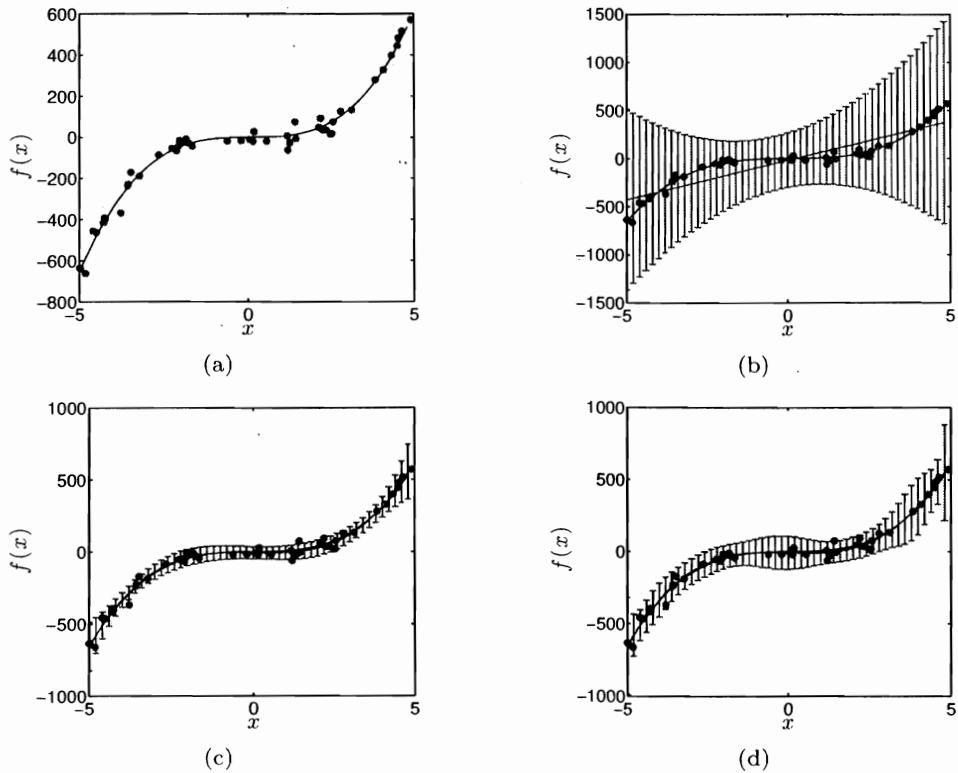


FIGURE 2.17: (a) Example data set. (b), (c) and (d) Predictive error bars for a linear, cubic and 6th order model, respectively.

2.10.1 Predictive variability – an example

Figure 2.17(a) shows the function $f(x) = 5x^3 - x^2 + x$ and datapoints sampled from this function and corrupted by Gaussian noise with mean zero and variance 1000. In Figures 2.17(b), 2.17(c) and 2.17(d) we can see $t_{\text{new}} \pm \sigma_{\text{new}}^2$ for linear, cubic and 6th order models, respectively (MATLAB script: `predictive_variance_example.m`).

The linear model has very high predictive variance. It is unable to model the deterministic trend in the data very well and much of the variability of the data is assumed to be noise. The cubic model is better able to model the trend (it is the correct order) and this is reflected in its much more confident predictions. The 6th order model is overly complex – it has too much freedom and can therefore fit the data well for quite a large range of parameter values. This uncertainty in $\hat{\mathbf{w}}$ feeds through to increased predictive variability – if we are less sure on the parameter values, we're going to be less sure of the predictions too. This point can be demonstrated by computing $\text{cov}\{\hat{\mathbf{w}}\}$ for the 3rd and 6th order models and then sampling functions just as we did in Section 2.9.3. Figure 2.18 shows 20 functions drawn from a Gaussian with mean $\hat{\mathbf{w}}$ and covariance $\text{cov}\{\hat{\mathbf{w}}\}$ for the 3rd and 6th order models (the plot is zoomed into a small region of x and the darker line shows the true

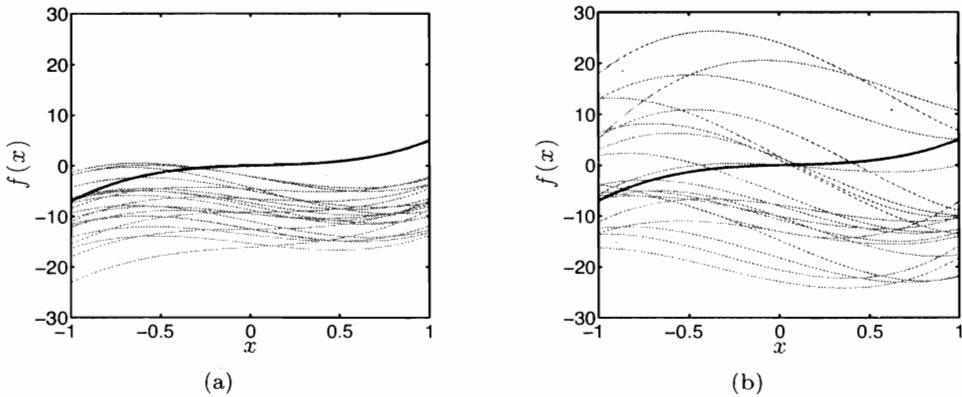


FIGURE 2.18: Examples of functions with parameters drawn from a Gaussian with mean $\hat{\mathbf{w}}$ and covariance $\text{cov}\{\hat{\mathbf{w}}\}$ for the example data set shown in Figure 2.17(a).

function) (MATLAB script: `predictive_variance_example.m`). The increased variability in possible functions caused by the increase in parameter uncertainty is clear for the 6th order model.

A final interesting point is that for all models, the predictive variance increases as we move towards the edge of the data. The model is less confident in areas where it has less data – an appealing property. In Chapter 1 we pointed out that making point predictions indefinitely into the future (i.e. beyond the range of the training data) was not very sensible. We now have a model that will make predictions beyond the range of the training data, but will do so with increasing uncertainty, which is likely to be more useful. We also observe this effect towards the centre of the data (particularly Figure 2.17(d)) where there is a small gap (not many instances at around $x = 1$). Exercise EX 2.12 gives you the opportunity to investigate this effect further.

2.10.2 Expected values of the estimators

In Section 2.9.1 we computed the expected value of our estimate $\hat{\mathbf{w}}$. This expectation is taken with respect to the generating density $p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2\mathbf{I})$ and is repeated here:

$$\begin{aligned}\mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \{\hat{\mathbf{w}}\} &= \mathbf{E}_{p(\mathbf{t}|\mathbf{X})} \left\{ (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} \right\} \\ &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{E}_{p(\mathbf{t}|\mathbf{X})} \{\mathbf{t}\} \\ &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X}\mathbf{w} \\ &= \mathbf{I}\mathbf{w} = \mathbf{w}\end{aligned}$$

where we have used the expression for $\hat{\mathbf{w}}$ ($\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t}$) and the fact that the expected value of a Gaussian random variable (\mathbf{t}) is equal to the mean of the Gaussian ($\mathbf{X}\mathbf{w}$). So, the expected value of our estimate, $\hat{\mathbf{w}}$, is the true value, \mathbf{w} . This is an important property of $\hat{\mathbf{w}}$; it tells us that $\hat{\mathbf{w}}$ is an unbiased estimator

– it is neither consistently too high nor too low. Another way of thinking about this is to think back to the experiment at the start of Section 2.9. There, for a set of attributes x_1, \dots, x_N , we generated many sets of responses and looked at how much influence different particular noise values had on $\hat{\mathbf{w}}$. Because $\hat{\mathbf{w}}$ is unbiased, it should, on average, be correct. So, if we took the average of all of the different $\hat{\mathbf{w}}$ values obtained in our experiment, it should be very close to the truth. In fact, taking this average we get $\hat{w}_0 = -2.0007$ and $\hat{w}_1 = 3.0008$, which are both very close to the true values: $w_0 = -2, w_1 = 3$.

We can do the same for the estimate of the noise variance, $\widehat{\sigma^2}$. Recall from Equation 2.37 the expression for $\widehat{\sigma^2}$:

$$\widehat{\sigma^2} = \frac{1}{N}(\mathbf{t}^\top \mathbf{t} - \mathbf{t}^\top \mathbf{X} \hat{\mathbf{w}}).$$

Taking the expectation with respect to $p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)$ and doing some manipulation gives:

$$\begin{aligned} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \widehat{\sigma^2} \right\} &= \frac{1}{N} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \mathbf{t}^\top \mathbf{t} - \mathbf{t}^\top \mathbf{X} \hat{\mathbf{w}} \right\} \\ &= \frac{1}{N} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \mathbf{t}^\top \mathbf{t} - \mathbf{t}^\top \mathbf{X} (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} \right\} \\ &= \frac{1}{N} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \mathbf{t}^\top \mathbf{t} \right\} \\ &\quad - \frac{1}{N} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \mathbf{t}^\top \mathbf{X} (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{t} \right\}. \end{aligned} \tag{2.52}$$

Comment 2.8 – Matrix trace: The trace of a square matrix \mathbf{A} , denoted $\text{Tr}(\mathbf{A})$, is the sum of the diagonal elements of \mathbf{A} . For example, if

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1D} \\ A_{21} & A_{22} & \cdots & A_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ A_{D1} & A_{D2} & \cdots & A_{DD} \end{bmatrix}$$

then

$$\text{Tr}(\mathbf{A}) = \sum_{d=1}^D A_{dd}.$$

It follows that if $\mathbf{A} = \mathbf{I}_D$, i.e. the $D \times D$ identity matrix,

$$\text{Tr}(\mathbf{I}_D) = \sum_{d=1}^D 1 = D.$$

A useful identity that we will often use is that

$$\text{Tr}(\mathbf{AB}) = \text{Tr}(\mathbf{BA}).$$

Comment 2.8 - Matrix trace (continued): Also, the trace of a scalar is just equal to the scalar value (a scalar could be thought of as a 1×1 matrix), i.e.

$$\text{Tr}(a) = a$$

or, if $\mathbf{w} = [w_1, \dots, w_D]^\top$,

$$\text{Tr}(\mathbf{w}^\top \mathbf{w}) = \mathbf{w}^\top \mathbf{w}$$

because the result of $\mathbf{w}^\top \mathbf{w}$ is a scalar.

We have seen the expectation of the form $\mathbf{t}\mathbf{t}^\top$ before but not $\mathbf{t}^\top \mathbf{t}$ ($= \mathbf{t}^\top \mathbf{I}_N \mathbf{t}$) or $\mathbf{t}^\top \mathbf{A}\mathbf{t}$. When \mathbf{t} is a Gaussian random variable, expectations of the form $\mathbf{t}^\top \mathbf{A}\mathbf{t}$ are given by:

$$\begin{aligned} \mathbf{t} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ \mathbf{E}_{p(\mathbf{t})} \left\{ \mathbf{t}^\top \mathbf{A}\mathbf{t} \right\} &= \text{Tr}(\mathbf{A}\boldsymbol{\Sigma}) + \boldsymbol{\mu}^\top \mathbf{A}\boldsymbol{\mu}, \end{aligned}$$

where $\text{Tr}()$ is the trace function (see Comment 2.8). For the first term on the right hand side of Equation 2.52, $\mathbf{A} = \mathbf{I}_N$ (note that $\mathbf{t}^\top \mathbf{t} = \mathbf{t}^\top \mathbf{I}_N \mathbf{t}$, where \mathbf{I}_N is the $N \times N$ identity matrix) and in the second term, $\mathbf{A} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$. In both cases, $\boldsymbol{\mu} = \mathbf{X}\mathbf{w}$ and $\boldsymbol{\Sigma} = \sigma^2 \mathbf{I}_N$. Substituting the necessary values into Equation 2.52 gives

$$\begin{aligned} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \widehat{\sigma^2} \right\} &= \frac{1}{N} \left(\text{Tr}(\sigma^2 \mathbf{I}_N) + \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} \right) \\ &\quad - \frac{1}{N} \left(\text{Tr}(\sigma^2 \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top) + \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X} \mathbf{w} \right), \end{aligned}$$

since $\mathbf{I}_N \mathbf{I}_N = \mathbf{I}_N$. Now, $\text{Tr}(\sigma^2 \mathbf{A}) = \sigma^2 \text{Tr}(\mathbf{A})$ and $\text{Tr}(\mathbf{I}_N) = N$ by definition. Using these, we can simplify the expression to:

$$\begin{aligned} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \widehat{\sigma^2} \right\} &= \sigma^2 + \frac{1}{N} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} - \frac{\sigma^2}{N} \text{Tr}(\mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top) - \frac{1}{N} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} \\ &= \sigma^2 - \frac{\sigma^2}{N} \text{Tr}(\mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top) \\ &= \sigma^2 \left(1 - \frac{1}{N} \text{Tr}(\mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top) \right). \end{aligned}$$

Finally, we need to use the fact that $\text{Tr}(\mathbf{AB}) = \text{Tr}(\mathbf{BA})$ and therefore the first \mathbf{X} inside the trace function can be moved to be the last:

$$\begin{aligned} \mathbf{E}_{p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2)} \left\{ \widehat{\sigma^2} \right\} &= \sigma^2 \left(1 - \frac{1}{N} \text{Tr}((\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{X}) \right) \\ &= \sigma^2 \left(1 - \frac{1}{N} \text{Tr}(\mathbf{I}_D) \right) \\ &= \sigma^2 \left(1 - \frac{D}{N} \right), \end{aligned} \tag{2.53}$$

where D is the number of attributes (the number of columns in \mathbf{X}).

Assuming that $D < N$ (i.e. the number of attributes we measure for each data point is smaller than the number of data points), then our estimate of the variance

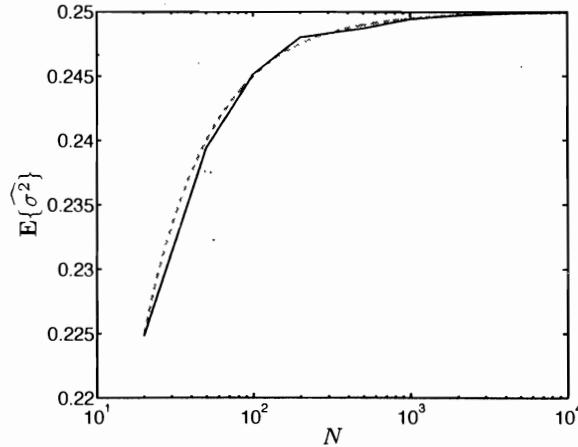


FIGURE 2.19: Evolution of the theoretical and empirical estimates of $E_{p(t|X, w, \sigma^2)} \{\widehat{\sigma}^2\}$ as the number of data points increases.

will, on average, be lower than the true variance:

$$E_{p(t|X, w, \sigma^2)} \{\widehat{\sigma}^2\} < \sigma^2.$$

Unlike \widehat{w} , this estimator is **biased**.

We can see this bias by returning to our synthetic experiment. The average value of $\widehat{\sigma}^2$ over all of the datasets was 0.2264. The true value, $\sigma^2 = 0.5^2 = 0.25$. We can see that the average value is indeed too low. For this example, $D = 2$ and $N = 20$, so our theoretical expected value is

$$E_{p(t|X, w, \sigma^2)} \{\widehat{\sigma}^2\} = \sigma^2 \left(1 - \frac{D}{N}\right) = 0.25 \left(1 - \frac{2}{20}\right) = 0.2250$$

which is close to the observed average.

From Equation 2.53, we notice that one way to decrease the bias is to make D/N smaller. D is normally fixed, but we can increase N . In Figure 2.19 we can see the effect of increasing N from 20 to 10,000 (MATLAB script: `w_variation_demo.m`). The theoretical (dashed) curve and the empirical (solid) curve (created by re-running our previous experiment with different numbers of observations, N) are in close agreement and converge towards the true value of $\sigma^2 = 0.25$ as the amount of data increases.

It is possible to provide an intuitive explanation for the bias in $\widehat{\sigma}^2$. The expression for the ML estimate of σ^2 is:

$$\widehat{\sigma}^2 = \frac{1}{N} (\mathbf{t}^\top \mathbf{t} - \mathbf{t}^\top \mathbf{X} \widehat{\mathbf{w}}). \quad (2.54)$$

It is possible to rearrange this to be equal to the sum of squared errors between the predictions and the true responses (see Exercise EX 2.11):

$$\widehat{\sigma}^2 = \frac{1}{N} \sum_{n=1}^N (t_n - \mathbf{x}^\top \widehat{\mathbf{w}})^2.$$

This tells us that the closer the model gets to the data, the smaller $\widehat{\sigma^2}$. Now imagine the true value of \mathbf{w} and our estimate $\widehat{\mathbf{w}}$. Which will get closer to the data? The maximum likelihood estimate, $\widehat{\mathbf{w}}$, is identical to the minimum loss estimate. It is, by definition, the set of parameters that gets closest to the data and therefore minimises $\widehat{\sigma^2}$. The value of $\widehat{\sigma^2}$ that we would get if we used the true value \mathbf{w} instead of $\widehat{\mathbf{w}}$ in Equation 2.54 would have to be the same or higher than the value we get with $\widehat{\mathbf{w}}$. Because we are finding the value of \mathbf{w} that minimises the noise we will, on average, end up with a lower level of noise than the true value.

2.10.3 Summary

In the preceding sections, we have covered a lot of material. An introduction to random variables provided the foundations required to be able to model the errors between the data and the proposed deterministic model. By explicitly modelling these errors, we have seen how the least squares solution from Chapter 1 is equivalent to the solution obtained by maximising a different quantity called the *likelihood* if the noise in the data is assumed to be normally distributed. The benefit of the likelihood approach is the ability to quantify the uncertainty in our parameter estimates and hence also, crucially, in our predictions. This allows us to move away from exact predictions (which will certainly be wrong) to ranges of values (e.g. $t_{\text{new}} \pm \sigma_{\text{new}}^2$). In most applications this will be much more useful. Finally, we looked at some theoretical properties of the maximum likelihood parameter values and saw that although our estimate $\widehat{\mathbf{w}}$ is unbiased, $\widehat{\sigma^2}$ is, on average, biased to be too low.

2.11 Exercises

- EX 2.1. Would the errors in the 100 m linear regression (shown in Figure 2.1) be best modelled with a discrete or continuous random variable?
- EX 2.2. By using the fact that when rolling a die, all outcomes are equally likely and by using the constraints given in Equations 2.1 and 2.2, compute the probabilities of the dice landing with each of the six faces facing up.
- EX 2.3. Y is a random variable that can take any positive integer value. The likelihood of these outcomes is given by the Poisson pdf

$$p(y) = \frac{\lambda^y}{y!} \exp\{-\lambda\}.$$

By using the fact that for a discrete random variable the pdf gives the probabilities of the individual events occurring and that probabilities are additive, (a) compute the probability that $Y \leq 4$ for $\lambda = 5$, i.e. $P(Y \leq 4)$. (b) Using the result of (a) and the fact that one outcome has to happen, compute the probability that $Y > 4$. (Hint, one of the two events, $Y \leq 4$ and $Y > 4$, *has to happen*.)

- EX 2.4. Y is a random variable with a uniform density, $p(y) = \mathcal{U}(a, b)$. Derive $\mathbf{E}_{p(y)} \{\sin(y)\}$. Note that $\int \sin(y) dy = -\cos(y)$. Compute $\mathbf{E}_{p(y)} \{\sin(y)\}$

for $a = 0, b = 1$. Modify `approx_expected_value.m` to compute a sample-based approximation to this value and observe how the approximation improves with the number of samples drawn.

- EX 2.5. Assume that $p(\mathbf{w})$ is the Gaussian pdf for a D -dimensional vector \mathbf{w} given in Equation 2.28. By expanding the vector notation and re-arranging, show that using $\Sigma = \sigma^2 \mathbf{I}$ as the covariance matrix, assumes independence of the D elements of \mathbf{w} . You will need to be aware that the determinant of a matrix that only has entries on the diagonal ($|\sigma^2 \mathbf{I}|$) is the product of the diagonal values and that the inverse of the same matrix is constructed by simply inverting each element on the diagonal. (Hint, a product of exponentials can be expressed as an exponential of a sum).
- EX 2.6. Using the same setup as Exercise EX 2.5 above, see what happens if we use a diagonal covariance matrix with different elements on the diagonal, i.e.

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_D^2 \end{bmatrix}$$

- EX 2.7. Show that for a 1st order polynomial, the diagonal elements of the Hessian matrix of second derivatives of the log likelihood is equivalent to (they will differ by a multiplicative constant) the second derivatives in Equation 1.9.
- EX 2.8. Assume that a dataset of N values, x_1, \dots, x_N , was sampled from a Gaussian distribution. Assuming that the data are IID, find the maximum likelihood estimate of the Gaussian mean and variance. (Hint, start by writing down the combined likelihood of all N data-points and note that the product of an exponential function can be written as the exponential of a sum.).
- EX 2.9. Assume that a dataset of N binary values, x_1, \dots, x_N , was sampled from a Bernoulli distribution. Compute the maximum likelihood estimate for the Bernoulli parameter.
- EX 2.10. Obtain the maximum likelihood estimates of the mean vector and covariance matrix of a multivariate Gaussian density given N observations $\mathbf{x}_1, \dots, \mathbf{x}_N$.
- EX 2.11. Show that the maximum likelihood estimate of the noise variance in our linear model,

$$\widehat{\sigma^2} = \frac{1}{N} (\mathbf{t}^\top \mathbf{t} - \mathbf{t}^\top \mathbf{X} \widehat{\mathbf{w}}),$$

can also be expressed as

$$\widehat{\sigma^2} = \frac{1}{N} \sum_{n=1}^N (t_n - \mathbf{x}_n^\top \mathbf{w})^2.$$

(Hint, work backwards from the second expression.)

- EX 2.12. Using `predictive_variance_example.m`, generate a dataset and remove all values for which $-1.5 \leq x \leq 1.5$. Observe the effect this has on the predictive variance in this range.

- EX 2.13. Compute the Fisher Information Matrix for the parameter of a Bernoulli distribution.
- EX 2.14. Compute the Fisher Information Matrix for the components of the mean vector in a multivariate Gaussian density.

Further reading

- [1] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2007.

This book is an excellent resource for many machine learning concepts. In particular, it includes a detailed discussion of the bias-variance trade-off.

- [2] J.H. McColl. *Probability*. Elsevier, 1995.

A very accessible introduction to probability theory.

- [3] Paul Meyer. *Introductory Probability and Statistical Applications*. Addison-Wesley, 1978.

An excellent resource for introductory probability theory.

- [4] J. Rosenthal. *A First Look at Rigorous Probability Theory*. World Scientific Publishing Company, 2006.

This is a very accessible book to begin exploring measure theory – the branch of mathematics that underpins probability theory.

- [5] Michael Tipping and Christopher Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 61(3):611–622, 1999.

An interesting application of maximum likelihood. Here it is applied to one of the first probabilistic approaches to the the classical statistical problem of Principal Components Analysis.

Chapter 3

The Bayesian Approach to Machine Learning

In the previous chapter, we saw how explicitly adding noise to our model allowed us to obtain more than just point predictions. In particular, we were able to quantify the uncertainty present in our parameter estimates and our subsequent predictions. Once content with the idea that there will be uncertainty in our parameter estimates, it is a small step towards considering our parameters themselves as random variables. Bayesian methods are becoming increasingly important within Machine Learning and we will devote the next two chapters to providing an introduction to an area that many people find challenging. In this chapter, we will cover some of the fundamental ideas of Bayesian statistics through two examples. Unfortunately, the calculations required to perform Bayesian inference are often not analytically tractable. In Chapter 4 we will introduce three approximation methods that are popular in the machine learning community.

3.1 A coin game

Imagine you are walking around a fairground and come across a stall where customers are taking part in a coin tossing game. The stall owner tosses a coin 10 times for each customer. If the coin lands heads on six or fewer occasions, the customer wins back their £1 stake plus an additional £1. Seven or more and the stall owner keeps their money. The binomial distribution (described in Section 2.3.2) describes the probability of a certain number of successes (heads) in N binary events. The probability of y heads from N tosses where each toss lands heads with probability r is given by:

$$P(Y = y) = \binom{N}{y} r^y (1 - r)^{N-y}. \quad (3.1)$$

You assume that the coin is fair and therefore set $r = 0.5$. For $N = 10$ tosses, the probability distribution function can be seen in Figure 3.1, where the bars corresponding to $y \leq 6$ have been shaded. Using Equation 3.1, it is possible to calculate the probability of winning the game, i.e. the probability that Y is less than or equal

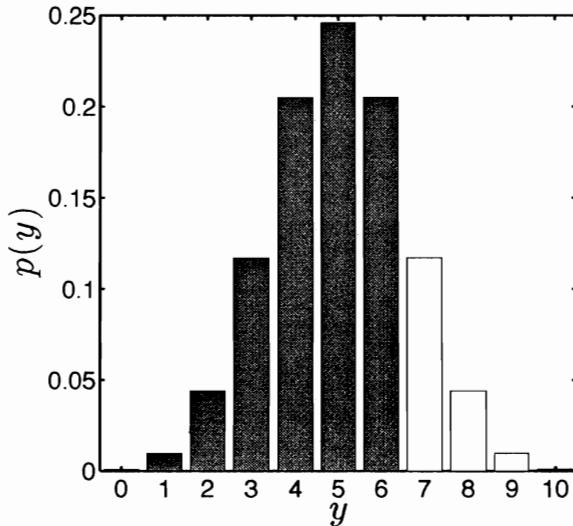


FIGURE 3.1: The binomial density function (Equation 3.1) when $N = 10$ and $r = 0.5$.

to 6, $P(Y \leq 6)$:

$$\begin{aligned}
 P(Y \leq 6) &= 1 - P(Y > 6) = 1 - [P(Y = 7) + P(Y = 8) + P(Y = 9) \\
 &\quad + P(Y = 10)] \\
 &= 1 - [0.1172 + 0.0439 + 0.0098 + 0.0010] \\
 &= 0.8281.
 \end{aligned}$$

This seems like a pretty good game – you’ll double your money with probability 0.8281. It is also possible to compute the expected return from playing the game. The expected value of a function $f(X)$ of a random variable X is computed as (introduced in Section 2.2.8):

$$\mathbf{E}_{P(x)} \{f(X)\} = \sum_x f(x)P(x),$$

where the summation is over all possible values that the random variable can take. Let X be the random variable that takes a value of 1 if we win and a value of 0 if we lose: $P(X = 1) = P(Y \leq 6)$. If we win ($X = 1$), we get a return of £2 (our original stake plus an extra £1) so $f(1) = 2$. If we lose, we get a return of nothing so $f(0) = 0$. Hence our expected return is

$$f(1)P(X = 1) + f(0)P(X = 0) = 2 \times P(Y \leq 6) + 0 \times P(Y > 6) = 1.6562.$$

Given that it costs £1 to play, you win, on average, $1.6562 - 1$ or approximately 66p per game. If you played 100 times, you’d expect to walk away with a profit of £65.62.

Given these odds of success, it seems sensible to play. However, whilst waiting you notice that the stall owner looks reasonably wealthy and very few customers seem to

be winning. Perhaps the assumptions underlying the calculations are wrong. These assumptions are:

1. The number of heads can be modelled as a random variable with a binomial distribution and the probability of a head on any particular toss is r .
2. The coin is fair – the probability of heads is the same as the probability of tails – $r = 0.5$.

It seems hard to reject the binomial distribution – events are taking place with only two possible outcomes and the tosses do seem to be independent. This leaves r , the probability that the coin lands heads. Our assumption was that the coin was fair – the probability of heads was equal to the probability of tails. Maybe this is not the case? To investigate this, we can treat r as a parameter (like w and σ^2 in the previous chapter) and fit it to some data.

3.1.1 Counting heads

There are three people in the queue to play. The first one plays and gets the following sequence of heads and tails:

H,T,H,H,H,H,H,H,H,H,

nine heads and one tail. It is possible to compute the maximum likelihood value of r as follows. The likelihood is given by the binomial distribution:

$$P(Y = y|r, N) = \binom{N}{y} r^y (1 - r)^{N-y}. \quad (3.2)$$

Taking the natural logarithm gives:

$$L = \log P(Y = y|r, N) = \log \binom{N}{y} + y \log r + (N - y) \log(1 - r).$$

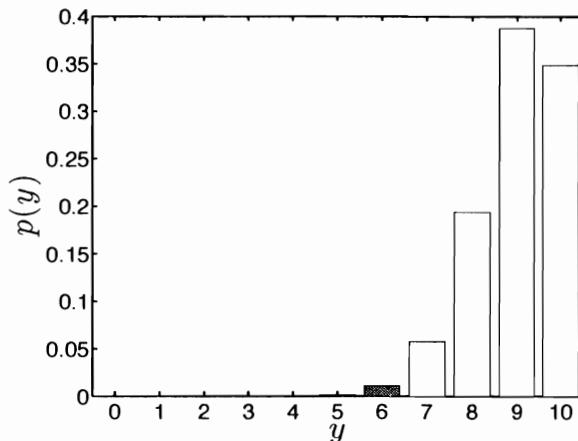


FIGURE 3.2: The binomial density function (Equation 3.1) when $N = 10$ and $r = 0.9$.

As in Chapter 2, we can differentiate this expression, equate to zero and solve for the maximum likelihood estimate of the parameter:

$$\begin{aligned}\frac{\partial L}{\partial r} &= \frac{y}{r} - \frac{N-y}{1-r} = 0 \\ y(1-r) &= r(N-y) \\ y &= rN \\ r &= \frac{y}{N}.\end{aligned}$$

Substituting $y = 9$ and $N = 10$ gives $r = 0.9$. The corresponding distribution function is shown in Figure 3.2 and the re-calculated probability of winning is $P(Y \leq 6) = 0.0128$. This is much lower than that for $r = 5$. The expected return is now

$$2 \times P(Y \leq 6) + 0 \times P(Y > 6) = 0.0256.$$

Given that it costs £1 to play, we expect to make $0.0256 - 1 = -0.9744$ per game – a loss of approximately 97p. $P(Y \leq 6) = 0.0128$ suggests that only about 1 person in every 100 should win, but this does not seem to be reflected in the number of people who *are* winning. Although the evidence from this run of coin tosses suggests $r = 0.9$, it seems too biased given that several people *have* won.

3.1.2 The Bayesian way

The value of r computed in the previous section was based on just 10 tosses. Given the random nature of the coin toss, if we observed several sequences of tosses it is likely that we would get a different r each time. Thought about this way, r feels a bit like a random variable, R . Maybe we can learn something about the distribution of R rather than try and find a particular value. We saw in the previous section that obtaining an exact value by counting is heavily influenced by the particular tosses in the short sequence. No matter how many such sequences we observe there will always be some uncertainty in r – considering it as a random variable with an associated distribution will help us measure and understand this uncertainty.

In particular, defining the random variable Y_N to be the number of heads obtained in N tosses, we would like the distribution of r conditioned on the value of Y_N :

$$p(r|y_N).$$

Given this distribution, it would be possible to compute the expected probability of winning by taking the expectation of $P(Y_{\text{new}} \leq 6|r)$ with respect to $p(r|y_N)$:

$$P(Y_{\text{new}} \leq 6|y_N) = \int P(Y_{\text{new}} \leq 6|r)p(r|y_N)dr,$$

where Y_{new} is a random variable describing the number of heads in a future set of 10 tosses.

In Section 2.2.7 we gave a brief introduction to Bayes' rule. Bayes' rule allows us to reverse the conditioning of two (or more) random variables, e.g. compute $p(a|b)$ from $p(b|a)$. Here we're interested in $p(r|y_N)$, which, if we reverse the conditioning, is $p(y_N|r)$ – the probability distribution function over the number of heads in N independent tosses where the probability of a head in a single toss is r . This is the

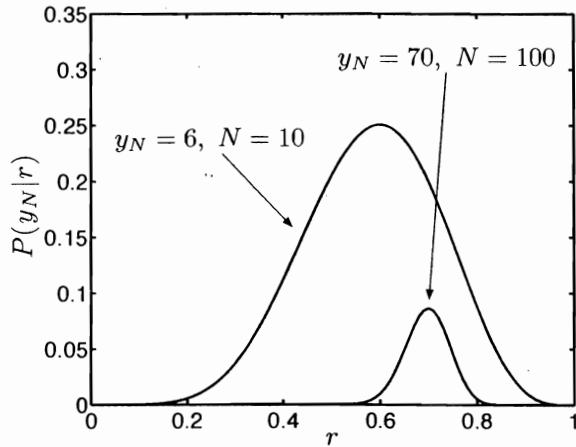


FIGURE 3.3: Examples of the likelihood $p(y_N|r)$ as a function of r for two scenarios.

binomial distribution function that we can easily compute for any y_N and r . In our context, Bayes' rule is (see also Equation 2.11):

$$p(r|y_N) = \frac{P(y_N|r)p(r)}{P(y_N)}. \quad (3.3)$$

This equation is going to be very important for us in the following chapters so it is worth spending some time looking at each term in detail.

The likelihood, $P(y_N|r)$: We came across likelihood in Chapter 2. Here it has exactly the same meaning: how likely is it that we would observe our data (in this case, the data is y_N) for a particular value of r (our model). For our example, this is the binomial distribution. This value will be high if r could have feasibly produced the result y_N and low if the result is very unlikely. For example, Figure 3.3 shows the likelihood $P(y_N|r)$ as a function of r for two different scenarios. In the first, the data consists of 10 tosses ($N = 10$), of which 6 were heads. In the second, there were $N = 100$ tosses, of which 70 were heads.

This plot reveals two important properties of the likelihood. First, it is not a probability density. If it were, the area under both curves would have to equal 1. We can see that this is not the case without working out the area because the two areas are completely different. Second, the two examples differ in how much they appear to tell us about r . In the first example, the likelihood has a non-zero value for a large range of possible r values (approximately $0.2 \leq r \leq 0.9$). In the second, this range is greatly reduced (approximately $0.6 \leq r \leq 0.8$). This is very intuitive: in the second example, we have much more data (the results of 100 tosses rather than 10) and so we *should* know more about r .

The prior distribution, $p(r)$: The prior distribution allows us to express any belief we have in the value of r *before* we see any data. To illustrate this, we shall consider the following three examples:

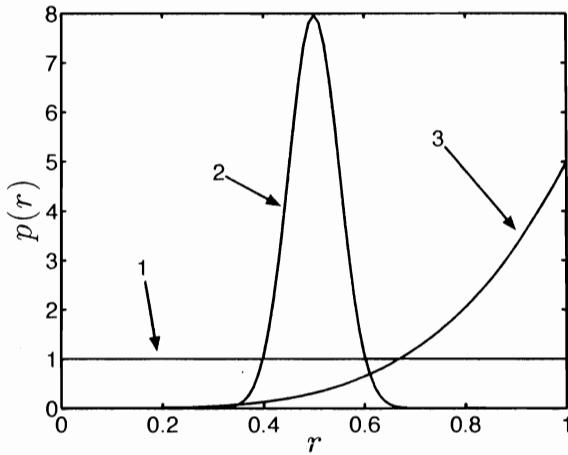


FIGURE 3.4: Examples of prior densities, $p(r)$, for r for three different scenarios.

1. We do not know anything about tossing coins or the stall owner.
2. We think the coin (and hence the stall owner) is fair.
3. We think the coin (and hence the stall owner) is biased to give more heads than tails.

We can encode each of these beliefs as different prior distributions. r can take any value between 0 and 1 and therefore it must be modelled as a continuous random variable. Figure 3.4 shows three density functions that might be used to encode our three different prior beliefs.

Belief number 1 is represented as a uniform density between 0 and 1 and as such shows no preference for any particular r value. Number 2 is given a density function that is concentrated around $r = 0.5$, the value we would expect for a fair coin. The density suggests that we do not expect much variance in r : it's almost certainly going to lie between 0.4 and 0.6. Most coins that any of us have tossed agree with this. Finally, number 3 encapsulates our belief that the coin (and therefore the stall owner) is biased. This density suggests that $r > 0.5$ and that there is a high level of variance. This is fine because our belief is just that the coin is biased: we don't really have any idea how biased at this stage.

We will not choose between our three scenarios at this stage as it is interesting to see the effect these different beliefs will have on $p(r|y_N)$.

The three functions shown in Figure 3.4 have not been plucked from thin air. They are all examples of beta probability density functions (see Section 2.5.2). The beta density function is used for continuous random variables constrained to lie between 0 and 1 – perfect for our example. For a random variable R with parameters α and β it is defined as:

$$p(r) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1} (1-r)^{\beta-1}. \quad (3.4)$$

$\Gamma(a)$ is known as the gamma function (see Section 2.5.2). In Equation 3.4 the gamma

functions ensure that the density is normalised (that is, it integrates to 1 and is therefore a probability density function). In particular:

$$\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)} = \int_{r=0}^{r=1} r^{\alpha-1}(1-r)^{\beta-1} dr,$$

ensuring that

$$\int_{r=0}^{r=1} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1}(1-r)^{\beta-1} dr = 1.$$

The two parameters α and β control the shape of the resulting density function and must both be positive. Our three beliefs as plotted in Figure 3.4 correspond to the following pairs of parameter values:

1. Know nothing: $\alpha = 1, \beta = 1$.
2. Fair coin: $\alpha = 50, \beta = 50$.
3. Biased: $\alpha = 5, \beta = 1$.

The problem of choosing these values is a big one. For example, why should we choose $\alpha = 5, \beta = 1$ for a biased coin? There is no easy answer to this. We shall see later that, for the beta distribution, they can be interpreted as a number of previous, hypothetical coin tosses. For other distributions no such analogy is possible and we will also introduce the idea that maybe these too should be treated as random variables. In the mean time, we will assume that these values are sensible and move on.

The marginal distribution of $y_N - P(y_N)$: The third quantity in our equation, $P(y_N)$, acts as a normalising constant to ensure that $p(r|y_N)$ is a properly defined density. It is known as the marginal distribution of y_N because it is computed by integrating r out of the joint density $p(y_N, r)$:

$$P(y_N) = \int_{r=0}^{r=1} p(y_N, r) dr.$$

This joint density can be factorised to give:

$$P(y_N) = \int_{r=0}^{r=1} P(y_N|r)p(r) dr,$$

which is the product of the prior and likelihood integrated over the range of values that r may take.

$p(y_N)$ is also known as the **marginal likelihood** as it is the likelihood of the data, y_N , averaged over all parameter values. We shall see in Section 3.4.1 that it can be a useful quantity in model selection but unfortunately, in all but a small minority of cases, it is very difficult to calculate.

The posterior distribution – $p(r|y_N)$: This **posterior** is the distribution in which we are interested. It is the result of updating our prior belief $p(r)$ in light of new evidence y_N . The shape of the density is interesting – it tells us something about how much information we have about r after combining what we knew beforehand

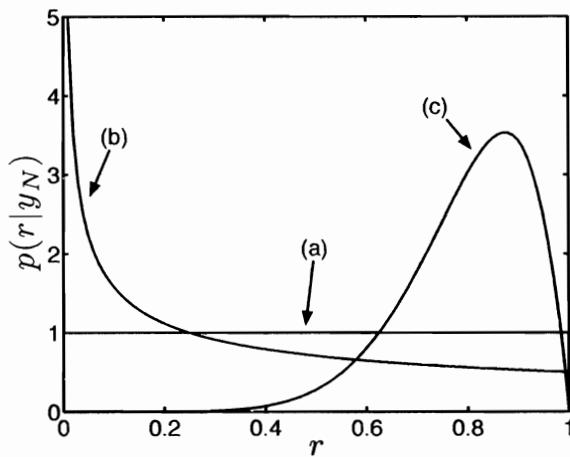


FIGURE 3.5: Examples of three possible posterior distributions $p(r|y_N)$.

(the prior) and what we've seen (the likelihood). Three hypothetical examples are provided in Figure 3.5 (these are purely illustrative and do not correspond to the particular likelihood and prior examples shown in Figures 3.3 and 3.4). (a) is uniform – combining the likelihood and the prior together has left all values of r equally likely. (b) suggests that r is most likely to be low but could be high. This might be the result of starting with a uniform prior and then observing more tails than heads. Finally (c) suggests the coin is biased to land heads more often. As it is a density, the posterior tells us not just which values are likely but also provides an indication of the level of uncertainty we still have in r having observed some data.

As already mentioned, we can use the posterior density to compute expectations. For example, we could compute

$$\mathbf{E}_{p(r|y_N)} \{P(Y_{10} \leq 6)\} = \int_{r=0}^{r=1} P(Y_{10} \leq 6|r)p(r|y_N) dr,$$

the expected value of the probability that we will win. This takes into account the data we have observed, our prior beliefs and the uncertainty that remains. It will be useful in helping to decide whether or not to play the game. We will return to this later but first we will look at the kind of posterior densities we obtain in our coin example.

Comment 3.1 – Conjugate priors: A likelihood-prior pair is said to be conjugate if it results in a posterior which is of the same form as the prior. This enables us to compute the posterior density analytically without having to worry about computing the denominator in Bayes' rule, the marginal likelihood. Some common conjugate pairs are listed in the table to the right.

Prior	Likelihood
Gaussian	Gaussian
Beta	Binomial
Gamma	Gaussian
Dirichlet	Multinomial

3.2 The exact posterior

The beta distribution is a common choice of prior when the likelihood is a binomial distribution. This is because we can use some algebra to compute the posterior density exactly. In fact, the beta distribution is known as the **conjugate** prior to the binomial likelihood (see Comment 3.1). If the prior and likelihood are conjugate, the posterior will be of the same form as the prior. Specifically, $p(r|y_N)$ will give a beta distribution with parameters δ and γ whose values will be computed from the prior and y_N . The beta and binomial are not the only conjugate pair of distributions and we will see an example of another conjugate prior and likelihood pair when we return to the Olympics data later in this chapter.

Using a conjugate prior makes things much easier from a mathematical point of view. However, as we mentioned in both our discussion on loss functions in Chapter 1 and noise distributions in Chapter 2, it is more important to base our choices on modelling assumptions than mathematical convenience. In the next chapter we will see some techniques we can use in the common scenario that the pair is non-conjugate.

Returning to our example, we can omit $p(y_N)$ from Equation 3.3, leaving:

$$p(r|y_N) \propto P(y_N|r)p(r).$$

Replacing the terms on the right hand side with a binomial and beta distribution gives:

$$p(r|y_N) \propto \left[\binom{N}{y_N} r^{y_N} (1-r)^{N-y_N} \right] \times \left[\frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1} (1-r)^{\beta-1} \right]. \quad (3.5)$$

Because the prior and likelihood are conjugate, we know that $p(r|y_N)$ has to be a beta density. The beta density, with parameters δ and γ , has the following general form:

$$p(r) = Kr^{\delta-1}(1-r)^{\gamma-1},$$

where K is a constant. If we can arrange all of the terms including r on the right hand side of Equation 3.5 into something that looks like $r^{\delta-1}(1-r)^{\gamma-1}$, we can be sure that the constant must also be correct (it has to be $\Gamma(\delta+\gamma)/(\Gamma(\delta)\Gamma(\gamma))$) because we know that the posterior density is a beta density). In other words, we know what the normalising constant for a beta density is so we do not need to compute $p(y_N)$. Re-arranging Equation 3.5 gives us:

$$\begin{aligned} p(r|y_N) &\propto \left[\binom{N}{y_N} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \right] \times \left[r^{y_N} r^{\alpha-1} (1-r)^{N-y_N} (1-r)^{\beta-1} \right] \\ &\propto r^{y_N + \alpha - 1} (1-r)^{N - y_N + \beta - 1} \\ &\propto r^{\delta-1} (1-r)^{\gamma-1} \end{aligned}$$

where $\delta = y_N + \alpha$ and $\gamma = N - y_N + \beta$.

Therefore

$$p(r|y_N) = \frac{\Gamma(\alpha + \beta + N)}{\Gamma(\alpha + y_N)\Gamma(\beta + N - y_N)} r^{\alpha+y_N-1} (1-r)^{\beta+N-y_N} \quad (3.6)$$

(note that when adding γ and δ , the y_N terms cancel). This is the posterior density of r based on the prior $p(r)$ and the data y_N . Notice how the posterior parameters are computed by adding the number of heads (y_n) to the first prior parameter (α) and the number of tails ($N - y_N$) to the second (β). This allows us to gain some intuition about the prior parameters α and β – they can be thought of as the number of heads and tails in $\alpha + \beta$ previous tosses. For example, consider the second two scenarios discussed in the previous section. For the fair coin scenario, $\alpha = \beta = 50$. This is equivalent to tossing a coin 100 times and obtaining 50 heads and 50 tails. For the biased scenario, $\alpha = 5, \beta = 1$, corresponding to six tosses and five heads. Looking at Figure 3.4, this helps us explain the differing levels of variability suggested by the two densities: the fair coin density has much lower variability than the biased one because it is the result of many more hypothetical tosses. The more tosses, the more we should know about r .

The analogy is not perfect. For example, α and β don't have to be integers and can be less than 1 (0.3 heads doesn't make much sense). The analogy also breaks down when $\alpha = \beta = 1$. Observing one head and one tail means that values of $r = 0$ and $r = 1$ are impossible. However, density 1 in Figure 3.4 suggests that all values of r are equally likely. Despite these flaws, the analogy will be a useful one to bear in mind as we progress through our analysis (see Exercises EX 3.1, EX 3.2, EX 3.3 and EX 3.4).

3.3 The three scenarios

We will now investigate the posterior distribution $p(r|y_N)$ for the three different prior scenarios shown in Figure 3.4 – no prior knowledge, a fair coin and a biased coin.

3.3.1 No prior knowledge

In this scenario (MATLAB script: `coin_scenario1.m`), we assume that we know nothing of coin tossing or the stall holder. Our prior parameters are $\alpha = 1, \beta = 1$, shown in Figure 3.6(a).

To compare different scenarios we will use the expected value and variance of r under the prior. The expected value of a random variable from a beta distribution with parameters α and β (the density function of which we will henceforth denote as $\mathcal{B}(\alpha, \beta)$) is given as (see Exercise EX 3.5):

$$\begin{aligned} p(r) &= \mathcal{B}(\alpha, \beta) \\ \mathbf{E}_{p(r)} \{R\} &= \frac{\alpha}{\alpha + \beta}. \end{aligned}$$

For scenario 1:

$$\mathbf{E}_{p(r)} \{R\} = \frac{\alpha}{\alpha + \beta} = \frac{1}{2}.$$

The variance of a beta distributed random variable is given by (see Exercise EX

3.6):

$$\text{var}\{R\} = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}, \quad (3.7)$$

which for $\alpha = \beta = 1$ is

$$\text{var}\{R\} = \frac{1}{12}.$$

Note that in our formulation of the posterior (Equation 3.6) we are not restricted to updating our distribution in blocks of 10 – we can incorporate the results of any number of coin tosses. To illustrate the evolution of the posterior, we will look at how it changes toss by toss.

A new customer hands over £1 and the stall owner starts tossing the coin. The first toss results in a head. The posterior distribution after one toss is a beta distribution with parameters $\delta = \alpha + y_N$ and $\gamma = \beta + N - y_N$:

$$p(r|y_N) = \mathcal{B}(\delta, \gamma).$$

In this scenario, $\alpha = \beta = 1$ and as we have had $N = 1$ tosses and seen $y_N = 1$ heads,

$$\begin{aligned} \delta &= 1 + 1 = 2 \\ \gamma &= 1 + 1 - 1 = 1. \end{aligned}$$

This posterior distribution is shown as the solid line in Figure 3.6(b) (the prior is also shown as a dashed line). This single observation has had quite a large effect – the posterior is very different from the prior. In the prior, all values of r were equally likely. This has now changed – higher values are more likely than lower values with zero density at $r = 0$. This is consistent with the evidence – observing one head makes high values of r slightly more likely and low values slightly less likely. The density is still very broad as we have only observed one toss. The expected value of r under the posterior is:

$$\mathbf{E}_{p(r|y_N)} \{R\} = \frac{2}{3}$$

and we can see that observing a solitary head has increased the expected value of r from $1/2$ to $2/3$. The variance of the posterior is (using Equation 3.7):

$$\text{var}\{R\} = \frac{1}{18}$$

which is lower than the prior variance ($1/12$). So, the reduction in variance tells us that we have less uncertainty about the value of r than we did (we have learnt something) and the increase in expected value tells us that what we've learnt is that heads are slightly more likely than tails.

The stall owner tosses the second coin and it lands tails. We have now seen one head and one tail and so $N = 2$, $y_N = 1$, resulting in:

$$\begin{aligned} \delta &= 1 + 1 = 2 \\ \gamma &= 1 + 2 - 1 = 2. \end{aligned}$$

The posterior distribution is shown as the solid dark line in Figure 3.6(c). The lighter dash-dot line is the posterior we saw after one toss and the dashed line is the prior. The density has changed again to reflect the new evidence. As we have now observed a tail, the density at $r = 1$ should be zero and is ($r = 1$ would suggest that

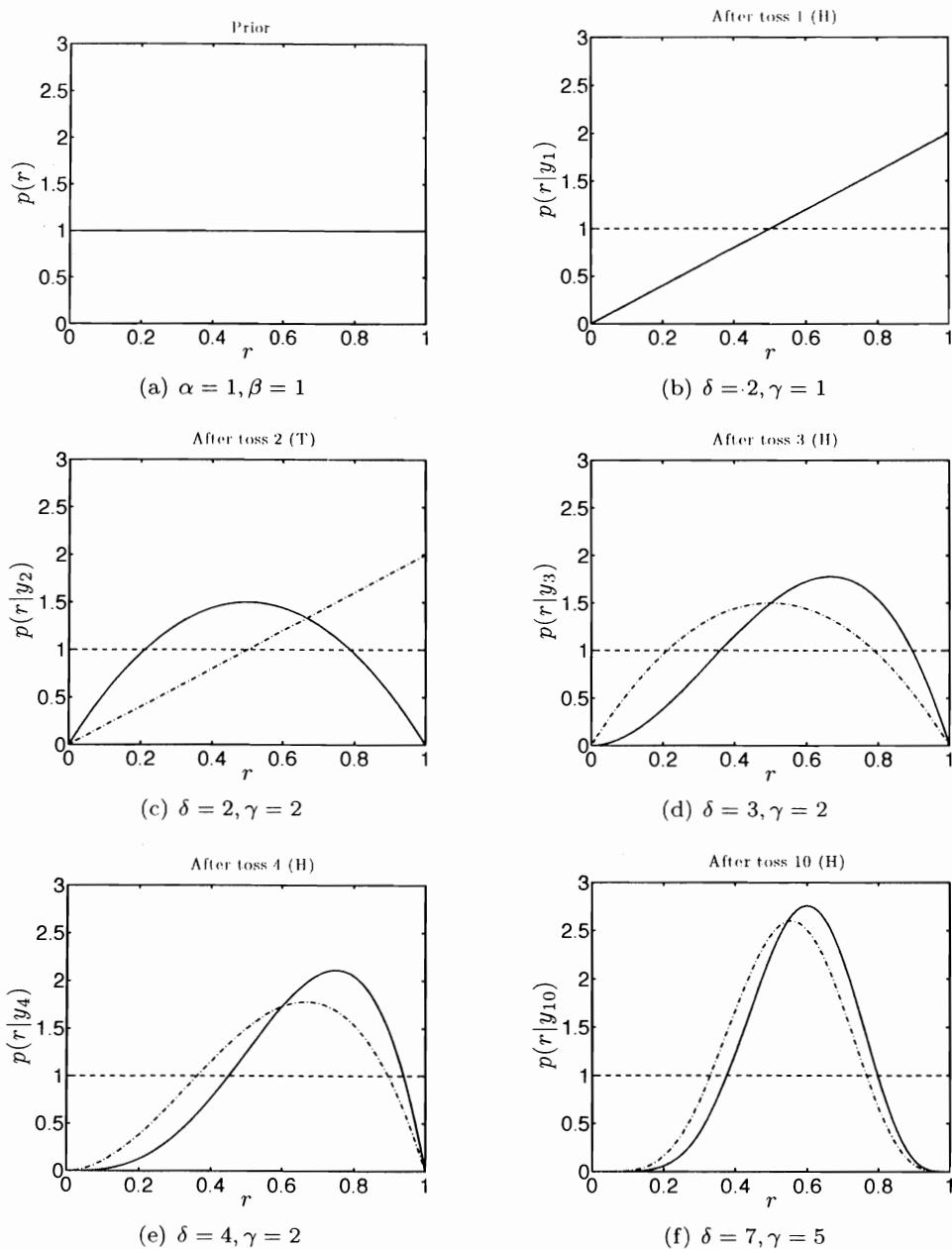


FIGURE 3.6: Evolution of $p(r|y_N)$ as the number of observed coin tosses increases.

the coin always lands heads). The density is now curved rather than straight (as we have already mentioned, the beta density function is very flexible) and observing a tail has made lower values more likely. The expected value and variance are now:

$$\mathbf{E}_{p(r|y_N)}\{R\} = \frac{1}{2}, \quad \text{var}\{R\} = \frac{1}{20}.$$

The expected value has decreased back to $1/2$. Given that the expected value under the prior was also $1/2$ you might conclude that we haven't learnt anything. However, the variance has decreased again (from $1/18$ to $1/20$) so we have less uncertainty in r and have learnt something. In fact, we've learnt that r is closer to $1/2$ than we assumed under the prior.

The third toss results in another head. We now have $N = 3$ tosses, $y_N = 2$ heads and $N - y_N = 1$ tail. Our updated posterior parameters are:

$$\begin{aligned}\delta &= \alpha + y_N = 1 + 2 = 3 \\ \gamma &= \beta + N - y_N = 1 + 3 - 2 = 2.\end{aligned}$$

This posterior is plotted in Figure 3.6(d). Once again, the posterior is the solid dark line, the previous posterior is the solid light line and the dashed line is the prior. We notice that the effect of observing this second head is to skew the density to the right, suggesting that heads are more likely than tails. Again, this is entirely consistent with the evidence – we have seen more heads than tails. We have only seen three coins, however, so there is still a high level of uncertainty – the density suggests that r could potentially still be pretty much any value between 0 and 1. The new expected value and variance are:

$$\mathbf{E}_{p(r|y_N)}\{R\} = \frac{3}{5}, \quad \text{var}\{R\} = \frac{1}{25}.$$

The variance has decreased again, reflecting the decrease in uncertainty that we would expect as we see more data.

Toss 4 also comes up heads ($y_N = 3, N = 4$), resulting in $\delta = 1 + 3 = 4$ and $\gamma = 1 + 4 - 3 = 2$. Figure 3.6(e) shows the current and previous posteriors and prior in the now familiar format. The density has once again been skewed to the right – we've now seen three heads and only one tail so it seems likely that r is greater than $1/2$. Also notice the difference between the $N = 3$ posterior and the $N = 4$ posterior for very low values of r – the extra head has left us pretty convinced that r is not 0.1 or lower. The expected value and variance are given by

$$\mathbf{E}_{p(r|y_N)}\{R\} = \frac{2}{3}, \quad \text{var}\{R\} = \frac{2}{63} = 0.0317,$$

where the expected value has increased and the variance has once again decreased. The remaining six tosses are made so that the complete sequence is

$$\text{H,T,H,H,H,H,T,T,T,H},$$

a total of six heads and four tails. The posterior distribution after $N = 10$ tosses ($y_N = 6$) has parameters $\delta = 1 + 6 = 7$ and $\gamma = 1 + 10 - 6 = 5$. This (along with the posterior for $N = 9$) is shown in Figure 3.6(f). The expected value and variance are

$$\mathbf{E}_{p(r|y_N)}\{R\} = \frac{7}{12} = 0.5833, \quad \text{var}\{R\} = 0.0187. \quad (3.8)$$

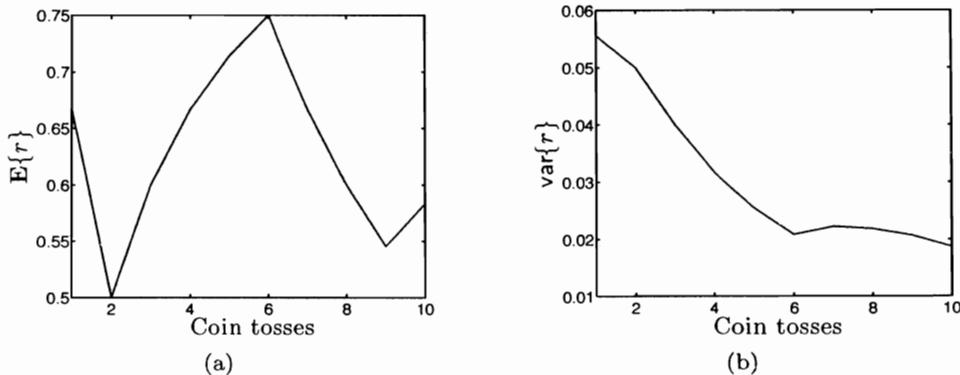


FIGURE 3.7: Evolution of expected value (a) and variance (b) of r as coin toss data is added to the posterior.

Our 10 observations have increased the expected value from 0.5 to 0.5833 and decreased our variance from $1/12 = 0.0833$ to 0.0187. However, this is not the full story. Examining Figure 3.6(f), we see that we can also be pretty sure that $r > 0.2$ and $r < 0.9$. The uncertainty in the value of r is still quite high because we have only observed 10 tosses.

Figure 3.7 summarises how the expected value and variance change as the 10 observations are included. The expected value jumps around a bit, whereas the variance steadily decreases as more information becomes available. At the seventh toss, the variance increases. The first seven tosses are:

$$\text{H,T,H,H,H,T}.$$

The evidence up to and including toss 6 is that heads is much more likely than tails (5 out of 6). Tails on the seventh toss is therefore slightly unexpected. Figure 3.8 shows the posterior before and after the seventh toss. The arrival of the tail has forced the density to increase the likelihood of low values of r and in doing so has increased the uncertainty.

The posterior density encapsulates all of the information we have about r . Shortly, we will use this to compute the expected probability of winning the game. Before we do so, we will revisit the idea of using point estimates by extracting a single value \hat{r} of r from this density. We will then be able to compare the expected probability of winning with the probability of winning computed from a single value of r . A sensible choice would be to use $\mathbf{E}_{p(r|y_N)}\{R\}$. With this value, we can compute the probability of winning – $P(Y_{\text{new}} \leq 6|\hat{r})$. This quantity could be used to decide whether or not to play. Note that to make the distinction between observed tosses and future tosses, we will use Y_{new} as a random variable that describes 10 future tosses.

After 10 tosses, the posterior density is beta with parameters $\delta = 7, \gamma = 5$. \hat{r} is therefore:

$$\hat{r} = \frac{\delta}{\delta + \gamma} = \frac{7}{12}.$$

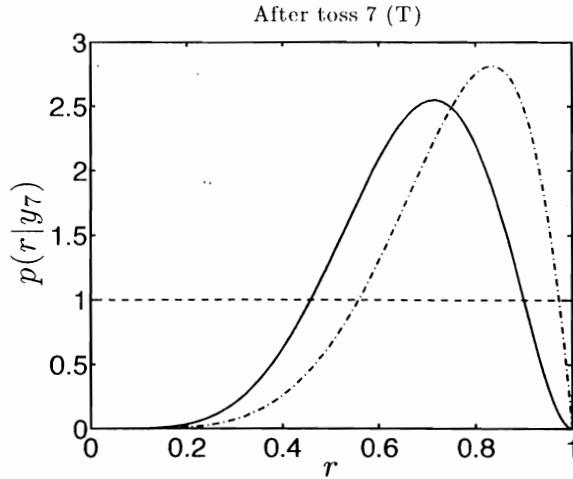


FIGURE 3.8: The posterior after six (light) and seven (dark) tosses.

The probability of winning the game follows as:

$$\begin{aligned}
 P(Y_{\text{new}} \leq 6 | \hat{r}) &= 1 - \sum_{y_{\text{new}}=7}^{10} P(Y_{\text{new}} = y_{\text{new}} | \hat{r}) \\
 &= 1 - 0.3414 \\
 &= 0.6586,
 \end{aligned}$$

suggesting that we will win more often than lose.

Using all of the posterior information requires computing

$$\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\}.$$

Re-arranging and manipulating the expectation provides us with the following expression:

$$\begin{aligned}
 \mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} &= \mathbf{E}_{p(r|y_N)} \{1 - P(Y_{\text{new}} \geq 7|r)\} \quad (3.9) \\
 &= 1 - \mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \geq 7|r)\} \\
 &= 1 - \mathbf{E}_{p(r|y_N)} \left\{ \sum_{y_{\text{new}}=7}^{y_{\text{new}}=10} P(Y_{\text{new}} = y_{\text{new}} | r) \right\} \\
 &= 1 - \sum_{y_{\text{new}}=7}^{y_{\text{new}}=10} \mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} = y_{\text{new}} | r)\}.
 \end{aligned}$$

To evaluate this, we need to be able to compute $\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} = y_{\text{new}} | r)\}$. From

the definition of expectations, this is given by:

$$\begin{aligned}\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} = y_{\text{new}}|r)\} &= \int_{r=0}^{r=1} P(Y_{\text{new}} = y_{\text{new}}|r)p(r|y_N) dr \\ &= \int_{r=0}^{r=1} \left[\binom{N_{\text{new}}}{y_{\text{new}}} r^{y_{\text{new}}} (1-r)^{N_{\text{new}}-y_{\text{new}}} \right] \left[\frac{\Gamma(\delta + \gamma)}{\Gamma(\delta)\Gamma(\gamma)} r^{\delta-1} (1-r)^{\gamma-1} \right] dr \\ &= \binom{N_{\text{new}}}{y_{\text{new}}} \frac{\Gamma(\delta + \gamma)}{\Gamma(\delta)\Gamma(\gamma)} \int_{r=0}^{r=1} r^{y_{\text{new}}+\delta-1} (1-r)^{N_{\text{new}}-y_{\text{new}}+\gamma-1} dr.\end{aligned}\quad (3.10)$$

This integral looks a bit daunting. However, on closer inspection the argument inside the integral is an unnormalised beta density with parameters $\delta + y_{\text{new}}$ and $\gamma + N_{\text{new}} - y_{\text{new}}$. In general, for a beta density with parameters α and β the following *must* be true:

$$\int_{r=0}^{r=1} \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1} (1-r)^{\beta-1} dr = 1,$$

and therefore:

$$\int_{r=0}^{r=1} r^{\alpha-1} (1-r)^{\beta-1} dr = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}.$$

Our desired expectation becomes:

$$\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} = y_{\text{new}}|r)\} = \binom{N_{\text{new}}}{y_{\text{new}}} \frac{\Gamma(\delta + \gamma)}{\Gamma(\delta)\Gamma(\gamma)} \frac{\Gamma(\delta + y_{\text{new}})\Gamma(\gamma + N_{\text{new}} - y_{\text{new}})}{\Gamma(\delta + \gamma + N_{\text{new}})}$$

which we can easily compute for a particular posterior (i.e. values of γ and δ) and values of N_{new} and y_{new} .

After 10 tosses, we have $\delta = 7$, $\gamma = 5$. Plugging these values in, we can compute the expected probability of success:

$$\begin{aligned}\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} &= 1 - \sum_{y_{\text{new}}=7}^{y_{\text{new}}=10} \mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} = y_{\text{new}}|r)\} \\ &= 1 - 0.3945 \\ &= 0.6055.\end{aligned}$$

Comparing this with the value obtained using the point estimate we can see that both predict we will win more often than not. This is in agreement with the evidence – the one person we have fully observed got six heads and four tails and hence won £2. The point estimate gives a higher probability – ignoring the posterior uncertainty makes it more likely that we will win.

Another customer plays the game. The sequence of tosses is:

H,H,T,T,H,H,H,H,H,H,

eight heads and two tails – the stall owner has won. Combining all 20 tosses that we have observed, we have $N = 20$, $y_N = 6 + 8 = 14$ heads and $N - y_N = 20 - 14 = 6$ tails. This gives $\delta = 15$ and $\gamma = 7$. The posterior density is shown in Figure 3.9, where the light line shows the posterior we had after 10 tosses and the dashed line the prior. The expected value and variance are:

$$\mathbf{E}_{p(r|y_N)} \{R\} = 0.6818, \text{var}\{R\} = 0.0094.$$

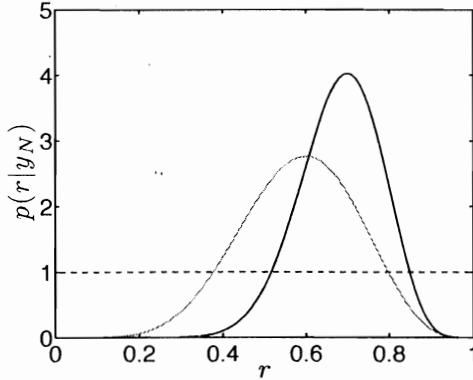


FIGURE 3.9: Posterior distribution after observing 10 tosses (light curve) and 20 tosses (dark curve). The dashed line corresponds to the prior density.

The expected value has increased and the variance has decreased (c.f. Equation 3.8). Both behaviours are what we would expect – eight heads and two tails should increase the expected value of r and the increased data should decrease the variance.

We can now recompute $\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\}$ in light of the new evidence. Plugging in the appropriate values, this is:

$$\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} = 0.4045.$$

The new evidence has pushed the density to the right, made high values of r (and hence the coin landing heads) more likely and reduced the probability of winning. For completeness, we can also compute $P(Y_{\text{new}} \leq 6|\hat{r}) = 0.3994$.

This corresponds to an expected return of:

$$2 \times 0.4045 - 1 = -0.1910,$$

equivalent to a loss of about 20p per go.

In this example we have now touched upon all of the important components of Bayesian Machine Learning – choosing priors, choosing likelihoods, computing posteriors and using expectations to make predictions. We will now repeat this process for the other two prior scenarios.

3.3.2 The fair coin scenario

For the fair coin scenario (MATLAB script: `coin_scenario2.m`), we assumed that $\alpha = \beta = 50$, which is analogous to assuming that we have already witnessed 100 tosses, half of which resulted in heads. The first thing to notice here is that 100 tosses corresponds to much more data than we are going to observe here (20 tosses). Should we expect our data to have the same effect as it did in the previous scenario?

Figure 3.10(a) shows the prior density and Figures 3.10(b), 3.10(c), 3.10(d), 3.10(e) and 3.10(f) show the posterior after 1, 5, 10, 15 and 20 tosses, respectively. For this scenario, we have not shown the previous posterior at each stage – it is too close to

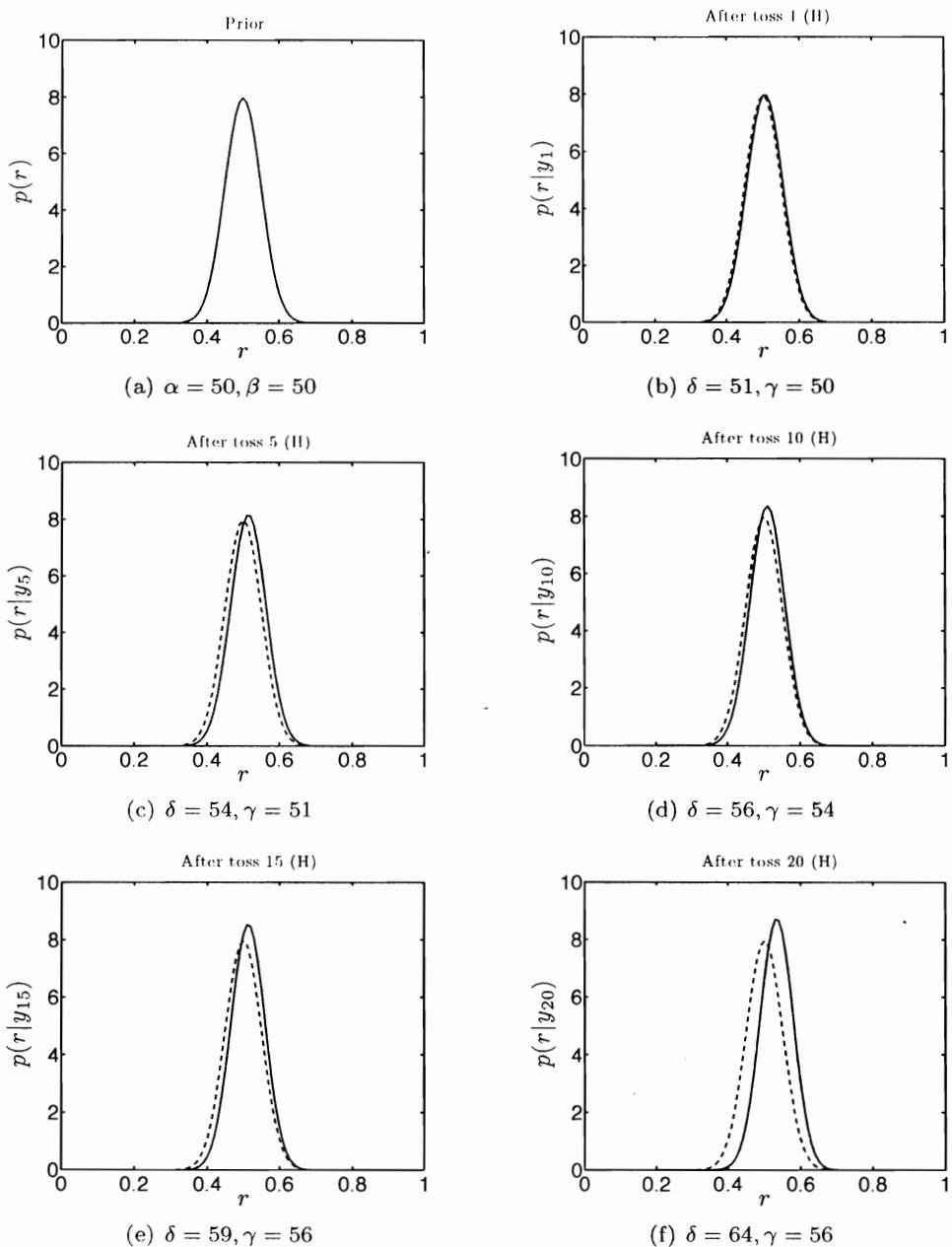


FIGURE 3.10: Evolution of the posterior $p(r|y_N)$ as more coin tosses are observed for the fair coin scenario. The dashed line shows the prior density.

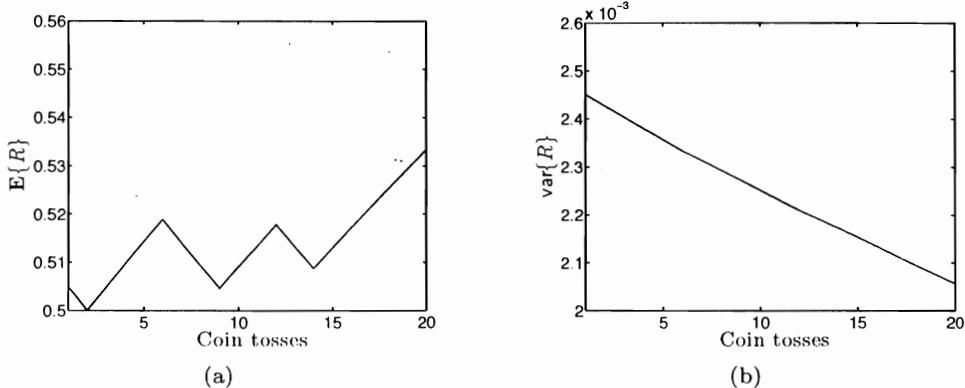


FIGURE 3.11: Evolution of $E_{p(r|y_N)}\{R\}$ (a) and $\text{var}\{R\}$ (b) as the 20 coin tosses are observed for the fair coin scenario.

the current one. However, in most cases, the change in posterior is so small that the lines almost lie right on top of one another. In fact, it is only after about 10 tosses that the posterior has moved significantly from the prior. Recalling our analogy for the beta prior, this prior includes the evidential equivalent of 100 tosses and so it is not surprising that adding another 10 makes much difference.

The evolution of $E_{p(r|y_N)}\{R\}$ and $\text{var}\{R\}$ as the 20 tosses are observed can be seen in Figure 3.11. We see very little change in either as the data appear compared to the changes we observed in Figure 3.6. Such small changes are indicative of a very *strong* prior density. The prior will dominate over the data until we've observed many more tosses – i.e. $p(r)$ dominates $p(y_N|r)$ in Equation 3.3. We have created a model that is stuck in its ways and will require a lot of persuasion to believe otherwise.

Just as in the previous section, we can work out $E_{p(r|y_N)}\{P(Y_{\text{new}} \leq 6|r)\}$. After all 20 tosses have been observed, we have $\delta = \alpha + y_N = 50 + 14 = 64$ and $\gamma = \beta + N - y_N = 50 + 20 - 14 = 56$. The expectation works out as:

$$E_{p(r|y_N)}\{P(Y_{\text{new}} \leq 6|r)\} = 0.7579. \quad (3.11)$$

As before, we can also see how much difference there is between this value and the value obtained using the point estimate \hat{r} , $P(Y_{\text{new}} \leq 6|\hat{r})$ (in this case, $\hat{r} = 64/(64 + 56) = 0.5333$):

$$P(Y_{\text{new}} \leq 6|\hat{r}) = 0.7680.$$

Both quantities predict that we will win more often than not. In light of what we've seen about the posterior, this should come as no surprise. The data has done little to overcome the prior assumption that the coin is fair and we already know that if the coin is fair, we will tend to win (a fair coin will result in us winning, on average, 66p per game – see the start of Section 3.1).

As an aside, consider how accurate our approximation $P(Y_{\text{new}} \leq 6|\hat{r})$ is to the proper expectation in this scenario and the previous one. In the previous one, the difference between the two values was

$$|E_{p(r|y_N)}\{P(Y_{\text{new}} \leq 6|r)\} - P(Y_{\text{new}} \leq 6|\hat{r})| = 0.0531.$$

In this example, the values are closer:

$$|\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} - P(Y_{\text{new}} \leq 6|\hat{r})| = 0.0101.$$

There is a good reason why this is the case – as the variance in the posterior decreases (the variance in scenario 2 is much lower than in scenario 1), the probability density becomes more and more condensed around one particular point. Imagine the variance decreasing to such an extent that there was a single value of r that had probability 1 of occurring with $p(r|y_N)$ being zero everywhere else. The expectation we are calculating is:

$$\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} = \int_{r=0}^{r=1} P(Y_{\text{new}} \leq 6|r)p(r|y_N) dr.$$

If $p(r|y_N)$ is zero everywhere except at one specific value, say \hat{r} , this becomes

$$\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} = P(Y_{\text{new}} \leq 6|\hat{r}).$$

In other words, as the variance decreases, $P(Y_{\text{new}} \leq 6|\hat{r})$ becomes a better and better approximation to the true expectation. This is not specific to this example – as the quantity of data increases (and uncertainty about parameters subsequently decreases), point approximations become more reliable.

3.3.3 A biased coin

In the final scenario we assume that the coin (and therefore the stall owner) is biased to generate more heads than tails (MATLAB script: `coin_scenario3.m`). This is encoded through a beta prior with parameters $\alpha = 5$, $\beta = 1$. The expected value is:

$$\mathbf{E}_{p(r)} \{r\} = 5/6,$$

five coins out of every six will come up heads. Just as for scenario 2, Figure 3.12(a) shows the prior density and Figures 3.12(b), 3.12(c), 3.12(d), 3.12(e) and 3.12(f) show the posterior after 1, 5, 10, 15 and 20 tosses, respectively. Given what we've already seen, there is nothing unusual here. The posterior moves quite rapidly away from the prior (the prior effectively has only the influence of $\alpha + \beta = 6$ data points). Figure 3.13 shows the evolution of expected value and variance. The variance curve has several bumps corresponding to tosses resulting in tails. This is because of the strong prior bias towards a high r value. We don't expect to see many tails under this assumption and so when we do, the model becomes less certain. Once again, we calculate the true quantity of interest, $\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\}$. The final posterior parameter values are $\delta = \alpha + y_N = 5 + 14 = 19$, $\gamma = 1 + N - y_N = 1 + 20 - 14 = 7$. Plugging these in,

$$\mathbf{E}_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} = 0.2915.$$

The approximation, noting that $\hat{r} = 19/(19 + 7) = 0.7308$ is

$$P(Y_{\text{new}} \leq 6|\hat{r}) = 0.2707.$$

Both values suggest we will lose money on average.

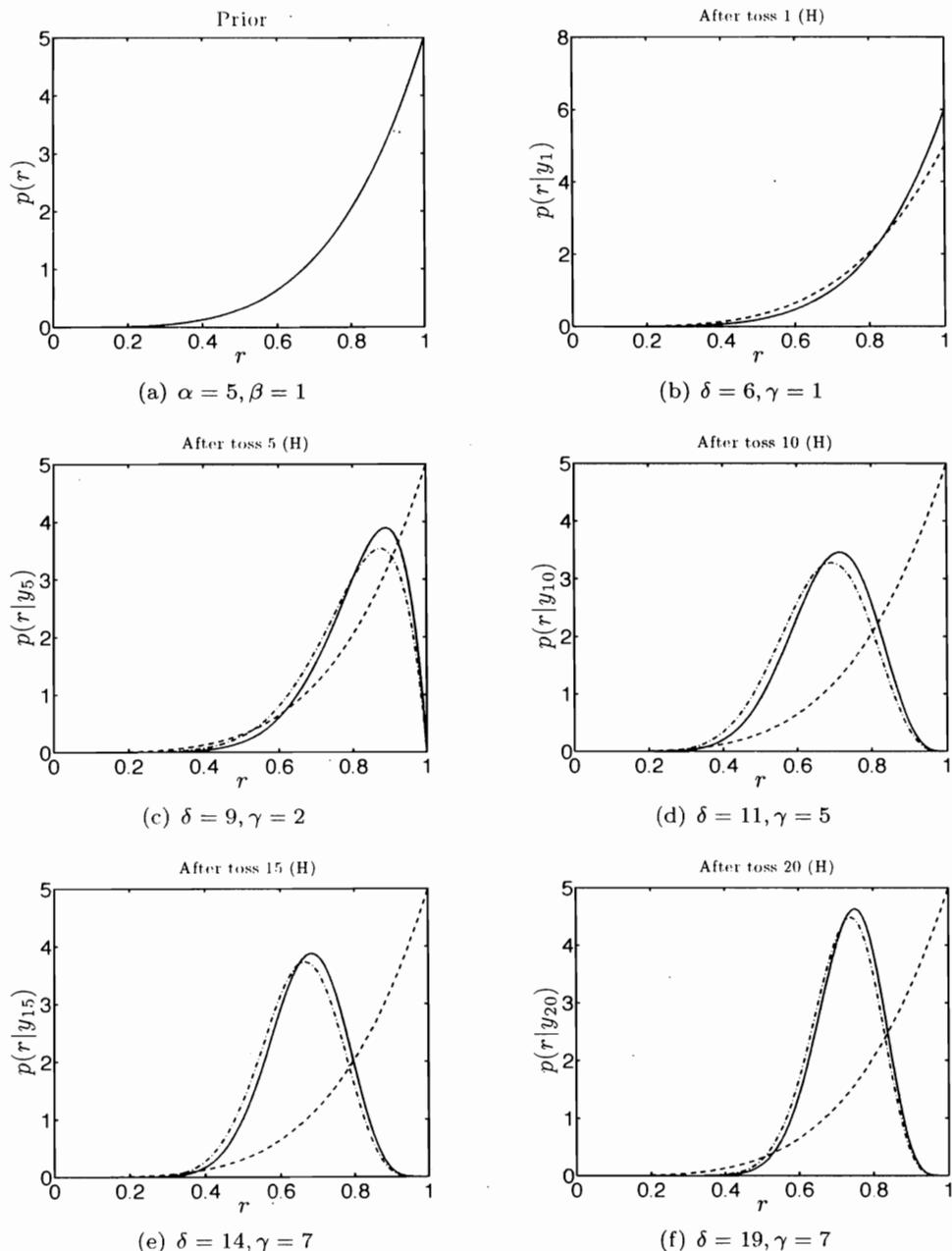


FIGURE 3.12: Evolution of the posterior $p(r|y_N)$ as more coin tosses are observed for the biased coin scenario. The dashed line shows the prior density and in the last four plots, the dash-dot line shows the previous posterior (i.e. the posterior after 4, 9, 14 and 19 tosses).

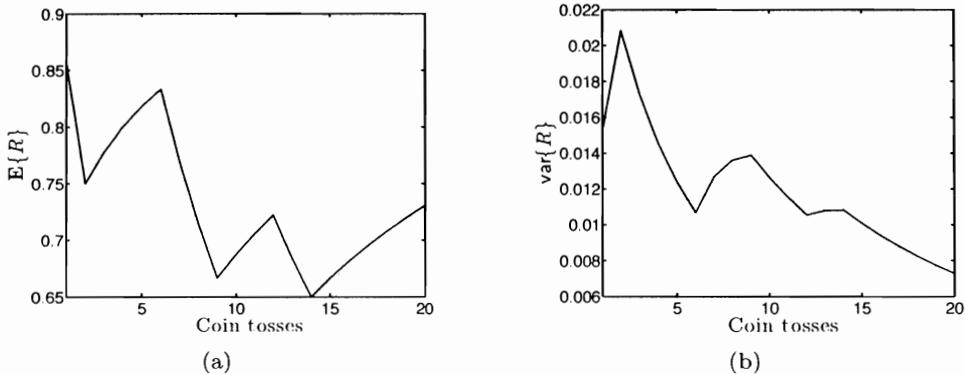


FIGURE 3.13: Evolution of $E_{p(r|y_N)} \{R\}$ (a) and $\text{var}\{R\}$ (b) as the 20 coin tosses are observed for the biased coin scenario.

3.3.4 The three scenarios – a summary

Our three different scenarios have given us different values for the expected probability of winning:

1. No prior knowledge: $E_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} = 0.4045$
2. Fair coin: $E_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} = 0.7579$
3. Biased coin: $E_{p(r|y_N)} \{P(Y_{\text{new}} \leq 6|r)\} = 0.2915$.

Which one should we choose? We could choose based on which of the prior beliefs seems most plausible. Given that the stall holder doesn't look like he is about to go out of business, scenario 3 might be sensible. We might decide that we really do not know anything about the stall holder and coin and look to scenario 1. We might believe that an upstanding stall holder would never stoop to cheating and go for scenario 2. It is possible to justify any of them. What we have seen is that the Bayesian technique allows you to combine the data observed (20 coin tosses) with some prior knowledge (one of the scenarios) in a principled way. The posterior density explicitly models the uncertainty that remains in r at each stage and can be used to make predictions (see Exercise EX 3.7 and EX 3.8).

3.3.5 Adding more data

Before we move on, it is worth examining the effect of adding more and more data. We have seen in each of our scenarios that the addition of more data results in the posterior diverging from the prior – usually through a decrease in variance. In fact, if we continue adding more data, we will find that the posteriors for all three scenarios start to look very similar. In Figure 3.14 we see the posteriors for the three scenarios after 100 and 1000 tosses. Compared with the posteriors for the three scenarios after small numbers of tosses have been observed (Figures 3.6(f), 3.10(d) and 3.12(d)) we notice that the posteriors are becoming more and more similar. This is particularly noticeable for scenarios 1 and 3 – by 1000 tosses they are indistinguishable. The difference between these two and the posteriors for scenario 2 is due to the

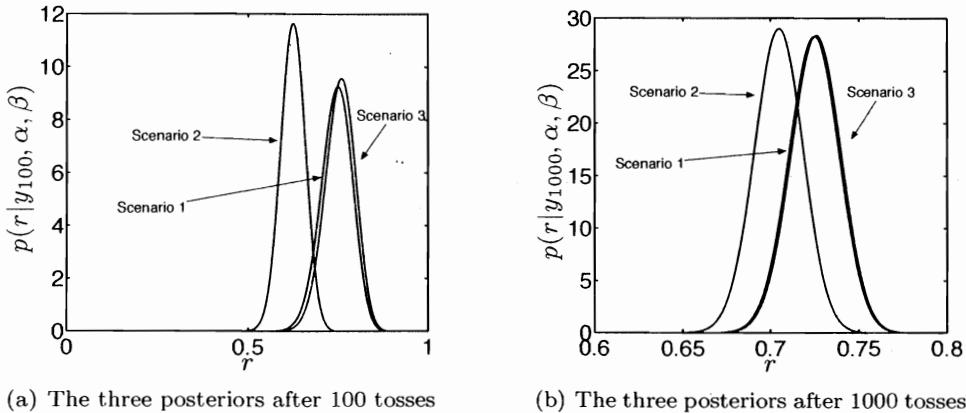


FIGURE 3.14: The posterior densities for the three scenarios after 100 coin tosses (left) and 1000 coin tosses (right).

high strength (low variance) of the prior for scenario 2 – the prior corresponds to a very strong belief and it will take a lot of contradictory data to remove that influence.

The diminishing effect of the prior as the quantity of data increases is easily explained if we look at the expression used to compute the posterior. Ignoring the normalising marginal likelihood term, the posterior is proportional to the likelihood multiplied by the prior. As we add more data, the prior is unchanged but the likelihood becomes a product (if the normal independence assumptions are made) of individual likelihood for more and more observations. This increase will gradually swamp the single contribution from the prior. It is also very intuitive – as we observe more and more data, beliefs we had before seeing any become less and less important.

3.4 Marginal likelihoods

Fortunately, subjective beliefs are not the only option for determining which of our three scenarios is best. Earlier in this chapter, when discussing the terms in Equation 3.3, we showed how the denominator $p(y_N)$ could be considered to be related to r as follows:

$$\begin{aligned} p(y_N) &= \int_{r=0}^{r=1} p(r, y_N) dr \\ &= \int_{r=0}^{r=1} p(y_N|r)p(r) dr. \end{aligned} \quad (3.12)$$

Now when considering different choices of $p(r)$, we need to be more strict about our conditioning. $p(r)$ should actually be written as $p(r|\alpha, \beta)$ as the density is conditioned on a particular pair of α and β values. Extending this conditioning through

Equation 3.12 gives

$$p(y_N|\alpha, \beta) = \int_{r=0}^{r=1} p(y_N|r)p(r|\alpha, \beta) dr. \quad (3.13)$$

The marginal likelihood (so called because r has been marginalised), $p(y_N|\alpha, \beta)$, is a very useful and important quantity. It tells us how likely the data (y_N) is given our choice of prior parameters α and β . The higher $p(y_N|\alpha, \beta)$, the better our evidence agrees with the prior specification. Hence, for our dataset, we could use $p(y_N|\alpha, \beta)$ to help choose the best scenario: select the scenario for which $p(y_N|\alpha, \beta)$ is highest.

To compute this quantity, we need to evaluate the following integral:

$$\begin{aligned} p(y_N|\alpha, \beta) &= \int_{r=0}^{r=1} p(y_N|r)p(r|\alpha, \beta) dr \\ &= \int_{r=0}^{r=1} \binom{N}{y_N} r^{y_N} (1-r)^{N-y_N} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1} (1-r)^{\beta-1} dr \\ &= \binom{N}{y_N} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_{r=0}^{r=1} r^{\alpha+y_N-1} (1-r)^{\beta+N-y_N-1} dr. \end{aligned}$$

This is of exactly the same form as Equation 3.10. The argument inside the integral is an unnormalised beta density and so we know that by integrating it we will get the inverse of the normal beta normalising constant. Therefore,

$$p(y_N|\alpha, \beta) = \binom{N}{y_N} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(\alpha+y_N)\Gamma(\beta+N-y_N)}{\Gamma(\alpha+\beta+N)}. \quad (3.14)$$

In our example, $N = 20$ and $y_N = 14$ (there were a total of 14 heads in the 2 sets of 10 tosses). We have three different possible pairs of α and β values. Plugging these values into Equation 3.14 gives

1. No prior knowledge, $\alpha = \beta = 1$, $p(y_N|\alpha, \beta) = 0.0476$.
2. Fair coin, $\alpha = \beta = 50$, $p(y_N|\alpha, \beta) = 0.0441$.
3. Biased coin, $\alpha = 5, \beta = 1$, $p(y_N|\alpha, \beta) = 0.0576$.

The prior corresponding to the biased coin has the highest marginal likelihood and the fair coin prior has the lowest. In the previous section we saw that the probability of winning under that scenario was $\mathbf{E}_{p(r|y_N, \alpha, \beta)} \{P(Y_{\text{new}} \leq 6|r)\} = 0.2915$ (note that we're now conditioning the posterior on the prior parameters – $p(r|y_N, \alpha, \beta)$).

A word of caution is required here. Choosing priors in this way is essentially choosing the prior that best agrees with the data. The prior no longer corresponds to our beliefs *before* we observe any data. In some applications this may be unacceptable. What it does give us is a single value that tells us how much the data backs up the prior beliefs. In the above example, the data suggests that the biased coin prior is best supported by the evidence.

3.4.1 Model comparison with the marginal likelihood

It is possible to extend the prior comparison in the previous section to using the marginal likelihood to optimise α and β . Assuming that α and β can take any value in the following ranges:

$$\begin{aligned} 0 &\leq \alpha \leq 50 \\ 0 &\leq \beta \leq 30, \end{aligned}$$

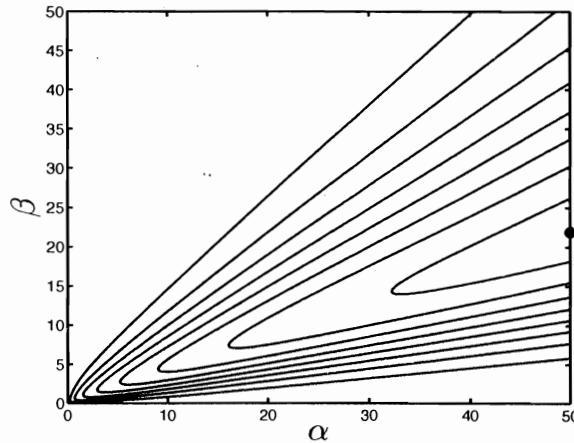


FIGURE 3.15: Marginal likelihood contours (as a function of the prior parameters, α and β) for the coin example. Circle towards the top right shows the optimum.

we can search for the values of α and β that maximise $p(y_N|\alpha, \beta)$.

Figure 3.15 shows the marginal likelihood as α and β are varied in their respective ranges. The optimum value is $\alpha = 50, \beta = 22$, resulting in a marginal likelihood of 0.1694. Choosing parameters in this way is known as Type II Maximum Likelihood (to distinguish it from standard (i.e. Type I) Maximum Likelihood introduced in Chapter 2).

3.5 Hyper-parameters

The Bayesian analysis presented thus far has all been based on the idea that we can represent any quantities of interest as random variables (e.g. r , the probability of a coin landing heads). r is not the only parameter of interest in our example. α and β are also parameters – could we do the same thing with them? In some cases we can be directed towards particular values based on our knowledge of the problem (we might know that the coin is biased). Often we will not know the exact value that they should take and should therefore treat them as random variables. To do so, we need to define a prior density over all random variables – $p(r, \alpha, \beta)$. This factorises as (see Section 2.2.5):

$$p(r, \alpha, \beta) = p(r|\alpha, \beta)p(\alpha, \beta).$$

In addition, it will often be useful to assume that α and β are independent: $p(\alpha, \beta) = p(\alpha)p(\beta)$. The quantity in which we are interested is the posterior over all parameters in the model:

$$p(r, \alpha, \beta|y_N).$$

Applying Bayes' rule, we have

$$\begin{aligned} p(r, \alpha, \beta | y_N) &= \frac{p(y_N | r, \alpha, \beta)p(r, \alpha, \beta)}{p(y_N)} \\ &= \frac{p(y_N | r)p(r, \alpha, \beta)}{p(y_N)} \\ &= \frac{p(y_N | r)p(r | \alpha, \beta)p(\alpha, \beta)}{p(y_N)}. \end{aligned}$$

Note that in the second step, we removed α and β from the likelihood $p(y_N | r)$. This is another example of conditional independence (see Section 2.7.1). The distribution over y_N depends on α and β but only through their influence on r . Conditioned on a particular value of r , this dependence is broken.

$p(\alpha, \beta)$ will normally require some additional parameters – i.e. $p(\alpha, \beta | \kappa)$, where κ controls the density in the same way that α and β control the density for r . κ is known as a **hyper-parameter** because it is a parameter controlling the prior on the parameters controlling the prior on r . When computing the marginal likelihood, we integrate over all random variables and are just left with the data conditioned on the hyper-parameters:

$$p(y_N | \kappa) = \iiint p(y_N | r)p(r | \alpha, \beta)p(\alpha, \beta | \kappa) dr d\alpha d\beta.$$

Unfortunately, adding this extra complexity to the model often means that computation of the quantities of interest – the posterior $p(r, \alpha, \beta | y_N, \kappa)$ (and any predictive expectations) and the marginal likelihood $p(y_N | \kappa)$ – is analytically intractable and requires one of the approximation methods that we will introduce in Chapter 4.

At this point, one could imagine indefinitely adding layers to the model. For example, κ could be thought of as a random variable that comes from a density parameterised by other random variables. The number of levels in the hierarchy (how far we go before we fix one or more parameters) will be dictated by the data we are trying to model (perhaps we can specify exact values at some level) or how much computation we can tolerate. In general, the more layers we add the more complex it will be to compute posteriors and predictions.

3.6 Graphical models

When adding extra layers to our model (hyper-parameters, etc.), they can quickly become unwieldy. It is popular to describe them graphically. A **graphical model** is a network where nodes correspond to random variables and edges to dependencies between random variables. For example, in Section 2.2.4 we introduced various properties of random variables through a model that consisted of two random variables – one representing the toss of a coin (X) and one representing how I say the coin landed (Y). The model is defined through the conditional distribution $P(Y = y | X = x)$ and is represented graphically in Figure 3.16(a). The two nodes are joined by an arrow to show that Y is defined as being conditioned on X . Note also that the node for Y is

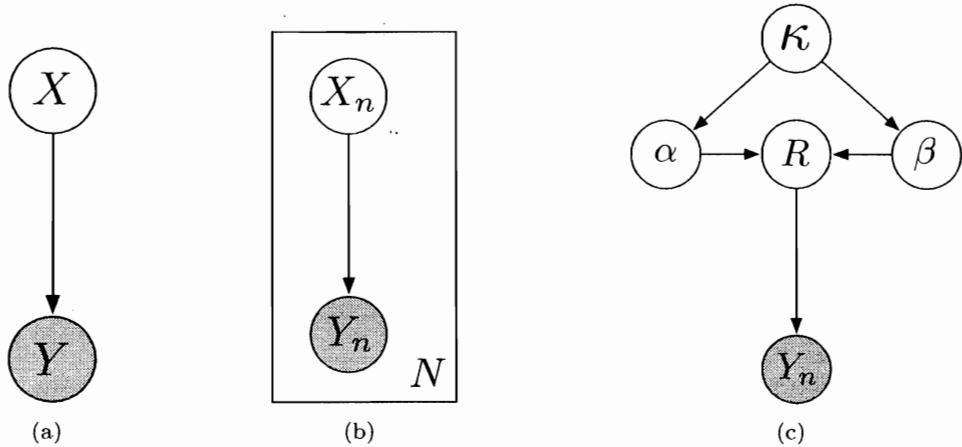


FIGURE 3.16: Graphical model examples. Nodes correspond to random variables with the shaded nodes corresponding to things that we observe. Arrows describe the dependencies between variables and the plates describe multiple instances. For example, in (b), there are N random variables Y_n ($n = 1, \dots, N$) and each is dependent on a random variable X_n . (c) is a graphical representation of the model used in the coin example with the addition of a prior on α and β parameterised by κ .

shaded. This is because, as far as the listener is concerned, this variable is *observed*. The listener does not see the coin actually landing and so doesn't observe X . Imagine that the procedure was repeated N times; we now have $2N$ random variables, X_1, \dots, X_N and Y_1, \dots, Y_N . Drawing all of these would be messy. Instead we can embed the nodes within a **plate**. Plates are rectangles that tell us that whatever is embedded within them is repeated a number of times. The number of times is given in the bottom right corner, as shown in Figure 3.16(b).

Figure 3.16(c) shows a graphical representation of our coin toss model. It has a single (observed) random variable that represents the number of heads in N tosses, y_N . This is conditioned on a random variable R which depends on random variables α and β . Finally, α and β are dependent on the hyper-parameter κ .

More information on graphical models can be found in the suggested reading at the end of the chapter.

3.6.1 Summary

In the previous sections we have introduced many new concepts. Perhaps the most important is the idea of treating all quantities of interest as random variables. To do this we must define a prior distribution over the possible values of these quantities and then use Bayes' rule (Equation 3.3) to see how the density changes as we incorporate evidence from observed data. The resulting posterior density can be examined and used to compute interesting expectations. In addition, we have shown

how the marginal likelihood (the normalisation constant in Bayes' rule) can be used to compare different models – for example, choosing the most likely prior in our coin tossing example – and discussed the possible pitfalls and objections to such an approach. Finally we have shown how the Bayesian method can be extended by treating parameters that define the priors over other parameters as random variables. Additions to the hierarchy such as this often make analytical computations intractable and we have to resort to sampling and approximation based techniques, which are the subject of the next chapter.

3.7 A Bayesian treatment of the Olympics 100 m data

We now return to the Olympics 100 m data. In the previous chapters we have fitted a linear (in the parameters) model by minimising the squared loss and then incorporated an explicit noise model and found optimal parameter values by maximising the likelihood. In this section, we will give the data a Bayesian treatment with the aim of making a prediction for the 2012 Olympics in London. This will involve several steps. First, we will need to define the prior and likelihood (as we did in the coin example) and use these to compute the posterior density over the parameters of our model, just as we computed the posterior over r in the coin example. Once we've computed the posterior, we can use it to make predictions for new Olympics years.

3.7.1 The model

We will use the k th order polynomial model that was introduced in Chapter 1 with the Gaussian noise model introduced in Chapter 2:

$$t_n = w_0 + w_1 x_n + w_2 x_n^2 + \cdots + w_K x_n^K + \epsilon_n,$$

where $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$. In vector form, this corresponds to

$$t_n = \mathbf{w}^\top \mathbf{x}_n + \epsilon_n$$

where $\mathbf{w} = [w_0, \dots, w_K]^\top$ and $\mathbf{x}_n = [1, x_n, x_n^2, \dots, x_n^K]^\top$. Stacking all of the responses into one vector $\mathbf{t} = [t_1, \dots, t_N]^\top$ and all of the inputs into a single matrix, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^\top$ (just as in Equation 1.18), we get the following expression for the whole dataset:

$$\mathbf{t} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon},$$

where $\boldsymbol{\epsilon} = [\epsilon_1, \dots, \epsilon_N]^\top$.

In this example, we are going to slightly simplify matters by assuming that we know the true value of σ^2 . We could use all of the methods introduced in this chapter to treat σ^2 as a random variable and we could get analytical results for the posterior distribution but the maths is messier which could detract from the main message.

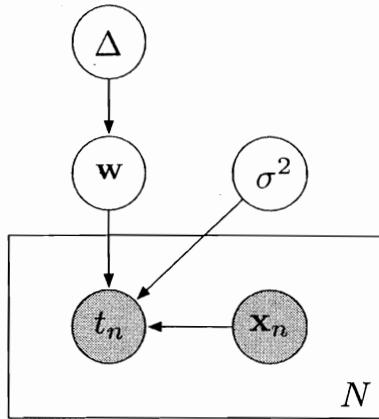


FIGURE 3.17: Graphical model for the Bayesian model of the Olympics men's 100 m data.

Substituting these various symbols into Bayes' rule gives:

$$\begin{aligned} p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \Delta) &= \frac{p(\mathbf{t}|\mathbf{w}, \mathbf{X}, \sigma^2, \Delta)p(\mathbf{w}|\Delta)}{p(\mathbf{t}|\mathbf{X}, \sigma^2, \Delta)} \\ &= \frac{p(\mathbf{t}|\mathbf{w}, \mathbf{X}, \sigma^2)p(\mathbf{w}|\Delta)}{p(\mathbf{t}|\mathbf{X}, \sigma^2, \Delta)}. \end{aligned}$$

where Δ corresponds to some set of parameters required to define the prior over \mathbf{w} that will be defined more precisely below. The graphical model can be seen in Figure 3.17. Expanding the marginal likelihood:

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \Delta) = \frac{p(\mathbf{t}|\mathbf{w}, \mathbf{X}, \sigma^2)p(\mathbf{w}|\Delta)}{\int p(\mathbf{t}|\mathbf{w}, \mathbf{X}, \sigma^2)p(\mathbf{w}|\Delta) d\mathbf{w}}. \quad (3.15)$$

We are interested in making predictions which will involve taking an expectation with respect to this posterior density. In particular, for a set of attributes \mathbf{x}_{new} corresponding to a new Olympics year, the density over the associated winning time t_{new} is given by:

$$p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2, \Delta) = \int p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \sigma^2)p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \Delta) d\mathbf{w}. \quad (3.16)$$

Notice again the conditioning on the right hand side. The posterior density of \mathbf{w} does not depend on \mathbf{w}_{new} and so it does not appear in the conditioning. Similarly, when we make predictions, we will not be using Δ and so it doesn't appear in $p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \sigma^2)$. Predictions could also take the form of probabilities. For example, we could compute the probability that the winning time will be under 9.5 seconds:

$$P(t_{\text{new}} < 9.5|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2, \Delta) = \int P(t_{\text{new}} < 9.5|\mathbf{x}_{\text{new}}, \mathbf{w}, \sigma^2)p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \Delta) d\mathbf{w}. \quad (3.17)$$

3.7.2 The likelihood

The likelihood $p(\mathbf{t}|\mathbf{w}, \mathbf{X}, \sigma^2)$ is exactly the quantity that we maximised in the previous chapter. Our model tells us that

$$\mathbf{t} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_N)$. This is a Gaussian random variable ($\boldsymbol{\epsilon}$) plus a constant. We showed in Section 2.7 that this is equivalent to the Gaussian random variable with the constant added to the mean. This gives us our likelihood:

$$p(\mathbf{t}|\mathbf{w}, \mathbf{X}, \sigma^2) = \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2 \mathbf{I}_N),$$

an N -dimensional Gaussian density with mean $\mathbf{X}\mathbf{w}$ and variance $\sigma^2 \mathbf{I}_N$. The analogous expression in the coin example is the binomial likelihood given in Equation 3.2.

3.7.3 The prior

Because we are interested in being able to produce an exact expression for our posterior, we need to choose a prior, $p(\mathbf{w}|\Delta)$, that is conjugate to the Gaussian likelihood. Conveniently, a Gaussian prior is conjugate to a Gaussian likelihood. Therefore, we will use a Gaussian prior for \mathbf{w} . In particular,

$$p(\mathbf{w}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0),$$

where we will choose the parameters $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ later. This is analogous to Equation 3.4 in the coin example. From now on we will not always explicitly condition on $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ in our expressions, i.e. for brevity, instead of writing $p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$ we will use $p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2)$ (see Exercise EX 3.10).

3.7.4 The posterior

We now turn our attention to computing the posterior. As in the coin example, we will use the fact that we *know* that the posterior will be Gaussian. This allows us to ignore the marginal likelihood in Equation 3.15 and just manipulate the likelihood and prior until we find something that is proportional to a Gaussian. As a first step, we can collect the terms in \mathbf{w} together and ignore any term that does not include \mathbf{w} :

$$\begin{aligned} p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2) &\propto p(\mathbf{t}|\mathbf{w}, \mathbf{X}, \sigma^2)p(\mathbf{w}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \\ &= \frac{1}{(2\pi)^{N/2}|\sigma^2 \mathbf{I}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{t} - \mathbf{X}\mathbf{w})^\top (\sigma^2 \mathbf{I})^{-1}(\mathbf{t} - \mathbf{X}\mathbf{w})\right) \\ &\quad \times \frac{1}{(2\pi)^{N/2}|\boldsymbol{\Sigma}_0|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0)\right) \\ &\propto \exp\left(-\frac{1}{2\sigma^2}(\mathbf{t} - \mathbf{X}\mathbf{w})^\top (\mathbf{t} - \mathbf{X}\mathbf{w})\right) \exp\left(-\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0)\right) \\ &= \exp\left\{-\frac{1}{2}\left(\frac{1}{\sigma^2}(\mathbf{t} - \mathbf{X}\mathbf{w})^\top (\mathbf{t} - \mathbf{X}\mathbf{w}) + (\mathbf{w} - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\mathbf{w} - \boldsymbol{\mu}_0)\right)\right\}. \end{aligned}$$

Multiplying the terms in the bracket out and once again removing any that don't involve \mathbf{w} gives:

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2) \propto \exp \left\{ -\frac{1}{2} \left(-\frac{2}{\sigma^2} \mathbf{t}^\top \mathbf{X} \mathbf{w} + \frac{1}{\sigma^2} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} + \mathbf{w}^\top \Sigma_0^{-1} \mathbf{w} - 2\mu_0^\top \Sigma_0^{-1} \mathbf{w} \right) \right\}. \quad (3.18)$$

We know that the posterior will be Gaussian. Therefore we can remove the constants (i.e. terms not involving \mathbf{w}) and re-arrange an expression for a multivariate Gaussian to make it look something like the expression we have above:

$$\begin{aligned} p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2) &= \mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w) \\ &\propto \exp \left(-\frac{1}{2} (\mathbf{w} - \boldsymbol{\mu}_w)^\top \boldsymbol{\Sigma}_w^{-1} (\mathbf{w} - \boldsymbol{\mu}_w) \right) \\ &\propto \exp \left\{ -\frac{1}{2} \left(\mathbf{w}^\top \boldsymbol{\Sigma}_w^{-1} \mathbf{w} - 2\boldsymbol{\mu}_w^\top \boldsymbol{\Sigma}_w^{-1} \mathbf{w} \right) \right\}. \end{aligned} \quad (3.19)$$

The linear and quadratic terms in \mathbf{w} Equation 3.18 must be equal to those in Equation 3.19. Taking the quadratic terms, we can solve for $\boldsymbol{\Sigma}_w$:

$$\begin{aligned} \mathbf{w}^\top \boldsymbol{\Sigma}_w^{-1} \mathbf{w} &= \frac{1}{\sigma^2} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X} \mathbf{w} + \mathbf{w}^\top \Sigma_0^{-1} \mathbf{w} \\ &= \mathbf{w}^\top \left(\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} + \Sigma_0^{-1} \right) \mathbf{w} \\ \boldsymbol{\Sigma}_w &= \left(\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} + \boldsymbol{\Sigma}_0^{-1} \right)^{-1} \end{aligned}$$

Similarly, equating the linear terms from Equations 3.18 and 3.19 (and using our new expression for $\boldsymbol{\Sigma}_w$) we can get an expression for $\boldsymbol{\mu}_w$:

$$\begin{aligned} -2\boldsymbol{\mu}_w^\top \boldsymbol{\Sigma}_w^{-1} \mathbf{w} &= -\frac{2}{\sigma^2} \mathbf{t}^\top \mathbf{X} \mathbf{w} - 2\boldsymbol{\mu}_0^\top \boldsymbol{\Sigma}_0^{-1} \mathbf{w} \\ \boldsymbol{\mu}_w^\top \boldsymbol{\Sigma}_w^{-1} \mathbf{w} &= \frac{1}{\sigma^2} \mathbf{t}^\top \mathbf{X} \mathbf{w} + \boldsymbol{\mu}_0^\top \boldsymbol{\Sigma}_0^{-1} \mathbf{w} \\ \boldsymbol{\mu}_w^\top \boldsymbol{\Sigma}_w^{-1} &= \frac{1}{\sigma^2} \mathbf{t}^\top \mathbf{X} + \boldsymbol{\mu}_0^\top \boldsymbol{\Sigma}_0^{-1} \\ \boldsymbol{\mu}_w^\top \boldsymbol{\Sigma}_w^{-1} \boldsymbol{\Sigma}_w &= \left(\frac{1}{\sigma^2} \mathbf{t}^\top \mathbf{X} + \boldsymbol{\mu}_0^\top \boldsymbol{\Sigma}_0^{-1} \right) \boldsymbol{\Sigma}_w \\ \boldsymbol{\mu}_w^\top &= \left(\frac{1}{\sigma^2} \mathbf{t}^\top \mathbf{X} + \boldsymbol{\mu}_0^\top \boldsymbol{\Sigma}_0^{-1} \right) \boldsymbol{\Sigma}_w \\ \boldsymbol{\mu}_w &= \boldsymbol{\Sigma}_w \left(\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{t} + \boldsymbol{\Sigma}_0^{-1} \boldsymbol{\mu}_0 \right), \end{aligned} \quad (3.20)$$

because $\boldsymbol{\Sigma}_w^\top = \boldsymbol{\Sigma}_w$ due to the fact that it must be symmetric. Therefore,

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2) = \mathcal{N}(\boldsymbol{\mu}_w, \boldsymbol{\Sigma}_w), \quad (3.21)$$

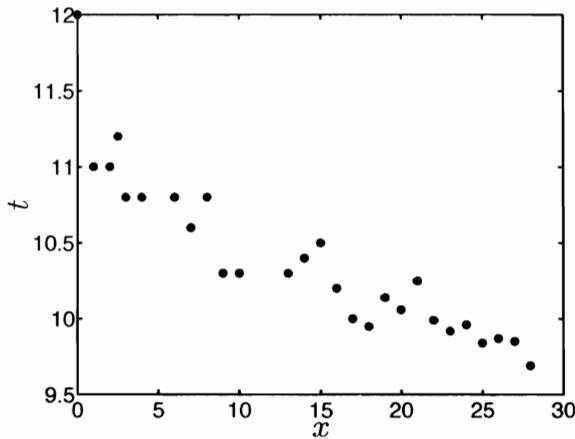


FIGURE 3.18: Olympics data with rescaled x values.

where

$$\Sigma_w = \left(\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} + \Sigma_0^{-1} \right)^{-1} \quad (3.22)$$

$$\mu_w = \Sigma_w \left(\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{t} + \Sigma_0^{-1} \mu_0 \right) \quad (3.23)$$

(see Exercise EX 3.12). These expressions do not look too far away from things we have seen before. In particular, compare Equation 3.23 with the regularised least squares solution given in Equation 1.21. In fact, if $\mu_0 = [0, 0, \dots, 0]^\top$, the expressions are almost identical. Given that the posterior is a Gaussian, the single most likely value of w is the mean of the posterior, μ_w . This is known as the **maximum a posteriori** (MAP) estimate of w and can also be thought of as the maximum value of the joint density $p(w, t | X, \sigma^2, \Delta)$ (the likelihood multiplied by the prior). We have already seen that the squared loss considered in Chapter 1 is very similar to a Gaussian likelihood and it follows from this that computing the most likely posterior value (when the likelihood is Gaussian) is equivalent to using regularised least squares (see Exercise EX 3.9). This comparison can often help to provide intuition regarding the effect of the prior.

3.7.5 A first-order polynomial

We will illustrate the prior and posterior with a 1st order polynomial as it is possible to visualise densities in the two-dimensional parameter space. The input vectors also have two elements, $\mathbf{x}_n = [1, x_n]^\top$. To aid visualisation, we will rescale the Olympics year by subtracting the year of the first Olympics (1896) from each year and then dividing each number by 4. This means that x_1 is now 0, x_2 is 1, etc. The data with this new x scaling is plotted in Figure 3.18.

Returning to the fairground, the first step in our analysis is the choice of prior parameters μ_0 and Σ_0 . For μ_0 , we will assume that we don't really know anything about what the parameters should be and choose $\mu_0 = [0, 0]^\top$. For the covariance

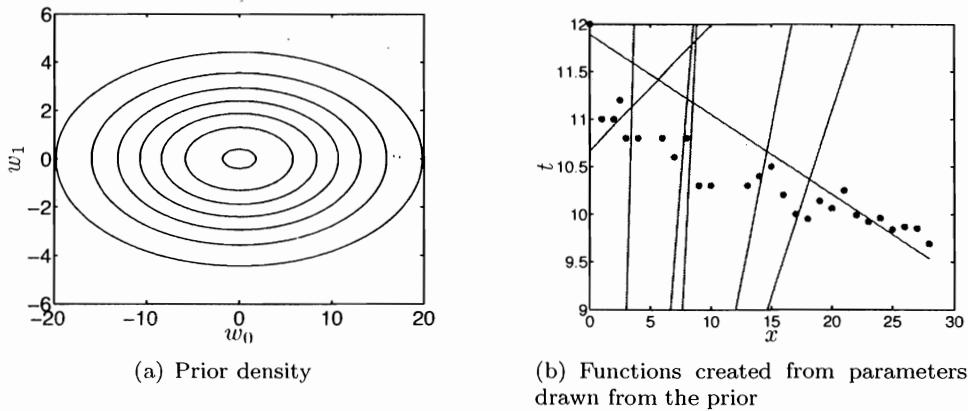


FIGURE 3.19: Gaussian prior used for the Olympics 100 m data (a) and some functions created with samples drawn from the prior (b).

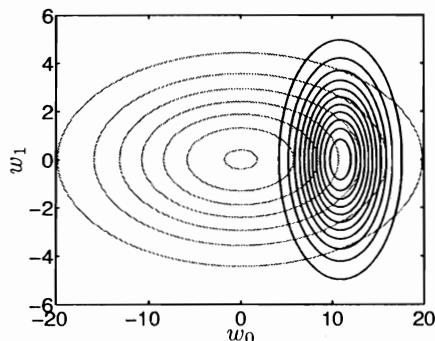
we will use the following:

$$\Sigma_0 = \begin{bmatrix} 100 & 0 \\ 0 & 5 \end{bmatrix}$$

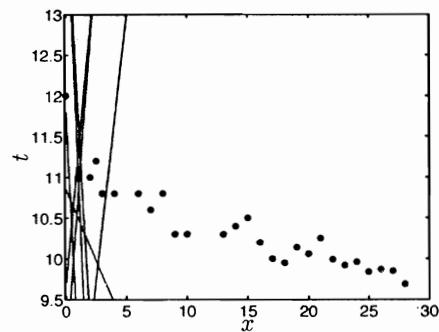
The larger value for the variance of w_0 is due to the fact that we saw in the maximum likelihood estimate that the optimal value of w_0 was much higher than that for w_1 . We have also assumed that the two variables are independent in the prior by setting the off-diagonal elements in the covariance matrix to zero. This does not preclude them from being dependent in the posterior. The contours of this prior density can be seen in Figure 3.19(a). It's hard to visualise what this means in terms of the model. To help, in Figure 3.19(b) we have shown functions corresponding to several sets of parameters drawn from this prior. To create these, we sampled \mathbf{w} from the Gaussian defined by μ_0 and Σ_0 and then substituted these into our linear model – $t_n = w_0 + w_1 x_n$. The examples show that the prior admits the possibility of many very different models.

Using $\sigma^2 = 10$ for illustrative purposes (MATLAB script: `olympbayes.m`), we can now compute the posterior distribution when we observe one data point. Using the data point corresponding to the first Olympics, our data is summarised as $\mathbf{x} = [1, 0]^\top$, $\mathbf{X} = [1, 0]$, $\mathbf{t} = [12]$. Plugging these values along with our prior parameters and $\sigma^2 = 10$ into Equations 3.21–3.23, we obtain the posterior distribution shown in Figure 3.20(a). The posterior now has much more certainty regarding w_0 but still knows very little about w_1 . This makes sense – we've provided a data point at $x = 0$ so this should be highly informative in determining the intercept but tells us very little about the gradient (one data point alone could never tell us much about the gradient). Some functions created with samples from this posterior are shown in Figure 3.20(b). They look quite different from those from the prior – in particular, they all pass quite close to our first data point.

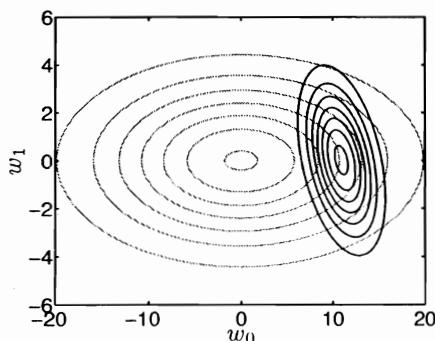
Figures 3.20(c), 3.20(d) and 3.20(e) show the evolution of the posterior after 2, 5 and 10 data points, respectively. Just as in the coin example, we notice that the posterior becomes more condensed (we are becoming more certain about the value of \mathbf{w}). Also, as it evolves, the posterior begins to tilt. This is indicative of a



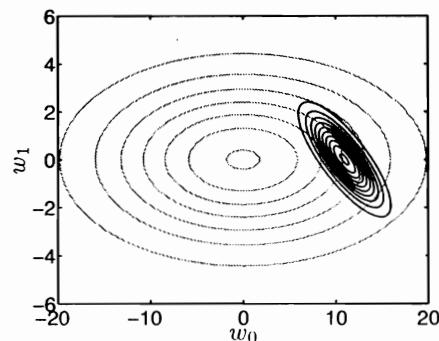
(a) Posterior density (dark contours) after the first data point has been observed. The lighter contours show the prior density



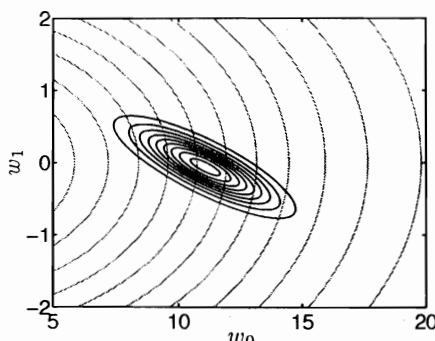
(b) Functions created from parameters drawn from the posterior after observing the first data point



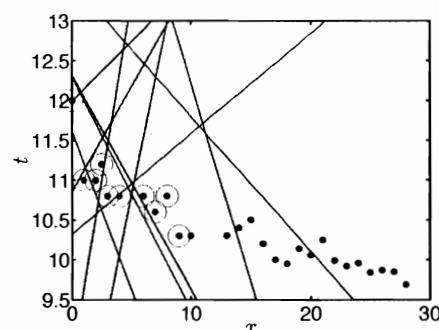
(c) Posterior density (dark contours) after the first two data points have been observed. The lighter contours show the prior density



(d) Posterior density (dark contours) after the first five data points have been observed. The lighter contours show the prior density



(e) Posterior density (dark contours) after the first 10 data points have been observed. The lighter contours show the prior density. (Note that we have zoomed in)



(f) Functions created from parameters drawn from the posterior after observing the first 10 data points (these data points are highlighted)

FIGURE 3.20: Evolution of the posterior density and example functions drawn from the posterior for the Olympics data after increasing numbers of observations have been added.

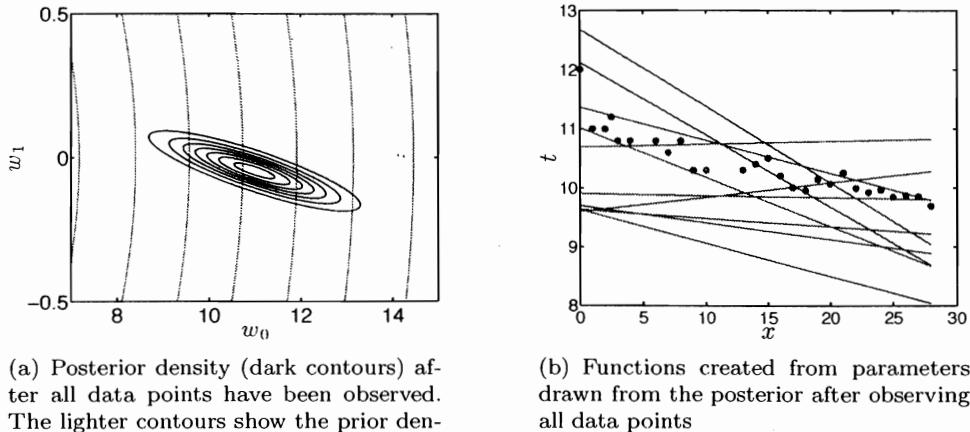


FIGURE 3.21: Posterior density (a) and sampled functions (b) for the Olympics data when all 27 data points have been added.

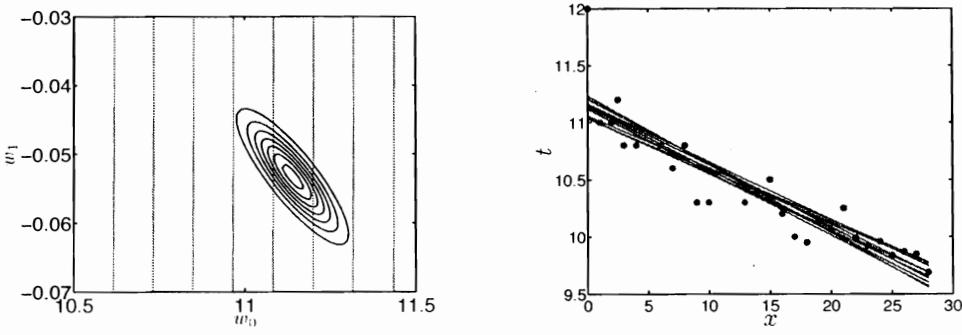
dependence developing between the two parameters – if we increase the intercept w_0 we must decrease the gradient. Recall that in the prior we assumed that the two parameters were independent (Σ_0 only had non-zero values on the diagonal) so this dependence is coming entirely from the evidence within the data. To help visualise what the posterior means at this stage, Figure 3.20(f) shows a set of functions made from parameters drawn from the posterior. When compared with Figure 3.20(b) we see that the posterior density is beginning to favour parameters that correspond to models suited to our data. Finally, in Figure 3.21(a) we see the posterior after all 27 data points have been included and in Figure 3.21(b) we see functions drawn from this posterior. The functions are really now beginning to follow the trend in our data. There is still a lot of variability however. This is due to the relatively high value of $\sigma^2 = 10$ that we chose to help visualise the prior and posteriors. For making predictions, we might want to use a more realistic value. In Figure 3.22(a) we show the posterior after all data has been observed for $\sigma^2 = 0.05$ (this is roughly the maximum likelihood value we obtained in Section 2.7.2). The posterior is now far more condensed – very little variability remains in \mathbf{w} , as can be seen by the homogeneity of the set of functions drawn in Figure 3.22(b). We will now turn our attention to making predictions.

3.7.6 Making predictions

Given a new observation \mathbf{x}_{new} , we are interested in the density:

$$p(t_{\text{new}} | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2).$$

Notice that this is not conditioned on \mathbf{w} – just as in the coin example, we are going to integrate out \mathbf{w} by taking an expectation with respect to the posterior,



(a) Posterior density (dark contours) after all have been observed. The lighter contours show the prior density. (Note that we have zoomed in)

(b) Functions created from parameters drawn from the posterior after observing all data points

FIGURE 3.22: Posterior density (a) and sampled functions (b) for the Olympics data when all 27 data points have been added with more realistic noise variance, $\sigma^2 = 0.05$.

$p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2)$. In particular, we need to compute:

$$\begin{aligned} p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) &= \mathbb{E}_{p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2)} \{p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \sigma^2)\} \\ &= \int p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \sigma^2)p(\mathbf{w}|\mathbf{t}, \mathbf{X}, \sigma^2) d\mathbf{w}. \end{aligned}$$

This is analogous to Equation 3.9 in the coin example.

$p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \sigma^2)$ is defined by our model as the product of \mathbf{x}_{new} and \mathbf{w} with some additive Gaussian noise:

$$p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{w}, \sigma^2) = \mathcal{N}(\mathbf{x}_{\text{new}}^\top \mathbf{w}, \sigma^2).$$

Because this expression and the posterior are both Gaussian, the result of the expectation is another Gaussian. In general, if $p(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then the expectation of another Gaussian density ($\mathcal{N}(\mathbf{x}_{\text{new}}^\top \mathbf{w}, \sigma^2)$) is given by:

$$p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) = \mathcal{N}(\mathbf{x}_{\text{new}}^\top \boldsymbol{\mu}_{\mathbf{w}}, \sigma^2 + \mathbf{x}_{\text{new}}^\top \boldsymbol{\Sigma}_{\mathbf{w}} \mathbf{x}_{\text{new}}).$$

For the posterior shown in Figure 3.22(a), this is:

$$p(t_{\text{new}}|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) = \mathcal{N}(9.5951, 0.0572)$$

and is plotted in Figure 3.23.

This density looks rather like the predictive densities we obtained from the maximum likelihood solution in Chapter 2. However, there is one crucial difference. With the maximum likelihood we chose one particular model: the one corresponding to the highest likelihood. To generate the density shown in Figure 3.23, we have averaged over all models that are consistent with our data and prior (we averaged over our posterior). Hence this density takes into account all uncertainty that remains in \mathbf{w} given a particular prior and the data.

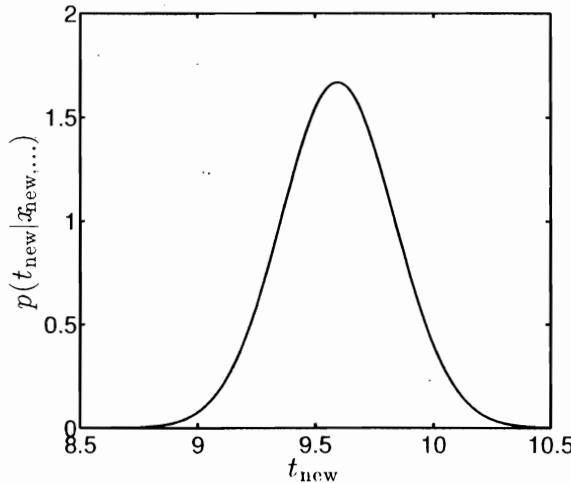


FIGURE 3.23: Predictive distribution for the winning time in the men’s 100 m sprint at the 2012 London Olympics.

3.8 Marginal likelihood for polynomial model order selection

In Section 1.5 we used a cross-validation procedure to select the order of polynomial to be used. The cross-validation procedure correctly identified that the dataset was generated from a 3rd order polynomial. In Section 3.4 we saw how the marginal likelihood could be used to choose prior densities. We will now see that it can also be used to choose models. In particular we will use it to determine which order polynomial function to use for some synthetic data.

The marginal likelihood for our Gaussian model is defined as:

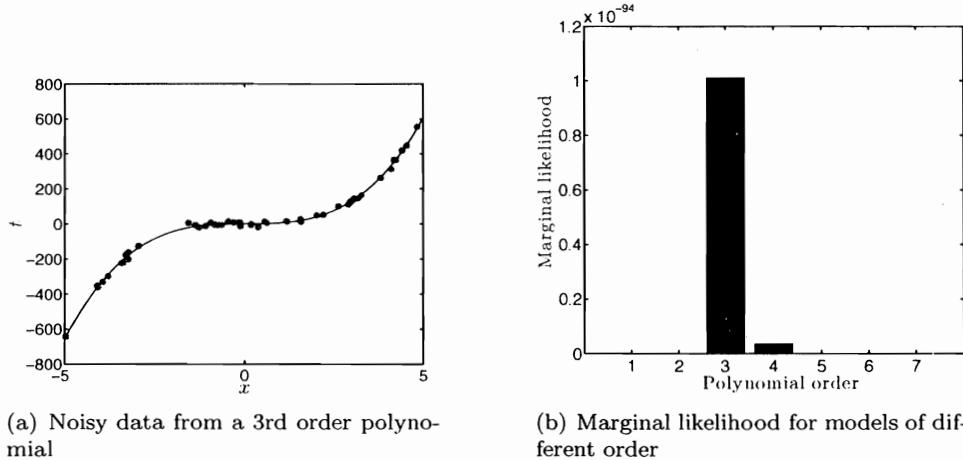
$$p(\mathbf{t}|\mathbf{X}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \int p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) p(\mathbf{w}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) d\mathbf{w}.$$

This is analogous to Equation 3.14 in the coin example. It is of the same form as the predictive density discussed in the previous section and is another Gaussian,

$$p(\mathbf{t}|\mathbf{X}, \boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) = \mathcal{N}(\mathbf{X}\boldsymbol{\mu}_0, \sigma^2\mathbf{I}_N + \mathbf{X}\boldsymbol{\Sigma}_0\mathbf{X}^\top), \quad (3.24)$$

which we evaluate at \mathbf{t} – the responses in the training set. Just as in Section 1.5, we will generate data from a noisy 3rd order polynomial and then compute the marginal likelihood for models from 1st to 7th order. For each possible model, we will use a Gaussian prior on \mathbf{w} with zero mean and an identity covariance matrix. For example, for the 1st order model

$$\boldsymbol{\mu}_0 = [0, 0]^\top, \quad \boldsymbol{\Sigma}_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



(a) Noisy data from a 3rd order polynomial

(b) Marginal likelihood for models of different order

FIGURE 3.24: Dataset sampled from the function $t = 5x^3 - x^2 + x$ (a) and marginal likelihoods for polynomials of increasing order (b).

and for the 4th order model

$$\mu_0 = [0, 0, 0, 0, 0]^T, \quad \Sigma_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The data and true polynomial are shown in Figure 3.24(a) (MATLAB script: `margpoly.m`). The true polynomial is $t = 5x^3 - x^2 + x$ and Gaussian noise has been added with mean zero and variance 150. The marginal likelihood for models from 1st to 7th order is calculated by plugging the relevant prior into Equation 3.24 and then evaluating this density at t , the observed responses. The values are shown in Figure 3.24(b). We can see that the marginal likelihood value is very sharply peaked at the true (3rd) model order. The advantage of this over the cross-validation method is that, for this model, it is computationally undemanding (we don't have to fit several different datasets). We can also use all the data. However, as we have already mentioned, calculating the marginal likelihood is, in general, very difficult and we will often find it easier to resort to cross-validation techniques.

The marginal likelihood is conditioned on the prior parameters and so changing them will have an effect on the marginal likelihood values and possibly the highest scoring model. To show the effect of this, we can define $\Sigma_0 = \sigma_0^2 \mathbf{I}$ and vary σ_0^2 . We have already seen the result for $\sigma_0^2 = 1$. If we decrease σ_0^2 , we see higher order models performing better. This can be seen in Figure 3.25. Decreasing σ_0^2 from 1 to 0.3 results in the 7th order polynomial becoming the most likely model. By decreasing σ_0^2 we are saying that the parameters have to take smaller and smaller values. For a 3rd order polynomial model to fit well, one of the parameters needs to be 5 (recall that $t = 5x^3 - x^2 + x$). As we decrease σ_0^2 , this becomes less and less likely and higher order models with lower parameter values become more likely.

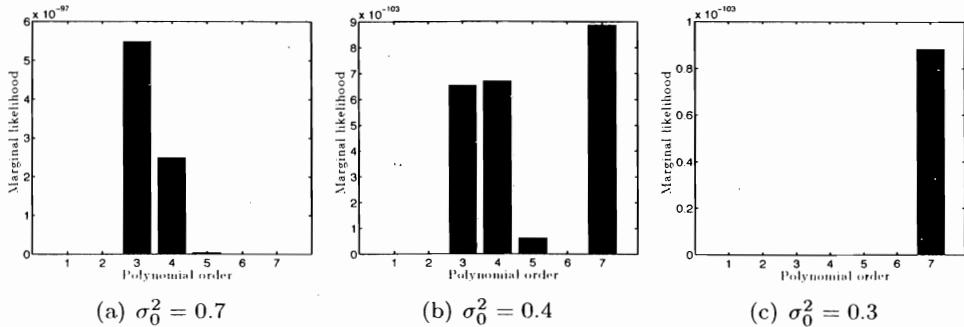


FIGURE 3.25: Marginal likelihoods for the 3rd order polynomial example with $\Sigma_0 = \sigma_0^2 \mathbf{I}$ as σ_0^2 is decreased.

This emphasises the importance of understanding what we mean by a model. In this example, the model consists of the order of polynomial *and* the prior specification and we must be careful to choose the prior sensibly (see Exercise EX 3.11).

3.9 Chapter summary

This chapter has provided an introduction to the Bayesian way of performing Machine Learning tasks – treating all parameters as random variables. We have performed a Bayesian analysis for a coin tossing model and the linear regression model introduced in Chapters 1 and 2. In both cases, we defined prior densities over parameters, defined likelihoods and computed posterior densities. In both examples, the prior and likelihood were chosen such that the posterior could be computed analytically. In addition, we computed predictions by taking expectations with respect to the posterior and introduced marginal likelihood as a possible model selection criteria.

Unfortunately, these expressions are not often analytically tractable and we must resort to sampling and approximation techniques. These techniques are the foundations of modern Bayesian inference and form an important area of Machine Learning research and development. The next chapter will describe three popular techniques – point estimates, Laplace approximations and Markov-chain Monte Carlo.

3.10 Exercises

- EX 3.1. For $\alpha, \beta = 1$, the beta distribution becomes uniform between 0 and 1. In particular, if the probability of a coin landing heads is given by r and a

beta prior is placed over r , with parameters $\alpha = 1, \beta = 1$, this prior can be written as follows:

$$p(r) = 1 \quad (0 \leq r \leq 1)$$

Using this prior, compute the posterior density for r if y heads are observed in N tosses (i.e. multiply this prior by the binomial likelihood and manipulate the result to obtain something that looks like a beta density).

- EX 3.2. Repeat the previous exercise for the following prior, also a particular form of the beta density:

$$p(r) = \begin{cases} 2r & 0 \leq r \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

What are the values of the prior parameters α and β that result in $p(r) = 2r$?

- EX 3.3. Repeat the previous exercise for the following prior (again, a form of beta density):

$$p(r) = \begin{cases} 3r^2 & 0 \leq r \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

What are the prior parameters here?

- EX 3.4. What are the effective prior sample sizes (α and β) for the previous three exercises (i.e. how many heads and tails are they equivalent to)?

- EX 3.5. If a random variable R has a beta density

$$p(r) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1} (1-r)^{\beta-1},$$

derive an expression for the expected value of r , $\mathbf{E}_{p(r)}\{r\}$. You will need the following identity for the gamma function:

$$\Gamma(n+1) = n\Gamma(n).$$

Hint: use the fact that

$$\int_{r=0}^{r=1} r^{\alpha-1} (1-r)^{\beta-1} = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} dr.$$

- EX 3.6. Using the setup in the previous exercise and the following identity:

$$\text{var}\{r\} = \mathbf{E}_{p(r)}\{r^2\} - (\mathbf{E}_{p(r)}\{r\})^2,$$

derive an expression for $\text{var}\{r\}$. You will need the gamma identity given in the previous exercise again.

- EX 3.7. At a different stall, you observe 20 tosses, of which 9 were heads. Compute the posteriors for the three scenarios, the probability of winning in each case and the marginal likelihoods.

- EX 3.8. Use Matlab to generate coin tosses where the probability of heads is 0.7. Generate 100 tosses and compute the posteriors for the three scenarios, the probabilities of winning and the marginal likelihoods.

- EX 3.9. In Section 3.7.4 we derived an expression for the Gaussian posterior for a linear model within the context of the Olympic 100 m data. Substituting $\mu_0 = [0, 0, \dots, 0]^\top$, we saw the similarity between the posterior mean

$$\mu_w = \frac{1}{\sigma^2} \left(\frac{1}{\sigma^2} \mathbf{X}^\top \mathbf{X} + \Sigma_0^{-1} \right)^{-1} \mathbf{X}^\top \mathbf{t}$$

and the regularised least squares solution

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X} + N\lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{t}.$$

For this particular example, find the prior covariance matrix Σ_0 that makes the two identical. In other words, find Σ_0 in terms of λ .

- EX 3.10. Redraw the graphical representation of the Olympic 100 m model to reflect the fact that the prior over w is actually conditioned on μ_0 and Σ_0 .
- EX 3.11. In Figure 3.25 we studied the effect reducing σ_0^2 had on the marginal likelihood. Using Matlab, investigate the effect of increasing σ_0^2 .
- EX 3.12. When performing a Bayesian analysis of the Olympic data, we assumed that σ^2 was known. If instead, we assume that w is known and an inverse Gamma prior is placed on σ^2 :

$$p(\sigma^2 | \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} (\sigma^2)^{-\alpha-1} \exp \left\{ -\frac{\beta}{\sigma^2} \right\},$$

the posterior over σ^2 will also be inverse Gamma. Derive the posterior parameters.

Further reading

- [1] Ben Calderhead and Mark Girolami. Estimating Bayes factors via thermodynamic integration and population mcmc. *Comput. Stat. Data Anal.*, 53:4028–4045, October 2009.

An article by the authors describing a novel approach for calculating the marginal likelihoods (Bayes factors) in models where it is not analytically tractable.

- [2] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC, second edition, 2004.

One of the most popular textbooks on Bayesian inference. Provides a detailed and practical description of Bayesian inference.

- [3] Michael Isard and Andrew Blake. Contour tracking by stochastic propagation of conditional density. In *European Conference on Computer Vision*, pages 343–356, Springer, 1996.

An interesting example of the use of Bayesian methods in the field of human computer interaction. The authors use a sampling technique to infer posterior probabilities over gestures being performed by users.

- [4] Michael Jordan, editor. *Learning in Graphical Models*. MIT Press, 1999.

An introduction to the field of graphical models and how to use them for learning tasks.

- [5] Christian Robert. *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*. Springer, second edition, 2007.

- [6] Tian-Rui Xu et al. Inferring signaling pathway topologies from multiple perturbation measurement of specific biochemical species. *Science Signalling*, 3(113), 2010.

A paper showing how Bayesian model selection via the marginal likelihood can be used to answer interesting scientific questions in the field of biology. It is also an interesting example of large-scale Bayesian sampling.

Chapter 4

Bayesian Inference

In the previous chapter we introduced the key concepts required to adopt a Bayesian approach to Machine Learning. Within the Bayesian framework, all unknown quantities are treated as random variables. Each parameter is described by a distribution rather than an individual value. Uncertainty in our parameter estimates is naturally channeled into any predictions we make. We saw two examples of *prior* and *likelihood* combinations that were *conjugate*, meaning that the *posterior* would be of the same form as the prior and could be computed analytically. Examples where we can justify the choice of a conjugate prior and likelihood combination are rare. In the remainder, we cannot compute the posterior and must resort to approximations. In this chapter, we will introduce three such approximation techniques.

4.1 Non-conjugate models

In the previous chapter we saw two models for which exact Bayesian inference was possible. In the first case, we were modelling the tossing of a coin and the combination of a beta prior and binomial likelihood meant that we could state that the posterior would also belong to the beta family. In the second example, a Gaussian prior coupled with a Gaussian likelihood resulted in a Gaussian posterior. The fact that we knew the form of the posterior meant that we didn't need to calculate a normalisation constant (the denominator in, for example, Equation 3.3). As long as we could find something proportional to the density of interest (i.e. proportional to a beta or a Gaussian), we could be certain that the normalisation would take care of itself. The beta-binomial and Gaussian-Gaussian combinations are not the only conjugate prior-likelihood pairs that we can use. Two other popular examples are the multinomial-Dirichlet and the gamma-Gaussian for discrete and continuous data, respectively.

For many models, it is not possible (or not justifiable from a modelling perspective) to pick a conjugate prior and likelihood and we are forced to approximate. In this chapter, we will introduce three approximation techniques through a binary classification problem. Binary classification is a common problem within Machine Learning and one for which no conjugate prior and likelihood combination exists. The three techniques that we will look at are: a point estimate, an approximate density, and sampling. All three are widely used within machine learning.

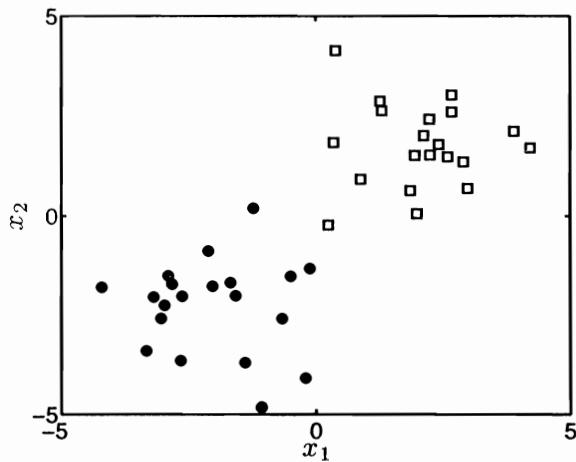


FIGURE 4.1: An example of a dataset with a binary response. Each object is defined by two attributes (x_1 and x_2) and a binary target, $t = \{0, 1\}$. Points with $t = 0$ are plotted as circles and those with $t = 1$ as squares.

4.2 Binary responses

Figure 4.1 shows a dataset that looks a bit different from those we have seen so far. Each object is described by two attributes, x_1 and x_2 , and has a binary response, $t = \{0, 1\}$. The objects are plotted with a symbol that depends on their response: if $t = 0$, the point is plotted as a circle and if $t = 1$, as a square. We will use this data to build a model that will enable us to predict the response (0 or 1; circle or square) for a new object. This task is known as classification – we want to be able to classify objects into one of a set of classes (in this case there are two classes). Classification is one of the major problems within Machine Learning and we will introduce several other classification algorithms in Chapter 5.

4.2.1 A model for binary responses

We will work with the following vector and matrix representations of our data:

$$\mathbf{x}_n = \begin{bmatrix} x_{n1} \\ x_{n2} \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}.$$

Our model (with parameters \mathbf{w}) will allow us to predict t_{new} for some new observations \mathbf{x}_{new} .

Just as in our Olympics example in Section 3.7, we will need to compute the posterior density over the parameters of the model. According to Bayes' rule this is given by:

$$p(\mathbf{w}|\mathbf{t}, \mathbf{X}) = \frac{p(\mathbf{t}|\mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{t}|\mathbf{X})} \quad (4.1)$$

where the marginal likelihood $p(\mathbf{t}|\mathbf{X})$ is given by:

$$p(\mathbf{t}|\mathbf{X}) = \int p(\mathbf{t}|\mathbf{X}, \mathbf{w})p(\mathbf{w}) d\mathbf{w}$$

Prior: We shall use a Gaussian density for the prior, $p(\mathbf{w})$. In particular, $p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$. To be consistent, given that $p(\mathbf{w})$ depends on σ^2 , we will denote the prior as $p(\mathbf{w}|\sigma^2)$. In previous chapters, the choice of a Gaussian density was often motivated by analytical convenience. Given that we are not going to be able to rely on conjugacy in this chapter, we are not restricted in our choice of prior density. However, our interest in this chapter is in the methods required to overcome non-conjugacy and for that, a Gaussian will suffice. Readers are recommended to try the methods introduced in this chapter with different forms of prior density, $p(\mathbf{w})$ exercises.

Likelihood: To make headway with the likelihood, $p(\mathbf{t}|\mathbf{X}, \mathbf{w})$, we start by assuming that the elements of \mathbf{t} are conditionally independent (see Section 2.7.1), conditioned on \mathbf{w} :

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^N p(t_n|\mathbf{x}_n, \mathbf{w}).$$

t_n is a binary variable indicating the class (0 or 1) of the n th object, \mathbf{x}_n . In the Gaussian Olympics example in the previous chapter, we treated t_n as a Gaussian random variable with mean $\mathbf{w}^\top \mathbf{x}_n$ and variance σ^2 , but this is only appropriate for real-valued t_n . Instead, we can model t_n as a binary random variable – a single coin toss for each n . Rather than a mean and variance, this random variable is characterised by the probability that the class is 1 (the probability of belonging to class 0 is 1 minus the probability of belonging to class 1). To avoid confusion, we will denote this random variable T_n (to distinguish it from the actual instance, t_n that we observe). Therefore, we can write each of the n likelihood terms as a probability:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^N P(T_n = t_n|\mathbf{x}_n, \mathbf{w}). \quad (4.2)$$

This likelihood function will be high if the model assigns high probabilities for class 1 when we observe class 1 and high probabilities for class 0 when we observe class 0. It has a maximum value of 1 where all of the training points are predicted perfectly.

Our task is now to choose a function of \mathbf{x}_n and \mathbf{w} , $f(\mathbf{x}_n; \mathbf{w})$, that produces a probability. A popular technique is to take a simple linear function (e.g. $f(\mathbf{x}_n; \mathbf{w}) = \mathbf{w}^\top \mathbf{x}_n$) and then pass the result through a second function that *squashes* its output to ensure it produces a valid probability. One such squashing function is the sigmoid function shown in Figure 4.2. As $\mathbf{w}^\top \mathbf{x}$ increases, the value converges to 1 and as it decreases, it converges to 0. The sigmoid function is defined as:

$$P(T_n = 1|\mathbf{x}_n, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)} \quad (4.3)$$

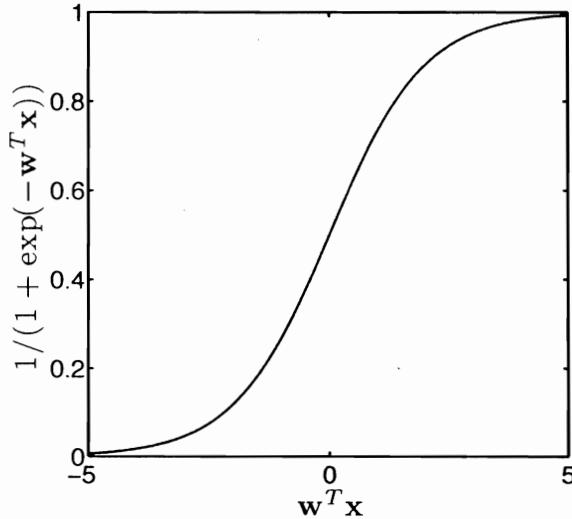


FIGURE 4.2: The sigmoid function that squashes a real value (e.g. $w^T x$) to always be between 0 and 1.

This expression gives us the probability that $T_n = 1$. In our likelihood we require the probability of the actual observation, some of which will be zero. Because T_n can only take the value 0 or 1, we can easily compute $P(T_n = 0|\mathbf{x}, \mathbf{w})$ using Equation 2.2:

$$\begin{aligned} P(T_n = 0|\mathbf{x}_n, \mathbf{w}) &= 1 - P(T_n = 1|\mathbf{x}_n, \mathbf{w}) \\ &= 1 - \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x}_n)} \\ &= \frac{\exp(-\mathbf{w}^T \mathbf{x}_n)}{1 + \exp(-\mathbf{w}^T \mathbf{x}_n)}. \end{aligned} \quad (4.4)$$

We combine Equations 4.3 and 4.4 to produce a single expression for $P(T_n = t_n|\mathbf{x}_n, \mathbf{w})$ as follows:

$$P(T_n = t_n|\mathbf{x}_n, \mathbf{w}) = P(T_n = 1|\mathbf{x}_n, \mathbf{w})^{t_n} P(T_n = 0|\mathbf{x}_n, \mathbf{w})^{1-t_n},$$

where the observed data (t_n) switches the relevant term on and the other off.

Substituting this into Equation 4.2 gives us the likelihood for all n training points:

$$\begin{aligned} p(\mathbf{t}|\mathbf{X}, \mathbf{w}) &= \prod_{n=1}^N P(T_n = 1|\mathbf{x}_n, \mathbf{w})^{t_n} P(T_n = 0|\mathbf{x}_n, \mathbf{w})^{1-t_n} \\ &= \prod_{n=1}^N \left(\frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x}_n)} \right)^{t_n} \left(\frac{\exp(-\mathbf{w}^T \mathbf{x}_n)}{1 + \exp(-\mathbf{w}^T \mathbf{x}_n)} \right)^{1-t_n} \end{aligned} \quad (4.5)$$

The posterior: This definition of the likelihood combined with the Gaussian prior we chose earlier is all we need, in theory, to work out the posterior density,

$p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$. Once we have the posterior density, we can predict the response (class) of new objects by taking an expectation with respect to this density:

$$P(t_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) = \mathbf{E}_{p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)} \left\{ \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_{\text{new}})} \right\}.$$

In practice, this is not straightforward. The posterior is not of any standard form. To be able to evaluate it at a particular \mathbf{w} , we would need to evaluate both the numerator and denominator of Equation 4.1. The numerator is fine – we could evaluate the Gaussian prior density at \mathbf{w} and the likelihood that we've just defined and multiply the two values together. The denominator is the problem, as we cannot analytically perform the integration required to compute the marginal likelihood:

$$Z^{-1} = p(\mathbf{t}|\mathbf{X}, \sigma^2) = \int p(\mathbf{t}|\mathbf{X}, \mathbf{w})p(\mathbf{w}|\sigma^2) d\mathbf{w}$$

In other words, we have a function $g(\mathbf{w}; \mathbf{X}, \mathbf{t}, \sigma^2) = p(\mathbf{t}|\mathbf{X}, \mathbf{w})p(\mathbf{w}|\sigma^2)$ which we know is proportional to the posterior, $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2) = Z^{-1}g(\mathbf{w}; \mathbf{X}, \mathbf{t}, \sigma^2)$, but we do not know the constant of proportionality, Z^{-1} (note that this constant is traditionally defined as Z^{-1} rather than Z). We are left with three options:

1. Find the single value of \mathbf{w} that corresponds to the highest value of the posterior. As $g(\mathbf{w}; \mathbf{X}, \mathbf{t}, \sigma^2)$ is proportional to the posterior, a maximum of $g(\mathbf{w}; \mathbf{X}, \mathbf{t}, \sigma^2)$ will also correspond to a maximum of the posterior. Z^{-1} is not a function of \mathbf{w} .
2. Approximate $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$ with some other density that we can compute analytically.
3. Sample directly from the posterior $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$, knowing only $g(\mathbf{w}; \mathbf{X}, \mathbf{t}, \sigma^2)$.

The first option is not very Bayesian – we will have to make predictions for new objects based on a single value of \mathbf{w} and not a density. It is, however, easy to do and this makes it a popular technique. The second option leaves us with a density that is easy to work with (we can choose any density we like) but if the chosen density is very different from the posterior, our model will not be very reliable. The final option allows us to sample from the posterior (and hence get good approximations to any expectations that we might require) but can be difficult.

These are the three options that are open to us in any problem where we cannot directly compute the posterior density. All three options have good and bad points and the choice of one over another will depend on the specifications (and computational limitations) of the problem at hand. We will now describe each in turn.

4.3 A point estimate – the MAP solution

In the previous section we showed that whilst we could not compute the posterior density $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$, we could compute something proportional to it, $g(\mathbf{w}; \mathbf{X}, \mathbf{t}, \sigma^2)$. This is equal to the prior multiplied by the likelihood. The value of \mathbf{w} that maximises

$g(\mathbf{w}; \mathbf{X}, \mathbf{t}, \sigma^2)$ will also correspond to the value at the maximum of the posterior. This will be the single most likely value of $\hat{\mathbf{w}}$ (under the posterior) and is a sensible choice if we decide to use a point estimate. Chapter 2 was devoted to finding the value of $\hat{\mathbf{w}}$ that maximised the likelihood. The idea here is very similar except now we are maximising the likelihood multiplied by the prior. This solution is the maximum a posteriori estimate (MAP) estimate that we first saw in Section 3.7.4 and is common within machine learning.

Comment 4.1 – The Newton–Raphson method: The Newton–Raphson method (also known as the Newton method) is a general method for finding points where functions are equal to zero, i.e. finding points where the function $f(x) = 0$. Given a current estimate of the zero point, x_n , we update it by moving to the point where the tangent to the function at x_n passes through the x -axis. This point can be computed by approximating the gradient as a change in $f(x)$ divided by a change in x . Defining $\partial f(x)/\partial x$ as $f'(x)$:

$$\begin{aligned} f'(x_n) &= \frac{f(x_n) - 0}{x_n - x_{n+1}} \\ (x_n - x_{n+1})f'(x_n) &= f(x_n) \\ x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)}. \end{aligned}$$

The method can also be used to find minima and maxima, as these are simply points where the gradient passes through zero. Therefore, we simply replace $f(x)$ with its derivative $f'(x)$ and $f'(x)$ with its derivative $f''(x)$:

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}.$$

This is readily extendable to functions of a vector – say \mathbf{x} . In this instance, $f'(x_n)$ is replaced by the vector of partial derivatives evaluated at \mathbf{x}_n and $1/f''(x_n)$ is replaced by the inverse of the Hessian matrix (see Comment 2.6) $-\partial^2 f(\mathbf{x})/\partial \mathbf{x} \partial \mathbf{x}^\top$ – evaluated at $\mathbf{x} = \mathbf{x}_n$.

As with finding the maximum likelihood solution, it is easiest to find the value of \mathbf{w} that maximises $\log g(\mathbf{w}; \mathbf{X}, \mathbf{t})$ rather than $g(\mathbf{w}; \mathbf{X}, \mathbf{t})$:

$$\log g(\mathbf{w}; \mathbf{X}, \mathbf{t}) = \log p(\mathbf{t}|\mathbf{X}, \mathbf{w}) + \log p(\mathbf{w}|\sigma^2).$$

Unlike the maximum likelihood solution for the linear model, we cannot obtain an exact expression for \mathbf{w} by differentiating this expression and equating it to zero. Instead, we can use any one of many optimisation algorithms that start with a guess for \mathbf{w} and then keep updating it in such a way that $g(\mathbf{w}; \mathbf{X}, \mathbf{t})$ increases until a maximum is reached. The Newton-Raphson procedure (see Comment 4.1) is one such method that updates \mathbf{w} using the following equation:

$$\mathbf{w}' = \mathbf{w} - \left(\frac{\partial^2 \log g(\mathbf{w}; \mathbf{X}, \mathbf{t})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \right)^{-1} \frac{\partial \log g(\mathbf{w}; \mathbf{X}, \mathbf{t})}{\partial \mathbf{w}} \quad (4.6)$$

The new version (\mathbf{w}') of \mathbf{w} is calculated by subtracting the inverse of the Hessian (see Comment 2.6) multiplied by the vector of partial derivatives. For any starting

value of \mathbf{w} , this iterative procedure will update \mathbf{w} until it reaches a point where the gradient is zero. To check that the point we have converged to corresponds to a maximum, we can check the Hessian to ensure that it is negative definite, just as we did for maximum likelihood in Section 2.7.3.

In order to compute the vector of first derivatives we first expand our expression for $\log g(\mathbf{w}; \mathbf{X}, \mathbf{t})$ using Equations 4.2 and 4.5:

$$\begin{aligned}\log g(\mathbf{w}; \mathbf{X}, \mathbf{t}) &= \sum_{n=1}^N \log P(T_n = t_n | \mathbf{x}_n, \mathbf{w}) + \log p(\mathbf{w} | \sigma^2) \\ &= \sum_{n=1}^N \log \left(\frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)} \right)^{t_n} \left(\frac{\exp(-\mathbf{w}^\top \mathbf{x}_n)}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)} \right)^{1-t_n} \\ &\quad + \log p(\mathbf{w} | \sigma^2)\end{aligned}$$

To stop this expression from becoming too complicated, we will use the following shorthand:

$$P_n = P(T_n = 1 | \mathbf{w}, \mathbf{x}_n) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)}.$$

Therefore, assuming that \mathbf{w} is D -dimensional, we have the following expression:

$$\begin{aligned}\log g(\mathbf{w}; \mathbf{X}, \mathbf{t}) &= \log p(\mathbf{w} | \sigma^2) + \sum_{n=1}^N \log P_n^{t_n} + \log(1 - P_n)^{1-t_n} \\ &= -\frac{D}{2} \log 2\pi - D \log \sigma - \frac{1}{2\sigma^2} \mathbf{w}^\top \mathbf{w} \\ &\quad + \sum_{n=1}^N t_n \log P_n + (1 - t_n) \log(1 - P_n),\end{aligned}$$

where the first three terms are the log of the (Gaussian) prior. To find the vector of partial derivatives, we can use the chain rule (see Comment 4.2) to give an expression in terms of the partial derivatives of P_n :

$$\begin{aligned}\frac{\partial \log g(\mathbf{w}; \mathbf{X}, \mathbf{t})}{\partial \mathbf{w}} &= -\frac{1}{\sigma^2} \mathbf{w} + \sum_{n=1}^N \left(\frac{t_n}{P_n} \frac{\partial P_n}{\partial \mathbf{w}} + \frac{1 - t_n}{1 - P_n} \frac{\partial(1 - P_n)}{\partial \mathbf{w}} \right) \\ &= -\frac{1}{\sigma^2} \mathbf{w} + \sum_{n=1}^N \left(\frac{t_n}{P_n} \frac{\partial P_n}{\partial \mathbf{w}} - \frac{1 - t_n}{1 - P_n} \frac{\partial P_n}{\partial \mathbf{w}} \right),\end{aligned}\tag{4.7}$$

where we have used the chain rule a second time to turn $\frac{\partial(1 - P_n)}{\partial \mathbf{w}}$ into $-\frac{\partial P_n}{\partial \mathbf{w}}$:

$$\begin{aligned}\frac{\partial(1 - P_n)}{\partial \mathbf{w}} &= \frac{\partial(1 - P_n)}{\partial P_n} \frac{\partial P_n}{\partial \mathbf{w}} \\ &= -\frac{\partial P_n}{\partial \mathbf{w}}.\end{aligned}$$

To calculate $\frac{\partial P_n}{\partial \mathbf{w}}$ we can use the chain rule once more:

$$\begin{aligned}
 \frac{\partial P_n}{\partial \mathbf{w}} &= \frac{\partial (1 + \exp(-\mathbf{w}^\top \mathbf{x}_n))^{-1}}{\partial (1 + \exp(-\mathbf{w}^\top \mathbf{x}_n))} \frac{\partial (1 + \exp(-\mathbf{w}^\top \mathbf{x}_n))}{\partial \mathbf{w}} \\
 &= -\frac{1}{(1 + \exp(-\mathbf{w}^\top \mathbf{x}_n))^2} \exp(-\mathbf{w}^\top \mathbf{x}_n) (-\mathbf{x}_n) \\
 &= \frac{\exp(-\mathbf{w}^\top \mathbf{x}_n)}{(1 + \exp(-\mathbf{w}^\top \mathbf{x}_n))^2} \mathbf{x}_n \\
 &= \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)} \frac{\exp(-\mathbf{w}^\top \mathbf{x}_n)}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)} \mathbf{x}_n \\
 &= P_n (1 - P_n) \mathbf{x}_n. \tag{4.8}
 \end{aligned}$$

Comment 4.2 – The chain rule: When taking partial derivatives, it is often convenient to use the chain rule. The chain rule states that:

$$\frac{\partial f(g(\mathbf{w}))}{\partial \mathbf{w}} = \frac{\partial f(g(\mathbf{w}))}{\partial g(\mathbf{w})} \frac{\partial g(\mathbf{w})}{\partial \mathbf{w}}.$$

As an example, let

$$f(\mathbf{w}) = t_n \log P_n$$

where

$$P_n = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x}_n)}.$$

To compute $\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}$, we can use the chain rule as follows:

$$\frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = \frac{\partial f(\mathbf{w})}{\partial P_n} \frac{\partial P_n}{\partial \mathbf{w}} = \frac{t_n}{P_n} \frac{\partial P_n}{\partial \mathbf{w}}.$$

Substituting Equation 4.8 into Equation 4.7 gives us the required vector of partial derivatives:

$$\begin{aligned}
 \frac{\partial \log g(\mathbf{w}; \mathbf{X}, \mathbf{t})}{\partial \mathbf{w}} &= -\frac{1}{\sigma^2} \mathbf{w} + \sum_{n=1}^N (\mathbf{x}_n t_n (1 - P_n) - \mathbf{x}_n (1 - t_n) P_n) \\
 &= -\frac{1}{\sigma^2} \mathbf{w} + \sum_{n=1}^N \mathbf{x}_n (t_n - t_n P_n - P_n + t_n P_n) \\
 &= -\frac{1}{\sigma^2} \mathbf{w} + \sum_{n=1}^N \mathbf{x}_n (t_n - P_n). \tag{4.9}
 \end{aligned}$$

To compute the Hessian matrix of second derivatives, we differentiate this again with respect to \mathbf{w}^\top . Noting that $\frac{\partial P_n}{\partial \mathbf{w}^\top} = \left(\frac{\partial P_n}{\partial \mathbf{w}}\right)^\top$ we obtain the following expression:

$$\begin{aligned}
 \frac{\partial^2 \log g(\mathbf{w}; \mathbf{X}, \mathbf{t})}{\partial \mathbf{w} \partial \mathbf{w}^\top} &= -\frac{1}{\sigma^2} \mathbf{I} - \sum_{n=1}^N \mathbf{x}_n \frac{\partial P_n}{\partial \mathbf{w}^\top} \\
 &= -\frac{1}{\sigma^2} \mathbf{I} - \sum_{n=1}^N \mathbf{x}_n \mathbf{x}_n^\top P_n (1 - P_n). \tag{4.10}
 \end{aligned}$$

One thing to notice from the Hessian is that because $0 \leq P_n \leq 1$, it will be negative definite for any set of \mathbf{x}_n and for any \mathbf{w} (see Section 2.7.3). Therefore,

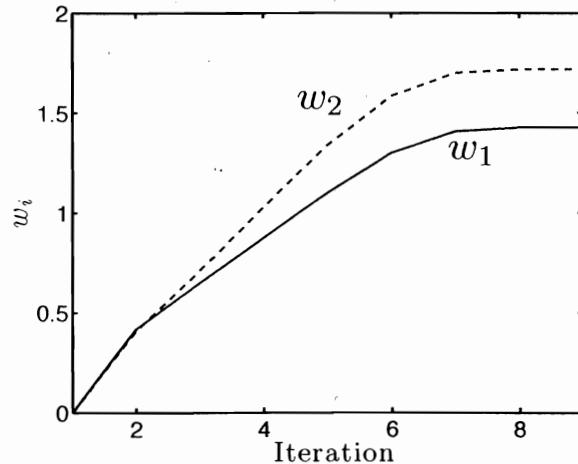


FIGURE 4.3: Evolution of the components of \mathbf{w} throughout the Newton-Raphson procedure to find the \mathbf{w} corresponding to the maximum of the posterior density.

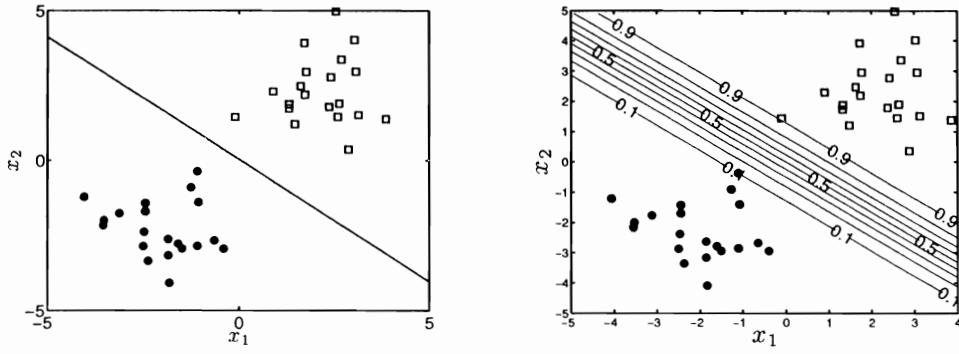
there can only be one optimum and it must be a maximum. Whatever value of \mathbf{w} the Newton-Raphson procedure converges to must correspond to the highest value of the posterior density. This is a consequence of the choice of prior and likelihood function and changing either may result in a harder posterior density to optimise.

We now have everything we need to perform the Newton-Raphson procedure and find a potential optimal value of \mathbf{w} . Starting with $\mathbf{w} = [0, 0]^\top$ and setting $\sigma^2 = 10$, the procedure converges (the change in \mathbf{w} becomes insignificant) after only nine iterations (MATLAB script: `logmap.m`). The evolution of the two components of \mathbf{w} over this period can be seen in Figure 4.3. Following the previous chapters, we will call the value of \mathbf{w} that corresponds to the maximum $\hat{\mathbf{w}}$.

Using $\hat{\mathbf{w}}$ we can compute the probability that the response equals 1 for any \mathbf{x} . In particular, if we observe \mathbf{x}_{new} , a new set of attributes, the probability that it should be given a response of 1 (it belongs to the square class) is given by

$$P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \hat{\mathbf{w}}) = \frac{1}{1 + \exp(-\hat{\mathbf{w}}^\top \mathbf{x}_{\text{new}})}. \quad (4.11)$$

Given that there are two possible responses (or classes) for this new object, a sensible strategy might be to assign it to the square class ($T_{\text{new}} = 1$) if the probability is greater than 0.5 and to the circle class ($T_{\text{new}} = 0$) otherwise. In this case, the set of \mathbf{x} values that correspond to $P(T = 1 | \mathbf{x}, \hat{\mathbf{w}}) = 0.5$ will form a line that can be thought of as a **decision boundary** – points on one side of the line will belong to one class, and points on the other side to the other class. To plot the decision boundary, we make use of the fact that $P(T = 1 | \mathbf{x}, \hat{\mathbf{w}}) = 0.5$ implies that $\hat{\mathbf{w}}^\top \mathbf{x} = 0$ (see Exercise EX 4.5). If we expand this expression, we can obtain the decision



(a) The data and the line where $P(T = 1|\mathbf{x}, \hat{\mathbf{w}}) = 0.5$. If we took 0.5 as a threshold, new points above the line would be classified as squares, below as circles

(b) The contours show $P(T = 1|\mathbf{x}, \hat{\mathbf{w}})$, the probability that a new object should be classified as a square, as a function of \mathbf{x}

FIGURE 4.4: Inferred function in the binary response example.

boundary as a function of x_1 and x_2 :

$$\begin{aligned} 0 &= \hat{\mathbf{w}}^T \mathbf{x} \\ &= \hat{w}_1 x_1 + \hat{w}_2 x_2 \\ \hat{w}_2 x_2 &= -\hat{w}_1 x_1 \\ x_2 &= -\frac{\hat{w}_1 x_1}{\hat{w}_2}, \end{aligned}$$

which is plotted in Figure 4.4(a). If we want to split the two classes with a straight line, this seems like quite a reasonable choice. In Figure 4.4(b) we plot contours of $P(T = 1|\mathbf{x}, \hat{\mathbf{w}})$ as a function of \mathbf{x} (MATLAB script: `logmap.m`). Close to the squares the probability is 1 (the squares are objects for which $t_n = 1$) and close to the circles it is 0. Between the two groups of data, the probability is around 0.5, reflecting the fact that objects here would be equidistant from both groups.

The outcome of this optimisation is that we have a model with which we can make predictions. The model is based on a point estimate, $\hat{\mathbf{w}}$, of the parameters that we have obtained by finding the value of \mathbf{w} that corresponds to a maximum of the posterior, $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$. This MAP solution is common in Machine Learning because it is reasonably easy to find $\hat{\mathbf{w}}$ in this way. One could follow the steps described above for any prior and likelihood combination and find an optimum value. The optimisations will not always be as well behaved as this – in some problems, the posterior might have several maxima (and maybe even some minima). It would be difficult to know if the maximum we had found using Newton-Raphson was the **global optimum**.

In Chapter 3 we have already seen the advantage of maintaining a density over \mathbf{w} rather than collapsing onto a point estimate. With this in mind, we will now move onto our second option when faced with a posterior we cannot compute exactly – finding a density that approximates $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$.

4.4 The Laplace approximation

There are various approximation methods used within Machine Learning to replace tricky posterior densities with approximations that are easier to handle. The most popular is the Laplace approximation.¹ The idea is to approximate the density of interest with a Gaussian. Given the ease with which we can manipulate Gaussians, this seems to be a sensible choice – the expectations required to make predictions are likely to be easy to calculate given a Gaussian posterior. However, we should always bear in mind that our predictions will then only be as good as our approximation. If our true posterior is not very Gaussian, our predictions will be easy to compute but not very useful.

The Gaussian density is defined by its mean and (co)variance. Using a Gaussian to approximate another density amounts to choosing suitable values for these parameters. To motivate the choices of parameters made by the Laplace approximation, imagine that, rather than having two parameters, our model only has one – w – and that we know \hat{w} – the value corresponding to the highest value of the posterior. Our first step is to approximate $\log g(w; \mathbf{X}, \mathbf{t}, \sigma^2)$ using a Taylor expansion (see Comment 4.3) around the maximum, \hat{w} :

$$\begin{aligned}\log g(w; \mathbf{X}, \mathbf{t}, \sigma^2) &\approx \log g(\hat{w}; \mathbf{X}, \mathbf{t}, \sigma^2) + \frac{\partial \log g(w; \mathbf{X}, \mathbf{t}, \sigma^2)}{\partial w} \Big|_{\hat{w}} \frac{(w - \hat{w})}{1!} \\ &\quad + \frac{\partial^2 \log g(w; \mathbf{X}, \mathbf{t}, \sigma^2)}{\partial w^2} \Big|_{\hat{w}} \frac{(w - \hat{w})^2}{2!} + \dots\end{aligned}$$

The second term is the first derivative (i.e. the gradient) evaluated at the maximum point and must therefore be zero. Discarding this and ignoring terms of 3rd order and above, we are left with the following expression:

$$\log g(w; \mathbf{X}, \mathbf{t}, \sigma^2) \approx \log g(\hat{w}; \mathbf{X}, \mathbf{t}, \sigma^2) - \frac{v}{2}(w - \hat{w})^2, \quad (4.12)$$

where v is the negative of the second derivative of $\log g(w; \mathbf{X}, \mathbf{t}, \sigma^2)$ evaluated at $w = \hat{w}$:

$$v = - \frac{\partial^2 \log g(w; \mathbf{X}, \mathbf{t}, \sigma^2)}{\partial w^2} \Big|_{\hat{w}}.$$

Now, the Gaussian density is defined as:

$$\frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{1}{2\sigma^2}(w - \mu)^2 \right\},$$

the log of which is equal to:

$$\log(K) - \frac{1}{2\sigma^2}(w - \mu)^2,$$

where K is the normalising constant. This looks very similar to Equation 4.12 with

¹Technically, it is actually a saddle-point approximation but has come to be known as the Laplace approximation within Machine Learning. In Computational Statistics, the Laplace approximation is a name given to something else entirely.

$\mu = \hat{w}$ and $\sigma^2 = 1/v$. This is the Laplace approximation – we approximate the posterior with a Gaussian that has its mean at the posterior mode (\hat{w}) and has variance inversely proportional to the curvature of the posterior (its second derivative) at its mode.

Comment 4.3 – Taylor expansions: The Taylor expansion is a way of approximating a function. The approximation is always made ‘about’ some value – the approximation will tend to diverge from the true function as we move away from that value. The definition of the Taylor series of $f(w)$ about \hat{w} is:

$$f(w) = \sum_{n=0}^{\infty} \frac{(w - \hat{w})^n}{n!} \left. \frac{\partial^n f(w)}{\partial w^n} \right|_{\hat{w}}$$

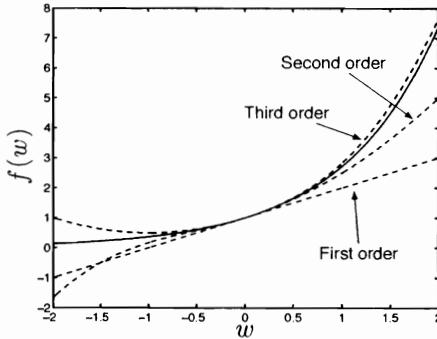
where $\left. \frac{\partial^n f(w)}{\partial w^n} \right|_{\hat{w}}$ is the n th derivative of $f(w)$ with respect to w , evaluated at \hat{w} . When $n = 0$, this derivative is simply the function $f(w)$. If we only compute a finite number of terms, we will have an approximation to the function. A first order approximation would just include terms $n = 0$ and $n = 1$ – an n th order approximation includes all terms up to and including term n . For example, we can approximate $f(w) = \exp(w)$ at $\hat{w} = 0$:

$$\exp(w) = \exp(\hat{w}) + \frac{w}{1!} \exp(\hat{w}) + \frac{w^2}{2!} \exp(\hat{w}) + \dots$$

Now, $\exp(\hat{w}) = 1$, so:

$$\exp(w) = 1 + \frac{w}{1!} + \frac{w^2}{2!} + \frac{w^3}{3!} + \dots$$

The approximation will get better and better as we add more and more terms. This can be seen in the figure on the right.



This idea is easily extended to multivariate densities. In particular, the Laplace approximation to our true posterior $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$ is:

$$p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2) \approx \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}),$$

where $\boldsymbol{\mu}$ is set to $\hat{\mathbf{w}}$ and $\boldsymbol{\Sigma}$ is the negative of the inverse Hessian:

$$\boldsymbol{\mu} = \hat{\mathbf{w}}, \quad \boldsymbol{\Sigma}^{-1} = - \left(\left. \frac{\partial^2 \log g(\mathbf{w}; \mathbf{X}, \mathbf{t})}{\partial \mathbf{w} \partial \mathbf{w}^\top} \right) \right|_{\hat{\mathbf{w}}} \quad (4.13)$$

4.4.1 Laplace approximation example: Approximating a gamma density

Before we look at what this approximation looks like in the binary response example it is useful to look at an example where we know the true density (see also

Exercises EX 4.1, EX 4.2 and EX 4.3) (MATLAB script: `lapexample.m`). This will allow us to see how good or bad the approximation is. The following is the gamma density for a random variable Y :

$$p(y|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} \exp\{-\beta y\}. \quad (4.14)$$

We will investigate how good the Laplace approximation is to this density. The gamma density has an analytic expression for its mode which means we do not need to go through an optimisation procedure similar to that in the last section. The mode, \hat{y} , is defined as:

$$\hat{y} = \frac{\alpha - 1}{\beta}.$$

The Laplace approximation to $p(y|\alpha, \beta)$ takes the form of a Gaussian:

$$p(y|\alpha, \beta) \approx \mathcal{N}(\mu, \sigma^2).$$

The mean μ will be equal to the mode of $p(y|\alpha, \beta)$, which we've already defined. To find the variance, σ^2 , of the approximating Gaussian, we need to find the second derivative of $\log p(y|\alpha, \beta)$ with respect to y . This is computed as follows:

$$\begin{aligned} \log p(y|\alpha, \beta) &= \alpha \log \beta - \log(\Gamma(\alpha)) + (\alpha - 1) \log y - \beta y \\ \frac{\partial \log p(y|\alpha, \beta)}{\partial y} &= \frac{\alpha - 1}{y} - \beta \\ \frac{\partial^2 \log p(y|\alpha, \beta)}{\partial y^2} &= -\frac{\alpha - 1}{y^2}. \end{aligned}$$

σ^2 will be equal to the negative inverse of this quantity evaluated at $y = \hat{y}$. In particular:

$$\sigma^2 = \frac{\hat{y}^2}{\alpha - 1} = \frac{\alpha - 1}{\beta^2}.$$

In Figure 4.5 we can see two examples of $p(y|\alpha, \beta)$ and the corresponding Laplace approximation. In the first, $p(y|\alpha, \beta)$ looks rather like a Gaussian and the approximation is pretty good. In the second, $p(y|\alpha, \beta)$ does not look very much like a Gaussian and the approximation is not accurate. In both cases the approximation gets worse as we move away from the mode. This is because the approximation is based on the characteristics of the function *at* the mode. We will see this property again as we return to the binary response model

4.4.2 Laplace approximation for the binary response model

Returning to our binary response model, we had to compute both the mode, \hat{w} , and the Hessian for the Newton–Raphson procedure. We therefore already have everything we need for the Laplace approximation to the posterior $p(w|X, t, \sigma^2)$. In Figure 4.6(a) we can see the approximate posterior and in Figure 4.6(b) we can see the same approximation on top of $g(w; X, t)$, the unnormalised posterior. As for the gamma example in the previous section, the shape of the approximation is pretty good around the mode but diverges considerably from the true posterior as we move away from the mode. This is to be expected – the Laplace approximation only

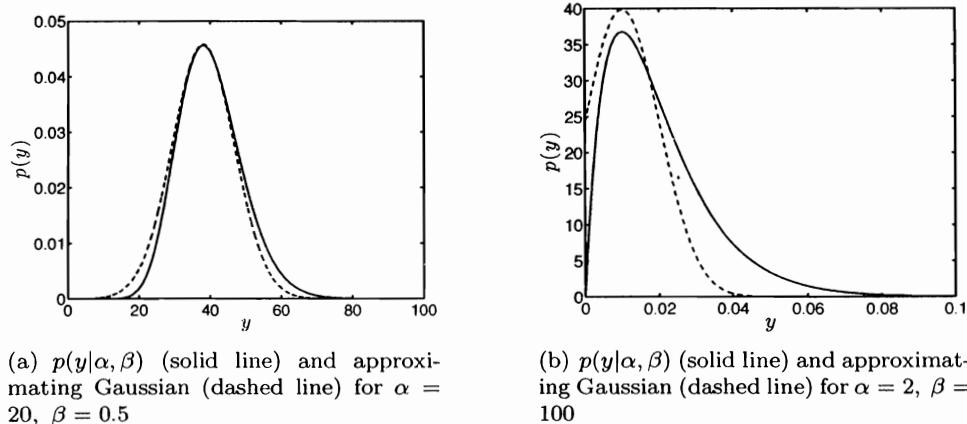


FIGURE 4.5: Examples of the Laplace approximation to the gamma density function given in Equation 4.14.

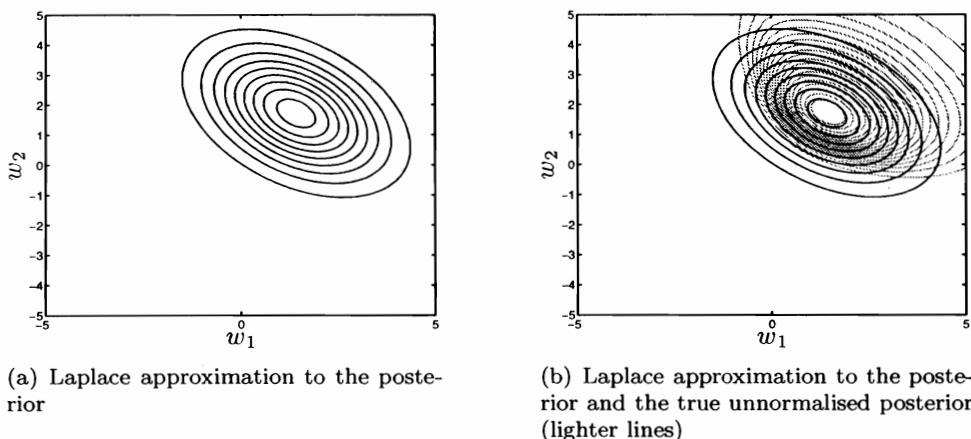


FIGURE 4.6: The Laplace approximation for the binary problem.

matches the shape (curvature) at the mode. We can also sample values of \mathbf{w} from the approximate posterior and look at the decision boundaries that they correspond to. Twenty such boundaries are plotted in Figure 4.7(a). There appears to be a lot of variability in these boundaries, although all of them seem to split the classes reasonably well.

The final step is to use the approximate posterior to compute predictions. We now have a density over \mathbf{w} rather than a single value and we know, from Chapter 3, that we compute a prediction by averaging over this density. In particular, we should be calculating the expected value of $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})$ with respect to the approximate posterior over \mathbf{w} (which we've denoted as $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$):

$$P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) = \mathbf{E}_{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})} \{ P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}) \} .$$

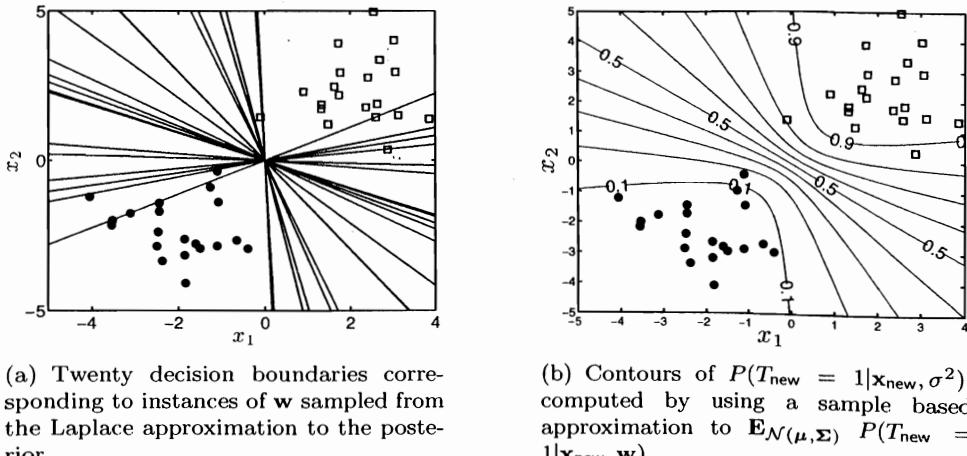


FIGURE 4.7: Decision boundaries sampled from the Laplace approximation and the predictive probability contours.

Unfortunately, we cannot compute the integral over \mathbf{w} required in this expectation. This might suggest that our choice of approximation was not sensible – we still cannot make predictions. However, we can easily sample from $\mathcal{N}(\mu, \Sigma)$ and so (see Equation 2.23) we can approximate the expectation with:

$$P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) = \frac{1}{N_s} \sum_{s=1}^{N_s} \frac{1}{1 + \exp(-\mathbf{w}_s^\top \mathbf{x}_{\text{new}})}, \quad (4.15)$$

where \mathbf{w}_s is the s th of N_s samples drawn from the approximate posterior. Using $N_s = 1000$, the contours of $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2)$ can be seen in Figure 4.7(b) (MATLAB script: `loglap.m`). Compare this with Figure 4.4(b). There is a big difference – the contours are no longer straight lines. Averaging over the posterior density for \mathbf{w} has had the effect of smudging the decision boundaries. The probabilities are now closer to 0.5 in all areas except those very close to the data objects. The model based on the point estimate, shown in Figure 4.4(b), could be said to be over-confident – take $x_1 = -3, x_2 = 5$ as an example. According to the predictions produced by the point estimate (Figure 4.4(b)), an object with these attributes would have a probability of approximately 1 of being a square despite the fact that it is quite distant from the other square objects. Compare this with the probability of approximately 0.6 given by the expectation with respect to the Laplace approximation to the posterior (Figure 4.7(b)). This value seems much more reasonable. Another way to understand the uncertainty that should be present in areas like this is to look at Figure 4.7(a) – there is very large variability in the possible decision boundaries at $x_1 = -3, x_2 = 5$. Some of these boundaries would classify this object as a square, some as a circle – the probability that it is a square, given the data that we have seen, is not 1.

In this section we have seen again that we should be wary of using point estimates. The Laplace approximation shown here can be used to approximate any

density (over real-valued random variables) for which we can find the mode and compute the second derivative. The approach assumes that the posterior can be reasonably approximated by a Gaussian, something that is not always the case (see Figure 4.5). In our binary response model, the approximation did not allow us to compute the expectation necessary for making predictions exactly. However, the ease with which we can sample from a Gaussian meant that it was straightforward to obtain a sample-based approximation to the expectation. In the next section, we will extend this idea through the introduction of a technique that will enable us to sample directly from $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$ despite the fact that we cannot compute the normalisation constant. The ability to generate these samples will allow us to use a sample-based approximation to the expectation without having to approximate the posterior.

4.5 Sampling techniques

The Laplace approximation in the previous section provided us with a method for approximating the posterior density $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$. Our interest in the posterior density is primarily to allow us to take all the uncertainty in \mathbf{w} into account when making predictions. We do this by averaging over all potential values of \mathbf{w} through the following expectation:

$$P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) = \mathbf{E}_{p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)} \{ P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}) \}.$$

Even substituting our approximation to the posterior into this expression, we could not analytically compute the integral required in this expectation. Fortunately, it was easy to sample from the Gaussian approximation, enabling us to use the sample based approximation given in Equation 4.15. In this instance, the benefit of making the approximation was that it enabled us to easily generate samples. In this section, we will look at a technique that enables us to cut out the approximation step and sample directly from the posterior. A set of samples from the true posterior generated in this way could be substituted directly into Equation 4.15 to compute the desired predictive probability, $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2)$. We're going to introduce a popular sampling technique known as the **Metropolis–Hastings** algorithm. However, before we go into this, it is perhaps useful to get more comfortable with the idea of sampling through a less abstract example.

4.5.1 Playing darts

In the game of darts, players take turns to throw three darts at a board like that shown in Figure 4.8. The darts are sharp and embed themselves into the board. The player receives a certain number of points for each dart, depending on where the dart lands. The scores from the three darts are added together and subtracted from the player's current total. Each player starts the game with the same total (normally 501) and the winner is the player who gets to zero first. The majority of the board is split into 20 segments and if the dart lands in the white parts of these segments, the score is equal to the number shown around the edge. If the dart lands

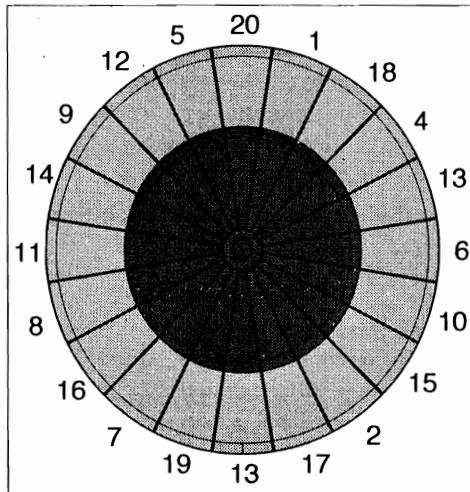


FIGURE 4.8: A dartboard.

in one of the shaded areas, the score is either double (lighter, outer shaded area) or triple (darker, inner area) the segment score. The circle in the centre of the board is known as the bull's-eye (50 points) and the circle around this as the bull (25 points). There is one slight complication to the rules – the player must get to zero with a double. So, for example, if a player currently has a total of 40, they could win by throwing a double 20 (the lightly shaded area just below the '20' label) or a single 20 (anywhere in the white bits of the '20' segment) followed by a double 10, etc. We will assume that the player does indeed need to score 40 to win, and has only one dart left with which to do it. In other words, they need to hit the double 20 – what is the probability that they will succeed?

Assume that there exists some probability density function defined over the position that the dart will land. In other words, when a player is aiming for, say, double 20, the position at which the dart will land could be considered as an instance of some random variable. We will use the vector \mathbf{y} to describe this position and therefore the density will look something like $p(\mathbf{y}|\Delta)$. Δ ought to depend (at least to some extent) on where the player is aiming. The extent to which this dependence exists depends on the players skill. For a professional trying to hit double 20, we might expect the density to be tightly concentrated around the double 20 area. For a poor player, aiming might make very little difference to where the dart ends up. So, Δ depends on the skill of the player and the strength of their technique – making $p(\mathbf{y}|\Delta)$ very hard to define.

At this point it would be easy to give up. But, taking a step backwards, we are not directly interested in $p(\mathbf{y}|\Delta)$, just the probability that the player throws a double 20. Do we need to be able to write down an analytic expression for $p(\mathbf{y}|\Delta)$ to work this out? Before we answer this, let us satisfy ourselves that we could work it out if we could write down $p(\mathbf{y}|\Delta)$. Define the random variable $T = f(\mathbf{y})$ where $f(\mathbf{y})$ is 1 if \mathbf{y} is inside the double 20 region and zero otherwise. T depends on \mathbf{y} and hence depends on Δ . So, we're interested in the following probability: $P(T = 1|\Delta)$. This is nothing more than an expectation. In particular, it looks rather like the expectations

we had to compute for the binary response model in the previous section:

$$P(T = 1|\Delta) = \mathbf{E}_{p(\mathbf{y}|\Delta)} \{f(\mathbf{y})\} = \int f(\mathbf{y})p(\mathbf{y}|\Delta) d\mathbf{y}. \quad (4.16)$$

In theory, if we could write down $p(\mathbf{y}|\Delta)$, we could work this out. However, we have also seen that we can compute quantities like this with a sample-based approximation. In particular, if \mathbf{y}_s is the s th of N_s samples from $p(\mathbf{y}|\Delta)$, our approximation would look like:

$$P(T = 1|\Delta) \simeq \frac{1}{N_s} \sum_{s=1}^{N_s} f(\mathbf{y}_s).$$

So, we do not need to be able to write down $p(\mathbf{y}|\Delta)$ to be able to compute $P(T = 1|\Delta)$ as long as we can sample from it. Fortunately, sampling from $p(\mathbf{y}|\Delta)$ is pretty easy – we get our player, some darts and a board and we ask the player to aim for double 20. The position of each dart thrown is a sample from $p(\mathbf{y}|\Delta)$. If we record \mathbf{y}_s for each of N_s throws, we can compute the sample-based approximation given in Equation 4.16. In fact, in this case it works out as just the proportion of times the player throws a double 20).

We can explicitly relate this procedure to our binary response model. First, the quantity of interest in the darts case, $P(T = 1|\Delta)$, is analogous to the predictive probability in the binary response model: $P(T_{\text{new}} = 1|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2)$. In both cases to compute this quantity, we must take an expectation with respect to some density: our darts distribution $p(\mathbf{y}|\Delta)$ is analogous to $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$ – the posterior density over our parameters. In the darts case, we approximated this expectation by drawing samples directly from the posterior (despite the fact that we couldn't write it down). In the binary response case, we approximated the posterior with something we could sample from and then sampled. We will now see how we can sample directly from $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$ (see Exercise EX 4.4).

4.5.2 The Metropolis–Hastings algorithm

In this section, we will introduce the Metropolis–Hastings² (MH) algorithm. Rather than go into too much detail we will introduce it as a recipe, describing the steps involved without proving why they work. References to further reading are provided at the end of the chapter.

Recall that we are attempting to sample from $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$ so that we can approximate the following expectation:

$$\begin{aligned} P(T_{\text{new}} = 1|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) &= \mathbf{E}_{p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)} \{P(T_{\text{new}} = 1|\mathbf{x}_{\text{new}}, \mathbf{w})\} \\ &= \int P(T_{\text{new}} = 1|\mathbf{x}_{\text{new}}, \mathbf{w})p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2) d\mathbf{w}, \end{aligned}$$

with

$$P(T = 1|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) \simeq \frac{1}{N_s} \sum_{s=1}^{N_s} P(T_{\text{new}} = 1|\mathbf{x}_{\text{new}}, \mathbf{w}_s).$$

²Named after Nicholas Metropolis and W. Keith Hastings – a physicist and statistician, respectively, who developed the technique to tackle problems in an area of physics known as statistical mechanics.

Metropolis–Hastings generates a sequence of samples $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_{s-1}, \mathbf{w}_s, \dots, \mathbf{w}_{N_s}$. Generating a sample (say \mathbf{w}_s) consists of two steps. In the first step, we need to propose a new sample – a candidate for \mathbf{w}_s . This is performed by proposing a movement from the previous sample (\mathbf{w}_{s-1}). Second, the proposed sample is tested to see whether or not it should be accepted. If accepted, it becomes our new sample \mathbf{w}_s . If it is not accepted, our new sample is set to be equal to the previous one: $\mathbf{w}_s = \mathbf{w}_{s-1}$. This is continued until we have collected what we believe to be enough samples.

Now, if our proposal is based on a movement from the previous sample, what do we do for our first sample \mathbf{w}_1 ? It turns out that it doesn't matter where we start – \mathbf{w}_1 can be anything. As long as we sample for long enough, our sampler is guaranteed to converge to the distribution of interest. So, we can pluck a \mathbf{w}_1 from anywhere (sampling it from the prior would probably be a sensible choice), set the Metropolis–Hastings algorithm off, wait for it to converge to the correct distribution and then harvest as many samples as we need. A word of caution: the sampler is guaranteed to converge *in theory*. In practice, it is important to use one (or ideally more) of the methods available to test convergence before we start harvesting samples. We will now look at the proposal and acceptance steps in more detail.

Proposing a new sample: Assume that we have already sampled $s - 1$ values using the MH scheme. We will propose a sample based on a movement from \mathbf{w}_{s-1} . Calling our proposed sample $\widetilde{\mathbf{w}}_s$ (we can only call it \mathbf{w}_s once it has been accepted), we need to define a density:

$$p(\widetilde{\mathbf{w}}_s | \mathbf{w}_{s-1}).$$

This density does not have to have any connection with the posterior $p(\mathbf{w} | \mathbf{X}, \mathbf{t}, \sigma^2)$ from which we're trying to sample. We are free to define it as we please. In practice, the choice will have an impact on how long it will take the MH sampler to converge. A common choice is to use a Gaussian centred on the current sample, \mathbf{w}_{s-1} :

$$p(\widetilde{\mathbf{w}}_s | \mathbf{w}_{s-1}, \Sigma) = \mathcal{N}(\mathbf{w}_{s-1}, \Sigma)$$

Sampling a sequence of values like this creates what is known as a **random walk**. In Figure 4.9 we show two such walks (MATLAB script: `randwalks.m`). One starts from $\mathbf{w}_1 = [0, 0]^\top$ and has covariance $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ whilst the other starts from $\mathbf{w}_1 = [2, 2]^\top$

and has covariance $\Sigma = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$. The latter walk moves less distance in each step due to the smaller diagonal (variance) elements in the covariance matrix. As we have already mentioned, the Gaussian is a popular choice for the proposal density. One reason is the ease with which we can sample from it – choosing a proposal distribution that was hard to sample from would make things unnecessarily complicated. Another reason is that it is symmetric: moving to $\widetilde{\mathbf{w}}_s$ from \mathbf{w}_{s-1} is just as likely as moving from $\widetilde{\mathbf{w}}_s$ to \mathbf{w}_{s-1} :

$$p(\widetilde{\mathbf{w}}_s | \mathbf{w}_{s-1}, \Sigma) = p(\mathbf{w}_{s-1} | \widetilde{\mathbf{w}}_s, \Sigma).$$

We will see the advantage of this as we move on to the acceptance step.

Accepting or rejecting: We now have $\widetilde{\mathbf{w}}_s$, a candidate for \mathbf{w}_s . We must now decide whether we should accept it or reject it. To do this, we compute the following ratio:

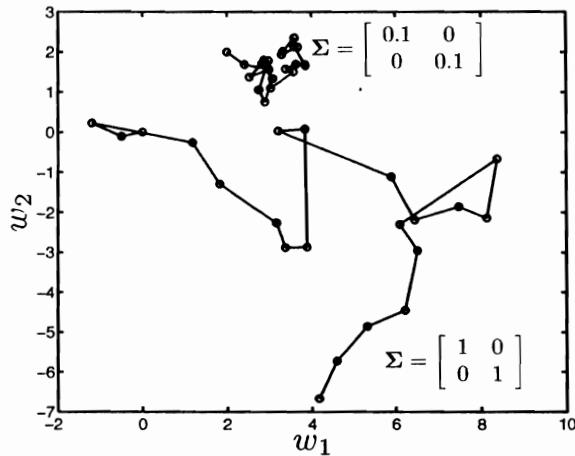


FIGURE 4.9: Two examples of random walks where the distribution over the next location is a Gaussian centred at the current location. The two walks have different covariance matrices, shown in the plot.

$$r = \frac{p(\tilde{\mathbf{w}}_s | \mathbf{X}, \mathbf{t}, \sigma^2)}{p(\mathbf{w}_{s-1} | \mathbf{X}, \mathbf{t}, \sigma^2)} \frac{p(\mathbf{w}_{s-1} | \tilde{\mathbf{w}}_s, \Sigma)}{p(\tilde{\mathbf{w}}_s | \mathbf{w}_{s-1}, \Sigma)}. \quad (4.17)$$

This is the ratio of the posterior density at the *proposed* sample to that at the old sample multiplied by the ratio of the proposal densities. The symmetry of the Gaussian proposal distribution discussed above allows us to ignore this last term, as it is always equal to 1. The first term is the ratio of posterior densities evaluated at the two different parameter values. We cannot compute the densities exactly because we cannot normalise them. However, because we are interested in a ratio, the normalisation constants cancel. So, we can substitute the ratio of posteriors with the ratio of the priors multiplied by the ratio of likelihoods. This leads us to the following expression:

$$r = \frac{g(\tilde{\mathbf{w}}_s; \mathbf{X}, \mathbf{t}, \sigma^2)}{g(\mathbf{w}_{s-1}; \mathbf{X}, \mathbf{t}, \sigma^2)} = \frac{p(\tilde{\mathbf{w}}_s | \sigma^2)}{p(\mathbf{w}_{s-1} | \sigma^2)} \frac{p(\mathbf{t} | \tilde{\mathbf{w}}_s, \mathbf{X})}{p(\mathbf{t} | \mathbf{w}_{s-1}, \mathbf{X})}$$

This ratio will always be positive, as the density functions are always positive. If it is 1 or greater, we accept the sample ($\mathbf{w}_s = \tilde{\mathbf{w}}_s$). If r is less than 1, we accept the sample with probability equal to r . In other words, if we propose a set of parameters that corresponds to a higher value of the posterior density than \mathbf{w}_{s-1} , we always accept it ($r > 1$). If we propose a set that corresponds to a lower value of posterior density, we accept it sometimes, but not always. The algorithm is depicted in Figure 4.10. Notice that we have described the accept/reject step in more detail. If $r < 1$, we should accept with probability r . This is achieved by drawing a value (u) from a uniform distribution between 0 and 1. Because it is uniform, the probability that u will be less than or equal to r is equal to r . Hence, we accept the proposal if $u \leq r$ and reject otherwise. The whole process is best illustrated with an example.

Figure 4.11 shows the Metropolis-Hastings algorithm in action, sampling from an arbitrary density (indicated by contours) (MATLAB script: `mhexample.m`). The

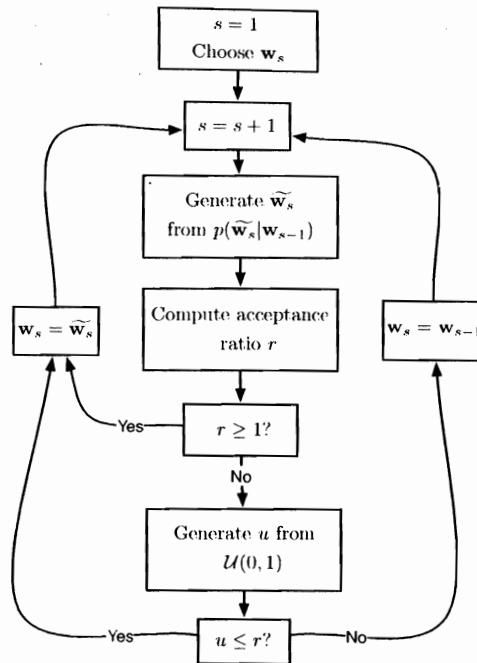


FIGURE 4.10: The Metropolis–Hastings algorithm.

starting point, w_1 , is shown in Figure 4.11(a). Our proposal density is Gaussian with $\Sigma = \mathbf{I}$. From the starting point, the first proposal is made, \tilde{w}_2 , shown in Figure 4.11(b). The proposal causes an increase in posterior density and is therefore accepted: $w_2 = \tilde{w}_2$. This acceptance is indicated by the solid line in Figure 4.11(b). The next proposal, \tilde{w}_3 , causes a slight decrease in posterior density but is accepted nonetheless (remember that if the proposal causes a decrease, there is still a probability of acceptance). This is shown by the new solid line in Figure 4.11(c). The next proposal, \tilde{w}_4 , causes a large decrease in the posterior density value. Such a proposal is highly unlikely to be accepted (the ratio is much less than 1) and in this instance it isn't. This is represented by the dashed line in Figure 4.11(c). Hence, $w_4 \neq \tilde{w}_4$ and is instead set to $w_4 = w_3$. This process continues in Figures 4.11(d) and 4.11(e), by which time we have 10 samples. Along the way, three proposals were rejected and in each of those instances the sample is set to be equal to the value of the previous (accepted) sample. Continuing this process, we can see the first 300 accepted samples in Figure 4.11(f). These samples look reasonably consistent with the density contours – samples seem to be more concentrated towards the centre of the density and very sparse around the edges.

The density we are sampling from in this example happens to be a Gaussian. So, we can go some way towards convincing ourselves that we are indeed sampling from the correct density by computing the mean and covariance of the samples and seeing if they correspond to the mean and covariance of the actual density. The

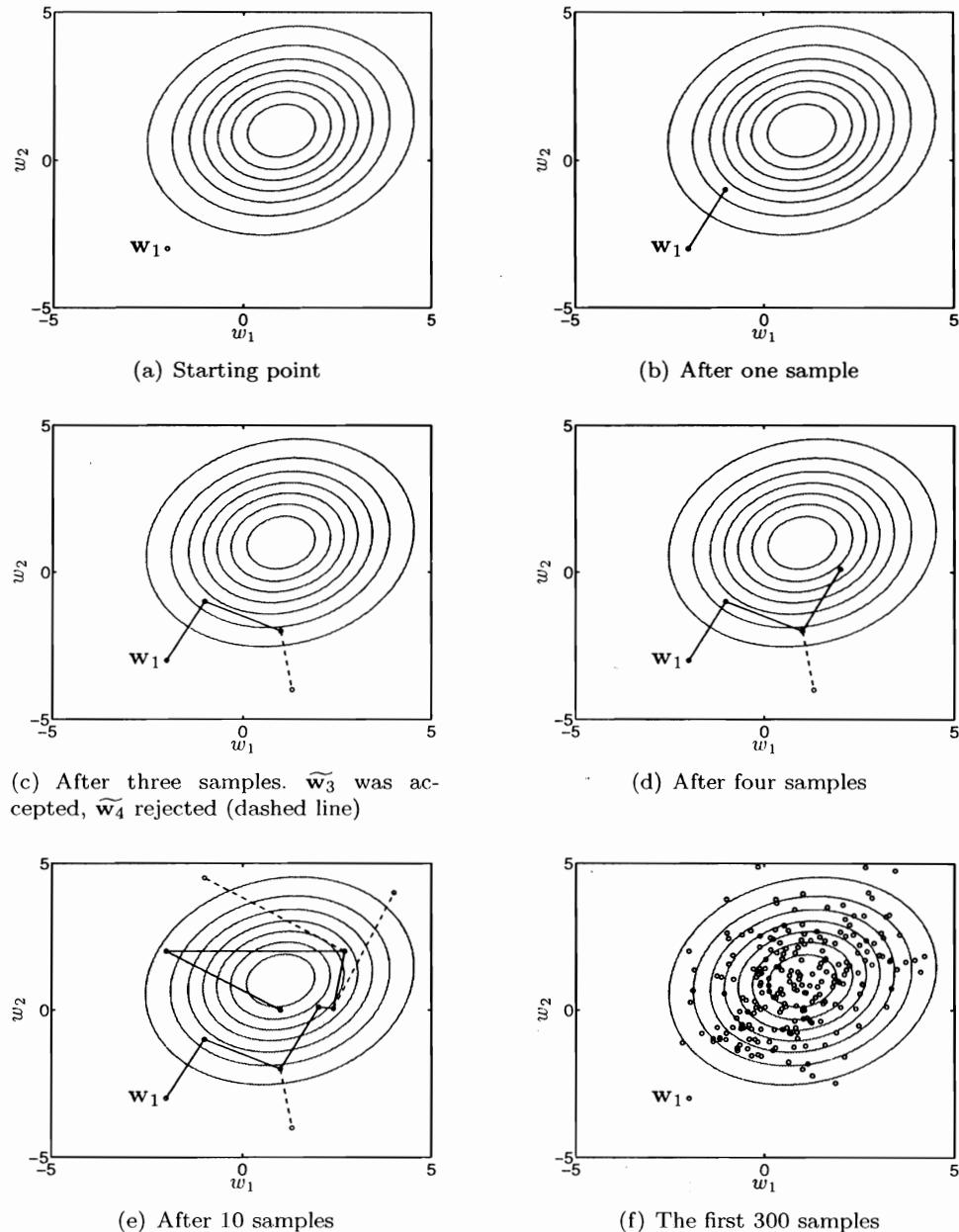


FIGURE 4.11: Example of the Metropolis-Hastings algorithm in operation.

actual mean and covariance are given by:

$$\boldsymbol{\mu} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} 3 & 0.4 \\ 0.4 & 3 \end{bmatrix}$$

After $N_s = 10,000$ samples, we can compute the sample-based approximations to the mean and covariance ($\boldsymbol{\mu}'$, \mathbf{S}') as follows:

$$\boldsymbol{\mu}' = \frac{1}{N_s} \sum_{s=1}^{N_s} \mathbf{w}_s, \quad \mathbf{S}' = \frac{1}{N_s} \sum_{s=1}^{N_s} (\mathbf{w}_s - \boldsymbol{\mu}')(\mathbf{w}_s - \boldsymbol{\mu}')^\top.$$

These work out as:

$$\boldsymbol{\mu}' = \begin{bmatrix} 0.9770 \\ 1.0928 \end{bmatrix}, \quad \mathbf{S}' = \begin{bmatrix} 3.0777 & 0.4405 \\ 0.4405 & 2.8983 \end{bmatrix}$$

which are both very similar to the true values.

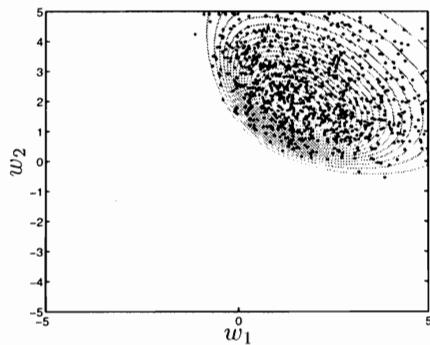
Before we move on to applying MH to our binary response model, we need to discuss two related concepts – **burn-in** and **convergence**. As we can start our sampler from anywhere (there is no restriction on \mathbf{w}_1), we don't necessarily know if we are starting the sampler in an area that we should be generating samples from (it might be an area of very low posterior density). Therefore, the first few samples may not be representative and should be discarded. This period between the starting point and convergence of the sampler is known as the burn-in period. Sadly, it is not possible to conclusively determine how long this period should be. In the example described above, it is no more than a couple of samples, but in some applications it could easily be hundreds or thousands. To overcome this problem we need a method for determining convergence. This is not convergence to a particular *value*, but convergence to a particular *distribution*. In other words, are the samples we are seeing coming from the correct distribution?

A popular method is to start several samplers simultaneously from different starting points. When all of the samplers are generating samples with similar characteristics (mean, variance, etc.), it suggests that they have all converged to the same distribution – the one we are trying to sample from.

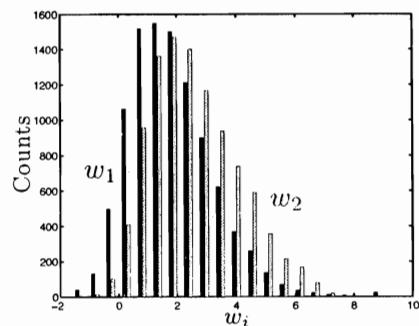
We will now return to our binary response model. Using the MH scheme described above, we generate 10000 samples from $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$ (MATLAB script: `logmh.m`). Our proposal density is a Gaussian with $\boldsymbol{\Sigma} = \gamma^2 \mathbf{I}$ where $\gamma^2 = 0.5$. In Figure 4.12(a) we show every 10th sample (plotting all 10,000 samples makes for a very crowded plot) along with the posterior contours. The samples and the contours look reasonably coherent. If we like, we can use the samples to create marginal posterior densities for the two individual parameters. Recall from Section 3.4.1 that to marginalise w_2 from the posterior we would need to integrate (sum if the random variable is discrete) over all values w_2 could take:

$$p(w_1|\mathbf{X}, \mathbf{t}, \sigma^2) = \int p(w_1, w_2|\mathbf{X}, \mathbf{t}, \sigma^2) dw_2,$$

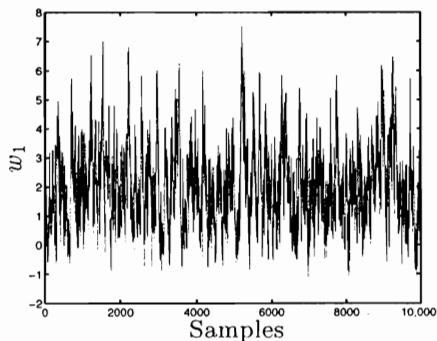
where $p(w_1, w_2|\mathbf{X}, \mathbf{t}, \sigma^2)$ is another way of writing $p(\mathbf{w}|\mathbf{X}, \mathbf{t}, \sigma^2)$. To get a sample-based approximation we take each of our samples, \mathbf{w}_s , and ignore w_2 . In other words, if we throw away the value of w_2 from each sample, we are left with a set of samples from $p(w_1|\mathbf{X}, \mathbf{t}, \sigma^2)$. In Figures 4.12(b)–4.12(d), we show three popular



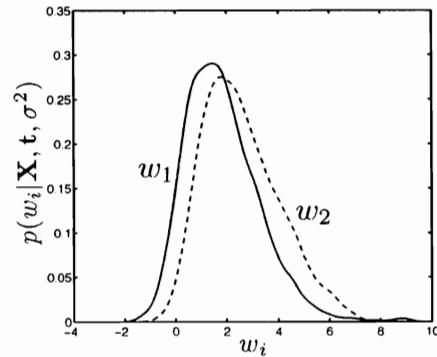
(a) 1000 of the MH samples along with the posterior contours



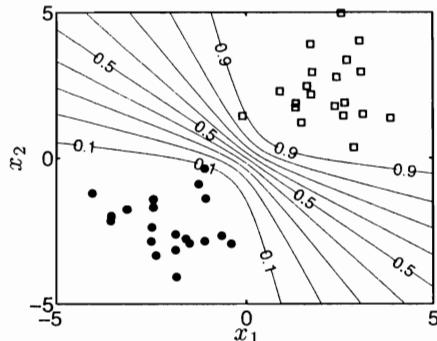
(b) Histograms of the samples for both w_1 (black) and w_2 (grey)



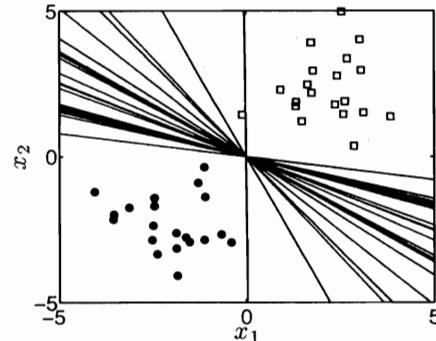
(c) All of the w_1 samples plotted against iteration, s



(d) Continuous densities fitted to the w_1 and w_2 samples



(e) Predictive probability contours. The contours show the probability of classifying an object at any location as a square. The probability of classifying an object as a circle at any point is 1 minus this value



(f) Decision boundaries created from 20 randomly selected MH samples

FIGURE 4.12: Results of applying the MH sampling algorithm to the binary response model.

ways of visualising these samples. In the first, (Figure 4.12(b)) we have split the range of possible values into 20 sections and counted the number of samples that fall in each section. The black bars show the numbers for w_1 and the grey bars for w_2 . If we were to take the number of samples falling into a particular section and divide it by the total number of samples, the numbers obtained could be thought of as the posterior probabilities that the w_1 (or w_2) falls into each of these sections. In the second example (Figure 4.12(c)), we have just plotted all 10,000 samples for w_1 (a similar plot for w_2 looks almost identical). This plot gives us confidence that the sampler has converged very quickly. If it hadn't, we might see an overall increasing or decreasing trend. In Figure 4.12(d) we show two continuous density functions that have been fitted to the samples. This is, in itself, a Machine Learning task for which there are various possible solutions. If the samples looked like they had come from a Gaussian, we could fit Gaussian densities to the two sets of samples (see Exercise EX 2.8). In this example, we have used a more general technique known as kernel density estimation. This can be performed in Matlab using the `ksdensity` function. We won't go into any more detail here – the important point is that there are many ways to visualise the samples and it is possible to turn them into (approximate) continuous density functions.

Finally we turn our attention back to the predictive probability, $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2)$. When using the Laplace approximation, we approximated this quantity by drawing samples $\mathbf{w}_1, \dots, \mathbf{w}_{N_s}$ from the approximate posterior and then computing:

$$P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}, \sigma^2) = \frac{1}{N_s} \sum_{s=1}^{N_s} \frac{1}{1 + \exp(-\mathbf{w}_s^T \mathbf{x}_{\text{new}})}.$$

We now have a set of samples from the true posterior, $p(\mathbf{w} | \mathbf{X}, \mathbf{t}, \sigma^2)$ and we can use them in exactly the same way. Figure 4.12(e) shows the predictive probability contours computed using these true posterior samples. Remember that these contours give the probability of classifying an object at any particular location as a square. The shape of the contours looks rather like the shape in Figure 4.7(b), which is not very surprising, as we saw in Figure 4.6(b) that the Laplace approximation didn't look too different from the true posterior. The only noticeable difference is that the contours in Figure 4.12(e) are slightly less tightly curved around the areas in which the data lie. This suggests that the probability reduces rather more slowly as we move away from the squares. The MH sampler is sampling from the true posterior and so the contours in Figure 4.12(e) should be considered closer to the truth than those for the Laplace approximation in Figure 4.6(a). This comparison is really just giving us an indication of how good the Laplace approximation is *at making predictions*. Figure 4.12(f) shows the decision boundaries corresponding to 20 of the MH samples picked at random (c.f. Figure 4.7(a)) (see Exercises EX 4.6, EX 4.7 and EX 4.8).

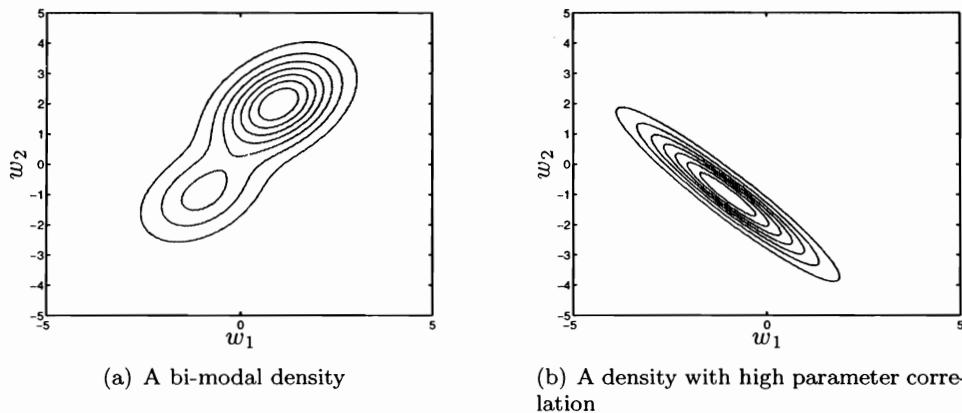


FIGURE 4.13: Two densities that would be tricky to sample from with MH.

4.5.3 The art of sampling

The Metropolis-Hastings algorithm seems to work well for our binary response model. This will not always be the case – sampling methods (like MH) can be tricky to use. The difficulty lies in the (often unknown) shape of the density from which we are attempting to sample. Consider the density shown in Figure 4.13(a). This density has two modes, one at $\mathbf{w} = [-1, -1]^T$ and one at $\mathbf{w} = [2, 2]^T$. MH likes to move towards the modes as these are moves that increase the posterior density and are therefore always accepted. Imagine \mathbf{w}_s somewhere near the mode at $\mathbf{w} = [2, 2]^T$. To move from here to the mode at $\mathbf{w} = [-1, -1]^T$ would require many downhills in a row. Although this is possible, it is incredibly unlikely. This is where the relationship between the theory and practice of algorithms like MH breaks down. In theory, we will move from the top of one mode to the top of the other at some point in the future (just because it is unlikely doesn't mean that it won't happen eventually). In practice, we might get very old while we wait. We will happily explore one of the modes without ever realising that the other exists.

A second problem is illustrated by the density in Figure 4.13(b). Here, we have a density with only one mode, but the two variables w_1 and w_2 are very dependent or highly correlated. If the value of w_1 is known, it is possible to narrow w_2 down to quite a small range. Densities such as this make it very difficult to choose proposal distributions: $p(\tilde{\mathbf{w}}_s | \mathbf{w}_{s-1})$. Pick any position on the density in Figure 4.13(b) and imagine proposing a movement based on a Gaussian density with a diagonal covariance matrix (as we used in all of our examples) which would have circular contours when plotted. The density shown in Figure 4.13(b) is far from circular and because the shape of the proposal density is so different from the shape of the density we are attempting to sample from, many samples will be rejected: the vast majority of moves that we sample from our proposal will involve moving steeply down the probability gradient.

These are not the only problems. For example, how do we know when we have taken enough samples? How do we know how many samples we need to discard at the beginning? Fortunately, there are ways to overcome all of these problems: more sophisticated algorithms, ways of choosing proposal densities, quantities that we can

evaluate that indicate convergence, etc. Further details are provided in the suggested reading.

4.6 Summary

The motivation for this chapter was the desire to do things in the Bayesian way when we are not able to compute the distributions of interest analytically. We have shown examples of three general techniques. First, finding the highest point of the posterior (the MAP estimate). This is a single value, and single values are not very Bayesian but it incorporates the prior and could therefore be considered a step up from the maximum likelihood solution. The second approach was to approximate the posterior with another density. We chose the Laplace approximation, which approximates the posterior with a Gaussian. In many applications, this density could be used to compute the required expectations (predictions) analytically. In our binary response application, the expectation was not analytically tractable, but sampling from a Gaussian is easy and so we approximated (again) with a sample-based approximation. The third approach involved using the Metropolis-Hastings algorithm to generate samples from the true posterior which could be used to compute expectations. This comes at additional computational cost, but (in theory at least) we get predictions that reflect the true posterior.

4.7 Exercises

- EX 4.1. For a data set consisting of N observations \mathbf{x}_n (each of which is D -dimensional) and *real* valued targets t_n , a linear regression model is defined as:

$$p(t_n | \mathbf{x}_n, \mathbf{w}) = \mathcal{N}(\mathbf{w}^\top \mathbf{x}_n, 1)$$

Making the standard IID assumption and assuming a Gaussian prior over the D -dimensional parameters \mathbf{w} , show that the Laplace approximation is equal to the true posterior.

- EX 4.2. In Chapter 3 we computed the posterior density over r , the probability of a coin giving heads, using a beta prior and a binomial likelihood. Recalling that the beta prior, with parameters α and β , is given by

$$p(r | \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} r^{\alpha-1} (1-r)^{\beta-1}$$

and the binomial likelihood, assuming y heads in N throws, is given by

$$p(y|r, N) = \binom{y}{N} r^y (1-r)^{N-y}$$

compute the Laplace approximation to the posterior. (Note, you should

be able to obtain a closed-form solution for the MAP value, \hat{r} , by setting the log posterior to zero, differentiating and equating to zero).

EX 4.3. Plot the true beta posterior and the Laplace approximation computed in Exercise EX 4.2 for various values of α , β , y and N .

EX 4.4. Given the expression for the area of a circle, $A = \pi r^2$, and using only uniformly distributed random variates, devise a sampling approach for computing π .

EX 4.5. Re-arrange the logistic function:

$$P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \hat{\mathbf{w}}) = \frac{1}{1 + \exp(-\hat{\mathbf{w}}^\top \mathbf{x}_{\text{new}})}$$

to show that $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \hat{\mathbf{w}}) = 0.5$ implies $\hat{\mathbf{w}}^\top \mathbf{x}_{\text{new}} = 0$.

EX 4.6. Assume that we observe N vectors of attributes $\mathbf{x}_1, \dots, \mathbf{x}_N$ and associated integer counts t_1, \dots, t_N . A Poisson likelihood would be suitable:

$$p(t_n | \mathbf{x}_n, \mathbf{w}) = \frac{V_n^{t_n} \exp\{-V_n\}}{t_n!}$$

where $V_n = \exp(\mathbf{w}^\top \mathbf{x}_n)$

Assuming a Gaussian prior on \mathbf{w} , derive the gradient and Hessian needed to use a Newton-Raphson routine to find the MAP solution for the parameters \mathbf{w} .

EX 4.7. Derive the Laplace approximation for the model in Exercise EX 4.6.

EX 4.8. Implement a Metropolis-Hastings sampling scheme for the model of Exercise EX 4.6 and compare the posterior with the Laplace approximation derived in Exercise EX 4.7.

Further reading

- [1] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael Jordan. An introduction to mcmc for machine learning. *Machine Learning*, 50:5–43, 2003.

A tutorial introduction to MCMC techniques from the specific point of view of Machine Learning.
- [2] Siddhartha Chib. Understanding the Metropolis–Hasting algorithm. *The American Statistician*, 49(4):327–335, 1995.

An excellent tutorial on the Metropolis–Hasting algorithm. A good starting point to look deeper into this family of algorithms.
- [3] Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer, 2010.

We have not covered sequential Monte Carlo techniques in this book, but they are becoming increasingly popular for performing Bayesian inference in complex models, particularly models with a temporal component, like target tracking.
- [4] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, second edition, 2004.

This is an excellent resource for practical Bayesian inference. In particular, it provides a solid introduction to other sampling techniques as well as procedures for determining if Metropolis-Hastings and other sampling algorithms have converged.
- [5] W.R. Gilks, S. Richardson, and D. Spiegelhalter, editors. *Markov Chain Monte-Carlo in Practice*. Chapman and Hall/CRC, 2005.

An edited volume providing several interesting practical sampling examples.
- [6] Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

A recent paper by the authors describing a sophisticated Metropolis algorithm for sampling from distributions with complex forms.
- [7] Jun Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, 2008.
- [8] Carl Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

In this chapter we briefly mentioned the non-parametric Gaussian process as an alternative to the parametric model. This book provides a comprehensive introduction to the use of Gaussian processes for both classification and regression.

- [9] Simon Rogers, Richard Scheltema, Mark Girolami, and Rainer Breitling. Probabilistic assignment of formulas to mass peaks in metabolomics experiments. *Bioinformatics*, 25(4):512–518, 2009.

A paper by the authors that described an alternative Bayesian sampling method (Gibbs sampling) being applied to the problem of detecting metabolites in mass spectrometry experiments.

- [10] Michael Tipping and Alex Smola. Sparse Bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

An example of both regression and classification based on a linear model. The Laplace approximation is used in the classification example.

- [11] Christopher Williams and David Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998.

One of the first papers to use Gaussian processes for classification. A good introduction into some real approximate Bayesian inference.

Chapter 5

Classification

In the previous chapters, we have introduced many of the main concepts that underpin Machine Learning methods. We have seen how, for a particular model, we can choose parameters and make predictions based on observed data. This has been done in three ways – finding the parameters that minimise a loss function, finding those that maximise a likelihood function and by treating the parameters as random variables. We will meet some of these approaches again in this and subsequent chapters as we tackle the main algorithmic families that make up the field of Machine Learning: classification, clustering and projection.

In this chapter we will deal with classification. The field of machine learning can boast of many classification algorithms and this set grows on a daily basis. We have chosen to introduce just four algorithms here. The four comprise a broad foundation to classification techniques in general and knowledge of these will enable the reader to solve a wide range of classification problems and explore the rest of the literature.

The four algorithms that we cover can be split into two types – those that produce probabilistic outputs and those that produce non-probabilistic outputs. Both types have their advantages and the choice will always be dataset dependent.

5.1 The general problem

Typically, we will be presented with a set of N training objects, $\mathbf{x}_1, \dots, \mathbf{x}_N$. Each is a vector with dimension D . For each object we are also provided with a label t_n that will describe which class object n belongs to. This label will typically take an integer value. For example, if there are two classes in our data, $t_n = \{0, 1\}$ or $t_n = \{-1, 1\}$. More generally, if there are C classes, $t_n = \{1, 2, \dots, C\}$. Our task is to predict the class t_{new} for an unseen object \mathbf{x}_{new} .

It is worthwhile drawing parallels between this setup and the one we saw in Chapters 1 to 3. In those chapters, we were provided with a set of objects x_1, \dots, x_N and associated real-valued labels. For many examples, the objects were Olympics years and the labels (responses) winning times for the men's 100 m sprint. Our aim was to predict the winning time for future Olympics games. The classification setup is very similar – it is just that in classification the response variable is an integer indicating a particular class rather than a real value. In fact, we have already seen an example of a classifier in Chapter 4. The binary response model is a well-known binary classification algorithm known as logistic regression.

Classification algorithms have been used successfully in many domains. Two particularly challenging examples are automatic disease diagnosis, where we are in-

terested in predicting whether a patient is healthy or unhealthy based on medical observations and text classification, where we are interested in classifying documents into topics or as relevant/irrelevant for a particular user. These two examples illustrate the diverse applications in which classification techniques can be found. Different domains have their own associated problems. In the first example, how do we handle the uneven cost of making errors? In the second, how do we handle complex data objects like text? We will address both of these issues in later sections.

5.2 Probabilistic classifiers

Probabilistic and non-probabilistic classifiers differ in the type of output they produce. In the probabilistic case, the output is the probability of a new object belonging to a particular class. Expressing the training data in matrix and vector form (\mathbf{X}, \mathbf{t}), this probability, for class c is:

$$P(T_{\text{new}} = c | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}). \quad (5.1)$$

As a probability it must satisfy the following constraints:

$$\begin{aligned} 0 &\leq P(T_{\text{new}} = c | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) \leq 1 \\ \sum_{c=1}^C P(T_{\text{new}} = c | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) &= 1. \end{aligned}$$

At first glance, obtaining a probability as an output may seem unnecessary. After all, we just said that our task was to predict the class T_{new} . If we are primarily interested in an assignment, then we might choose a non-probabilistic classifier. However, in many applications, the probability is useful, as it provides a level of *confidence* in the output. For example, consider a disease diagnosis application with two classes, healthy (0) and diseased (1). Providing the probability $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t})$ is much more informative than simply stating that $t_{\text{new}} = 1$. $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) = 0.6$ and $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) = 0.9$ both suggest that \mathbf{x}_{new} should be classified as diseased, but in the former case, the model is much less certain. Perhaps more tests are required before a decision can be made.

5.2.1 The Bayes classifier

Our first probabilistic classifier is known as the Bayes classifier, taking its name from the equation on which it is based. Given a set of training points from C classes, our aim is to be able to compute the predictive probabilities (Equation 5.1) for each of C potential classes. These probabilities can then form the basis of a decision making process (e.g. assign \mathbf{x}_{new} to the class with the highest probability) or be used to compute an expectation.

From Bayes' rule (see Section 2.2.7 and Chapters 3 and 4), we can obtain an expression for the predictive probability:

$$P(T_{\text{new}} = c | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) = \frac{p(\mathbf{x}_{\text{new}} | T_{\text{new}} = c, \mathbf{X}, \mathbf{t}) P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t})}{p(\mathbf{x}_{\text{new}} | \mathbf{X}, \mathbf{t})}.$$

The marginal likelihood, $p(\mathbf{x}_{\text{new}} | \mathbf{X}, \mathbf{t})$, can be expanded to a sum over the C possible classes, resulting in the following equation that defines the Bayes classifier:

$$P(T_{\text{new}} = c | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) = \frac{p(\mathbf{x}_{\text{new}} | T_{\text{new}} = c, \mathbf{X}, \mathbf{t}) P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t})}{\sum_{c'=1}^C p(\mathbf{x}_{\text{new}} | T_{\text{new}} = c', \mathbf{X}, \mathbf{t}) P(T_{\text{new}} = c' | \mathbf{X}, \mathbf{t})} \quad (5.2)$$

We are left with the task of defining $p(\mathbf{x}_{\text{new}} | T_{\text{new}} = c, \mathbf{X}, \mathbf{t})$ and $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t})$, the likelihood of \mathbf{x}_{new} belonging to the c th class and the prior probability of the c th class. We will now discuss each of these in turn.

5.2.1.1 Likelihood – class-conditional distributions

The likelihood term in Equation 5.2, $p(\mathbf{x}_{\text{new}} | T_{\text{new}} = c, \mathbf{X}, \mathbf{t})$, is a distribution specific to the c th class (it is conditioned on $T_{\text{new}} = c$), evaluated at \mathbf{x}_{new} . To create a Bayes classifier we need to define C of these class-conditional distributions. It is common to use the same type of distribution for each class, although there is no reason why this has to be the case. As with any choice of distribution, our decision should be based on the type of data being modelled and any additional knowledge we have about this data. Once we have chosen the distribution for the c th class, we are left with the job of choosing its parameters. For example, if we choose a Gaussian distribution, we need to choose the mean and (co)variance. Any parameters required to define the distribution for class c will be set using just the training data for class c . This stage in itself could be thought of as a Machine Learning problem and we will discuss it further in Section 5.2.1.3.

5.2.1.2 Prior class distribution

The second quantity in Equation 5.2 is $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t})$. This is the probability that the object belongs to class c conditioned on just the training data: \mathbf{X}, \mathbf{t} . It enables us to specify any prior beliefs in the class of \mathbf{x}_{new} before we see it. This allows us to account for uneven class sizes. For example, perhaps there is a class c that is extremely rare. Before we see the data we might like to bias against this class (choose $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t})$ to be very low) so that we will only classify \mathbf{x}_{new} as belonging to it if it has a very high likelihood. Alternatively, class c may be very rare but we always want to detect it – it may be crucial not to misclassify these rare instances. In this case, we might set $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t})$ to be high. This will result in more potential \mathbf{x}_{new} vectors being classified as class c . Of course, some of them will be incorrect (they truly belong to a different class) but we will not miss many that really belong to class c . These issues can also be resolved when we make a decision based on the set of predictive probabilities. We will discuss these issues at length in Section 5.4.

Regardless of our motives, the only technical restriction in the choice of $p(T_{\text{new}} = c | \mathbf{X}, \mathbf{t})$ is that they are positive and $\sum_c P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t}) = 1$. Two popular choices are:

1. Uniform prior: $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t}) = \frac{1}{C}$.

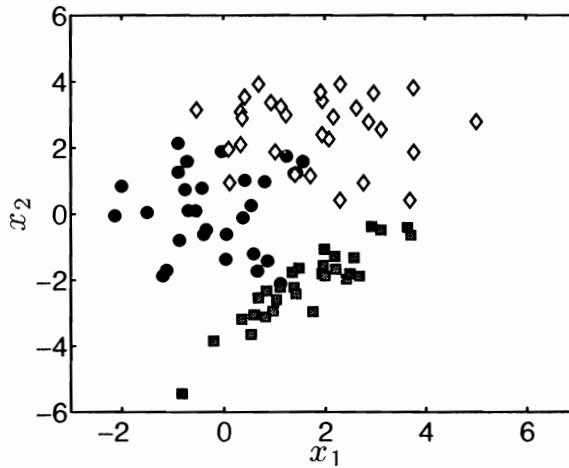


FIGURE 5.1: Three class classification dataset.

2. Class size prior: $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t}) = \frac{N_c}{N}$, where N is the number of objects in the training set and N_c is the number of objects in the training set belonging to class c .

Note that although we have written the prior as being conditioned on \mathbf{X} and \mathbf{t} it is not necessarily dependent on them. Neither example above uses \mathbf{X} in the prior definition and only the second example uses \mathbf{t} (through N_c).

5.2.1.3 Example – Gaussian class-conditionals

The data shown in Figure 5.1 has been generated from three classes. Each training object consists of a two-dimensional attribute vector $\mathbf{x}_n = [x_{n1}, x_{n2}]^\top$ and an associated label $t_n = \{1, 2, 3\}$. Class 1 is plotted as black circles, class 2 as white diamonds and class 3 as grey squares. Given that the attributes are real valued, we will use Gaussian class-conditional distributions:

$$p(\mathbf{x}_n | t_n = c, \mathbf{X}, \mathbf{t}) = \mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \quad (5.3)$$

where $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ need to be chosen based on the training points associated with class c . We will denote these points as \mathbf{X}^c . This is itself a Machine Learning task – we have some data (\mathbf{X}^c) and wish to infer something about the parameters of a model. In this example we will find the parameters $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ that maximise the likelihood of the observations \mathbf{X}^c . As an alternative, we could use a Bayesian approach. For example, defining a prior density for these parameters, $p(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$, we could compute a posterior from Bayes' rule:

$$p(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c | \mathbf{X}^c) = \frac{p(\mathbf{X}^c | \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)p(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{p(\mathbf{X}^c)}$$

and then compute the likelihood of \mathbf{x}_{new} by taking the following expectation:

$$p(\mathbf{x}_{\text{new}} | T_{\text{new}} = c, \mathbf{X}, \mathbf{t}) = \mathbf{E}_{p(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c | \mathbf{X}^c)} \{ p(\mathbf{x}_{\text{new}} | \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \} .$$

Assuming that the choice of prior $p(\mu_c, \Sigma_c)$ is conjugate with the Gaussian likelihood, the posterior and expectation could both be obtained analytically. Performing this analysis in a Bayesian manner is likely to be most useful when there is little data, and hence our estimates of μ_c and Σ_c are uncertain. See Exercises EX 5.1 and EX 5.2.

The maximum likelihood estimates for the mean and covariance of a Gaussian given a set of N data points can be obtained by differentiating the log likelihood with respect to each parameter, setting to zero and solving (just as we did in Chapter 2 for the linear model). Omitting the details (see Exercise EX 5.3), the maximum likelihood estimates are:

$$\mu_c = \frac{1}{N_c} \sum_{n=1}^{N_c} \mathbf{x}_n \quad (5.4)$$

$$\Sigma_c = \frac{1}{N_c} \sum_{n=1}^{N_c} (\mathbf{x}_n - \mu_c)(\mathbf{x}_n - \mu_c)^T, \quad (5.5)$$

where the summations are only over the data instances from the c th class. The three class-conditional distributions are shown (along with the data) in Figure 5.2 (MATLAB script: `plotcc.m`).

We are left with the task of deciding on the prior: $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t})$. As mentioned earlier, a common choice is $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t}) = \frac{N_c}{N}$, the proportion of training points in class c . In our example, there are $N_c = 30$ in each class and therefore $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t}) = \frac{1}{3}$.

5.2.1.4 Making predictions

Armed with the class-conditional distributions and the prior, we are able to make predictions. As a worked example, we will compute the posterior class probabilities for $\mathbf{x}_{\text{new}} = [2, 0]^T$. We summarise the various quantities that we need to compute

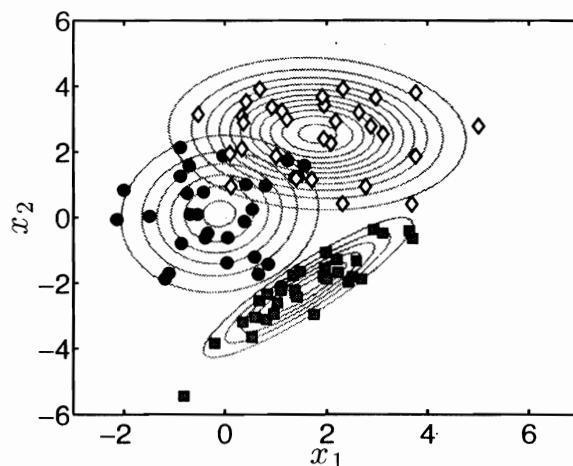


FIGURE 5.2: Three class classification dataset with the density contours for the three class-conditional distributions fitted using Equations 5.4 and 5.5.

TABLE 5.1: Likelihood and priors for $\mathbf{x}_{\text{new}} = [2, 0]^T$ for the Gaussian class-conditional Bayesian classification example.

c	$p(\mathbf{x}_{\text{new}} T_{\text{new}} = c, \mu_c, \Sigma_c)$	$P(T_{\text{new}} = c \mathbf{X}, \mathbf{t})$	$p(\mathbf{x}_{\text{new}} T_{\text{new}} = c, \mu_c, \Sigma_c)P(T_{\text{new}} = c \mathbf{X}, \mathbf{t})$
1	0.0138	$\frac{1}{3}$	0.0046
2	0.0061	$\frac{1}{3}$	0.0020
3	0.0002	$\frac{1}{3}$	0.0001

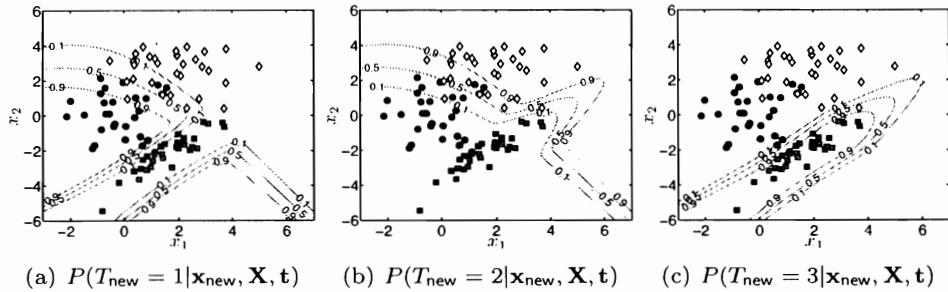


FIGURE 5.3: Contour plots of the classification probabilities for the Bayesian classifier with Gaussian class-conditional distributions.

and their values for \mathbf{x}_{new} in Table 5.1. The final column gives us the numerator of Equation 5.2. To convert these values into probabilities we have to divide each value by the sum of the three values: $0.0046 + 0.0020 + 0.0001 = 0.0067$. The resulting probabilities are:

$$\begin{aligned} P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) &= 0.6890 \\ P(T_{\text{new}} = 2 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) &= 0.3024 \\ P(T_{\text{new}} = 3 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) &= 0.0087, \end{aligned}$$

from which we can see that \mathbf{x}_{new} is approximately twice as likely to belong to class 1 (black circles) than class 2 (white diamonds) and is very unlikely to belong to class 3 (grey squares).

By evaluating the classification probabilities over a grid of many \mathbf{x}_{new} values, we can draw the contours of the classification probabilities. These can be seen in Figure 5.3 (MATLAB script: `bayesclass.m`). For each class the model assigns a high probability to the area of the space populated by training points of that class. However, there are some odd effects. Take 5.3(a) as an example: It has a high probability > 0.9 in the area around the middle left of the plot where most class 1 (black circles) data is located. However, it also has a high probability in the bottom right of the plot where there is no class 1 data (or data from any class). Similarly, the contours for class 2 (Figure 5.3(b)) give a high probability in the middle-right of the plot where there is no data belonging to class 2. These effects can be explained by noticing the steepness of the class-conditional contours for class 3 (Figure 5.2) when compared with those for classes 1 and 2. Its density decays so much faster than those for classes 1 and 2 that to the right of class 3, the density functions for both classes 1 and 2 are higher. This is an unfortunate property – it does not seem sensible to

label points to the bottom right of the plots as belonging with high probability to either class 1 or 2. It would be far better if, as in the binary response model we saw in Chapter 4, the probabilities would become less certain as we move away from the vicinity of the data.

5.2.1.5 The naive Bayes assumption

In the previous example, we used two-dimensional Gaussians for the class-conditional distributions. These distributions were able to capture dependencies between the two attribute variables for each class. For example, we can see how the class-conditional distribution for class 3 was able to capture the strong dependency that exists between x_1 and x_2 for the training data in class 3. Fitting a two-dimensional Gaussian involves choosing five parameter values: two for μ and three for Σ (Σ is symmetric so the two off-diagonal elements must be equal). This was perfectly feasible given that we had 30 training points in each class. Problems arise when the number of dimensions starts increasing. In general, fitting a D -dimensional Gaussian requires $D + D + \frac{D(D-1)}{2}$ parameters (D for the mean and $D + \frac{D(D-1)}{2}$ for the covariance matrix). For 10 dimensions, it is likely that 30 data points would not be sufficient to fit the resulting 65 parameters reliably.

A common way to partly overcome this problem (lack of data can only be completely solved by getting more data) is to make the *naive Bayes* assumption: the D -dimensional class-conditional distributions can be factorised into a product of D univariate distributions. In other words, conditioned on a particular class, the dimensions (e.g. x_1 and x_2) are independent. A univariate Gaussian requires two parameters – μ and σ^2 . Therefore, fitting D of these requires $2D$ parameters – 45 fewer in 10 dimensions than a single 10-dimensional Gaussian. The price we pay for this decrease in number of parameters is a decrease in model flexibility. In our Gaussian example, it means that we are restricting the shapes of the class-conditional distributions to be aligned with the axis – they cannot model any within-class dependencies. This is clear from Figure 5.4, where we see the density contours for the class-conditional distributions when we make the naive Bayes assumption:

$$p(\mathbf{x}_n | t_n = k, \mathbf{X}, \mathbf{t}) = \prod_{d=1}^2 p(x_{nd} | t_n = k, \mathbf{X}, \mathbf{t}).$$

Comparing this with Figure 5.2 it is clear that the model for class 3 no longer accurately reflects the characteristics of the data. Figure 5.5 shows the classification probability contours for the three classes (MATLAB script: `bayesclass.m`). It is interesting to note that although we know that the class-conditional distribution for class 3 is not particularly appropriate, the classification contours are still reasonable (notwithstanding the lack of uncertainty as we move away from the data).

5.2.1.6 Example – classifying text

Machine learning is widely used to perform automatic text classification. Learning from data makes a lot of sense within this domain – it is not straightforward to manually build a set of rules or models that could be used to classify text but at the same time there is a lot of data with which a classifier could be trained.

The 20 newsgroups dataset is a popular benchmark dataset on which to evaluate new algorithms. It consists of approximately 20,000 documents, each of which is

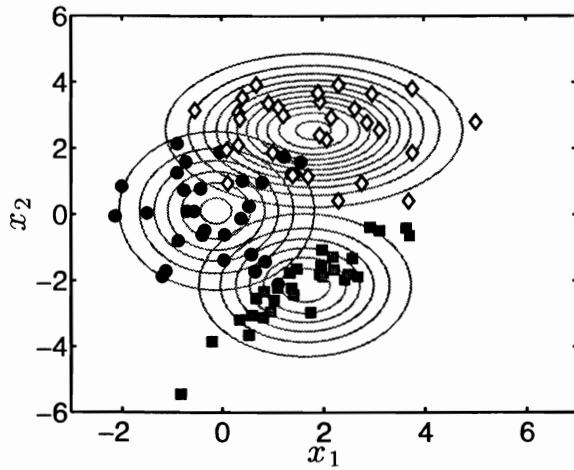


FIGURE 5.4: Density contours for Gaussian class-conditionals with the naive Bayes assumption.

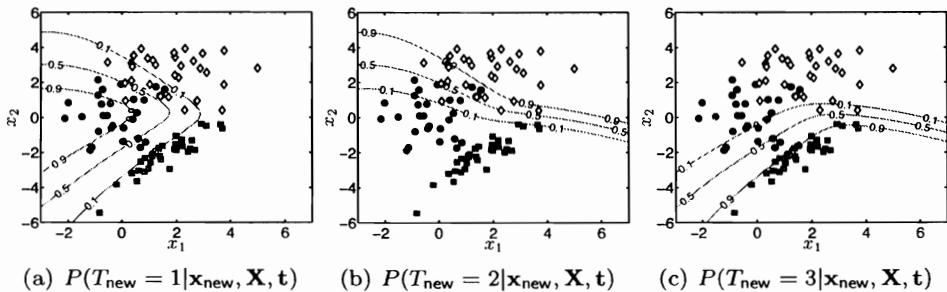


FIGURE 5.5: Contour plots of the classification probabilities for the Bayesian classifier with Gaussian class-conditional distributions and the naive Bayes assumption.

a post to one of 20 newsgroups. Considering each of these 20 newsgroups as a different class, we will build a classification system that can automatically assign a new document to one of these 20 classes. The groups cover a diverse set of topics including sport, computing and religion.

The algorithms that we have introduced work with numerical data and we therefore need a way of encoding a document as a vector of numerical values. The most common way of doing this is to use the *bag-of-words* model. If the total number of unique words in all documents (the vocabulary) is M ; each document is represented as an M -dimensional vector. The vector for the n th document, \mathbf{x}_n , is made up of the counts of the number of times each word appears. x_{nm} is therefore the number of times word m appears in document n .

Given that the vocabulary is likely to be large, we will make the naive-Bayes assumption. Therefore our class-conditional distribution can be decomposed into a

product over the words in the vocabulary:

$$p(\mathbf{x}_n | T_n = c, \dots) = \prod_{m=1}^M p(x_{nm} | T_n = c, \dots).$$

This assumption means that the number of parameters we require to define each class-conditional will be roughly equal (depending on the choice of distribution function) to the number of words. Adding any form of dependency between words would cause an explosion in the number of parameters that we would have to fit. For example, if we looked at pair-wise dependencies, we would need on the order of M^2 parameters. Given that a typical vocabulary might include some 50,000 words, this is already a significant challenge.

The bag-of-words model also assumes that the ordering of the words is not important. For example, \mathbf{x}_n would be identical for the following two sentences despite the fact that the second is nonsense:

- 1: The quick brown fox jumps over the lazy dog.
- 2: Dog quick lazy the jumps fox brown the over.

This assumption is not too restrictive: if our classifier is given a document that includes many instances of the word ‘baseball’ it is likely that this document is about sport regardless of the particular ordering of the words. Note that the bag-of-words model ignores ordering but does not necessarily imply independence. We could still define class-conditional distributions that allowed for dependencies between the elements of \mathbf{x}_n .

We will use multinomials (introduced in Section 2.3.3) for the class-conditional distributions. The multinomial distribution for the vector \mathbf{x}_n is defined as:

$$P(\mathbf{x}_n | \mathbf{q}) = \left(\frac{s_n!}{\prod_{m=1}^M x_{nm}!} \right) \prod_{m=1}^M q_m^{x_{nm}} \quad (5.6)$$

where $s_n = \sum_{m=1}^M x_{nm}$ and $\mathbf{q} = [q_1, \dots, q_M]^\top$ are a set of parameters, each of which is a probability ($\sum_m q_m = 1$). Note that the multinomial distribution automatically makes the naive Bayes assumption through the product over m .

There will be one multinomial (and hence one \mathbf{q}) for each class. Therefore, we need to determine the value of \mathbf{q}_c (the vector of probabilities for the c th class) based on the set of training objects – \mathbf{x}_n – corresponding to class c . We can do this with maximum likelihood (see Exercise EX 5.4), resulting in:

$$q_{cm} = \frac{\sum_{n=1}^{N_c} x_{nm}}{\sum_{m'=1}^M \sum_{n=1}^{N_c} x_{nm'}}$$

where the summations over n just include objects from class C . Defining the prior distribution $P(T_{\text{new}} = c | \mathbf{X}, \mathbf{t}) = \frac{1}{C}$, we could make predictions using Equation 5.2. However, before we do that there is a problem that needs to be addressed.

5.2.1.7 Smoothing

It is quite feasible that a particular word (say m) will never appear in documents from one class (say c) – not many religious newsgroup posts are likely to mention

'baseball'. This will result in $q_{cm} = 0$. Look back at Equation 5.6 – if any one or more q_{cm} is zero with nonzero x_{nm} , the product $\prod_{m=1}^M q_{cm}^{x_{nm}}$ will be zero. In other words, if we are trying to compute the classification probability for a new document \mathbf{x}_{new} that happens to include word m , the likelihood ($p(\mathbf{x}_{\text{new}}|T_{\text{new}} = c, \mathbf{q}_c)$) will equal zero and hence $P(T_{\text{new}} = c|\mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t}) = 0$. A document including a word that doesn't appear in any of the training documents will have probability 0 of belonging to all classes. This is another example of over-fitting to the training data and we can overcome this by placing a prior density on \mathbf{q} that encodes the belief that all probabilities are greater than 0. Once we have defined this prior, we can set \mathbf{q} with the MAP estimate (see Chapter 4) rather than the maximum likelihood estimate. We could also take expectations with respect to the posterior density of \mathbf{q} (see Exercises EX 5.5 and EX 5.6).

A suitable prior density for a vector of probabilities is the Dirichlet density, defined as:

$$p(\mathbf{q}_c|\boldsymbol{\alpha}) = \frac{\Gamma\left(\sum_{m=1}^M \alpha_m\right)}{\prod_{m=1}^M \Gamma(\alpha_m)} \prod_{m=1}^M q_{cm}^{\alpha_m - 1} \quad (5.7)$$

We will simplify this further by assuming that $\alpha_m = \alpha$ – i.e. the parameter used to define the Dirichlet is the same for each word. The MAP estimate can be obtained by maximising the prior multiplied by the multinomial likelihood (or the log of this product). Omitting the details here (see Exercise EX 5.7), the MAP estimate for q_{cm} is:

$$q_{cm} = \frac{\alpha - 1 + \sum_{n=1}^{N_c} x_{nm}}{M(\alpha - 1) + \sum_{m'=1}^M \sum_{n=1}^{N_c} x_{nm'}} \quad (5.8)$$

Once again, summations are only over the training objects from class c . For $\alpha > 1$, $q_{cm} > 0$ and the issue of zeros is no longer a problem. This technique is often referred to as *smoothing* – if we keep increasing α , each word probability q_{cm} will get closer and closer to $\frac{1}{M}$. This could also be considered as another example of regularisation (see Section 1.6).

The newsgroup data has been split into training and test sets holding $\approx 11,000$ and $\approx 7,000$ documents, respectively. Setting $\alpha = 2$, using Equation 5.8 to determine \mathbf{q}_c and setting the prior classification probability to $1/20$ (a uniform prior over the 20 classes) we can compute the classification probabilities using Equation 5.2 where $p(\mathbf{x}_{\text{new}}|t_{\text{new}} = c, \mathbf{X}, \mathbf{t})$ is given by Equation 5.6 with \mathbf{x}_n substituted for \mathbf{x}_{new} .

For each of the $\approx 7,000$ \mathbf{x}_{new} vectors, we have a set of 20 probabilities. The simplest way to evaluate how well the classifier is working is to assign each \mathbf{x}_{new} to the class for which it has the highest probability and compare these assignments to the known labels. If we do this, we find that the classifier is correct 78% of the time – not bad given that we have used the simplest possible model and have not attempted to optimise it in any way.

Figure 5.6 provides a graphical representation of the classification probabilities for the $\approx 7,000$ test points (MATLAB script: `newspred.m`). Each row corresponds to a single test point and the rows are ordered by true class. Each column corresponds to a predicted class. For example, the values in column 10 give the probabilities of test points being classified as belonging to class 10. The block-like structure present tells us that the algorithm is doing reasonably well – the probabilities are high where they ought to be. A plot like this also allows us to see whether there is any pattern

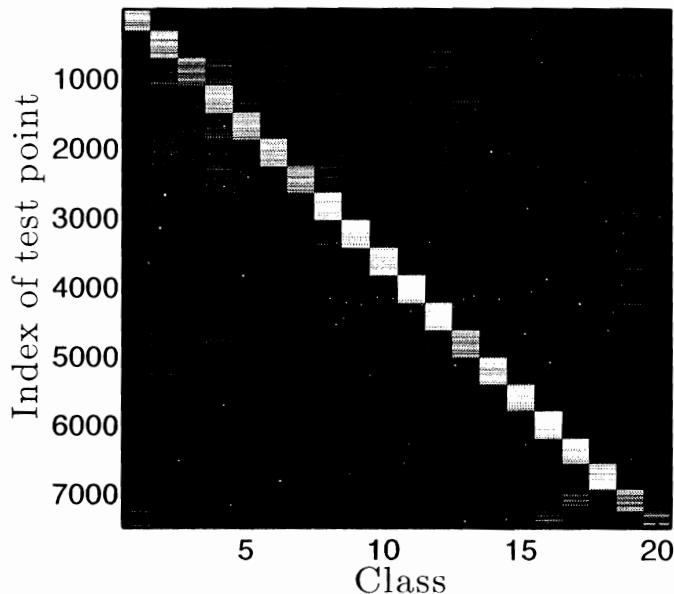


FIGURE 5.6: Graphical representation of the predictive probabilities for the Bayesian classifier on the 20 newsgroups data. Each row corresponds to one test point and the test points are ordered by true class. The whiter the colour, the higher the probability.

in the errors being made. For example, it seems that a large number of test points belonging to class 19 (the penultimate block) are wrongly classified as belonging to class 17. These two classes are from the newsgroups `talk.politics.guns` and `talk.politics.misc` and so it is not surprising that there might be some confusion here – many popular words will be shared by the two classes. Another example is the confusion between classes 20 and 16 for data points whose true class is 20. These two are from `talk.religion.misc` and `soc.religion.christian`, which are also clearly related. An analysis of the types of mistakes being made by classification algorithms will often enable us to improve performance. In this example, it may be sensible to consider whether classes 16 and 20 should be amalgamated into a larger class. If not, it might suggest that we should attempt to obtain more data (documents) from these two classes.

There are several ways to analyse the results being produced by a classification algorithm. We will look at these in more detail later in this chapter (Section 5.4). In the meantime, we will move on to our second probabilistic classifier.

5.2.2 Logistic regression

Although we called it a binary response model, Chapter 4 was entirely devoted to a binary classifier known as logistic regression. In Chapter 4 we didn't really discuss it from the viewpoint of classification but as a model for which analytical Bayesian inference was not possible. However, everything that one would need to

use this method is there in Chapter 4 and we will not reproduce it here. There are, however, a couple of points that are worth discussing – the motivation for what we called the ‘squashing function’ and generalisations to this type of model.

5.2.2.1 Motivation

In Chapter 4 we motivated the logistic likelihood,

$$P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x}_{\text{new}})},$$

by arguing that we wanted to use our familiar linear model ($\mathbf{w}^T \mathbf{x}$) but needed to transform it so that the output was a probability ($0 \leq P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}) \leq 1$).

Whilst there is nothing wrong with this as a motivation, the use of the logistic likelihood is usually more formally derived as a result of modelling the *log-odds ratio*. This is the log of the ratio between $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})$ and $P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w})$:

$$\log \left(\frac{P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})}{P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w})} \right)$$

There are no constraints on this value – it can take any real value. If $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}) \ll P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w})$ the log ratio will take on a large negative value and if $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}) \gg P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w})$ it will take a large positive one. Therefore, this quantity is a sensible candidate for modelling with our familiar linear model:

$$\log \left(\frac{P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})}{P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w})} \right) = \mathbf{w}^T \mathbf{x}_{\text{new}} \quad (5.9)$$

With a bit of re-arranging, and noting that

$$P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w}) = 1 - P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}),$$

we can obtain an expression for $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})$:

$$\begin{aligned} \log \left(\frac{P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})}{P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w})} \right) &= \mathbf{w}^T \mathbf{x}_{\text{new}} \\ \frac{P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})}{P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w})} &= \exp(\mathbf{w}^T \mathbf{x}_{\text{new}}) \\ \frac{P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})}{1 - P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})} &= \exp(\mathbf{w}^T \mathbf{x}_{\text{new}}) \\ P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})(1 + \exp(\mathbf{w}^T \mathbf{x}_{\text{new}})) &= \exp(\mathbf{w}^T \mathbf{x}_{\text{new}}) \\ P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}) &= \frac{\exp(\mathbf{w}^T \mathbf{x}_{\text{new}})}{1 + \exp(\mathbf{w}^T \mathbf{x}_{\text{new}})} \\ P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w}) &= \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{x}_{\text{new}})} \end{aligned}$$

By using the logistic likelihood for $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})$, we are actually modelling the log-odds ratio with a linear model. In the statistics community approaches like this are known as *generalised linear models* – linear models that are passed through some transformation to model the quantity of interest.

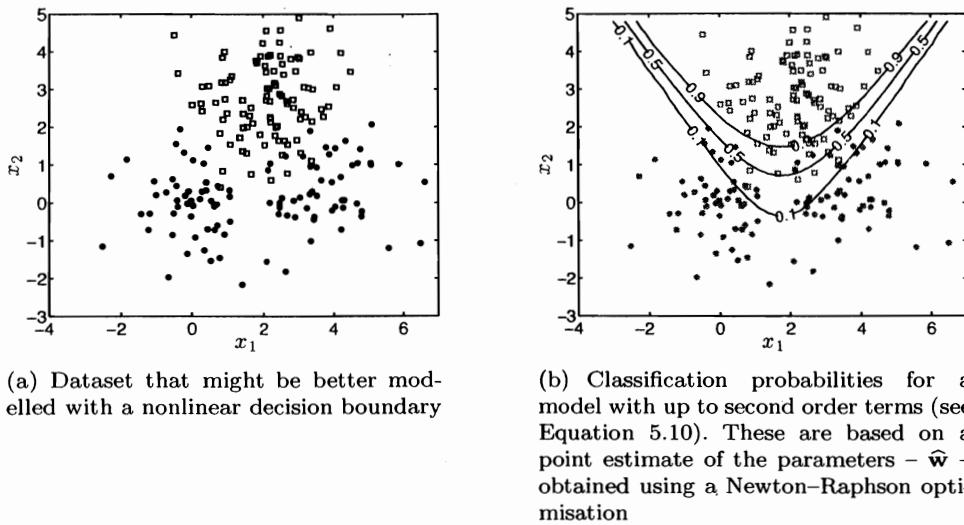


FIGURE 5.7: Binary data and classification probability contours for the logistic regression model described by Equation 5.10.

5.2.2.2 Nonlinear decision functions

The decision boundaries for individual \mathbf{w} values in Chapter 4 were all straight lines. The probability contours obtained with the Laplace approximation and the Metropolis-Hastings algorithm were curved as the result of averaging over many straight lines. By expanding \mathbf{x}_n to include terms like x_n^2 , we can obtain non-linear decision boundaries in logistic regression in a manner similar to the way we obtained nonlinear regression functions in Chapter 1. For example, the data in Figure 5.7(a) shows a binary classification dataset that may require a nonlinear decision boundary.

Using x_1 and x_2 to denote the individual attributes ($\mathbf{x} = [x_1, x_2]^\top$), we could use the following model for the log-odds ratio:

$$\log \left(\frac{P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{w})}{P(T_{\text{new}} = 0 | \mathbf{x}_{\text{new}}, \mathbf{w})} \right) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 \quad (5.10)$$

To show that this is able to produce nonlinear decision boundaries, we find the MAP estimate $\hat{\mathbf{w}}$ of the parameters, assuming a Gaussian prior $p(\mathbf{w} | \sigma^2) = \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ (see Section 4.3). We could, of course, submit this model to the more Bayesian treatments introduced in Chapter 4 if desired.

Plugging $\hat{\mathbf{w}}$ into the logistic likelihood allows us to compute the classification probabilities:

$$P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \hat{\mathbf{w}}) = \frac{1}{1 + \exp(-\hat{\mathbf{w}}^\top \mathbf{x}_{\text{new}})}$$

Evaluating this over a grid of \mathbf{x}_{new} values enables us to plot contours of the probability of belonging to class 1 over the space of attributes. The nonlinear decision boundary can be seen in Figure 5.7(b) (the training points have been greyed out a little to make the contours easier to see) (MATLAB script: `nonlinlogreg.m`). This is an appealing property of logistic regression but it is important to remember that

the problems of over-fitting and poor generalisation we saw when making the linear model more and more complex in Chapter 1 are just as troublesome in the classification domain. Remember that this is for a single value of \mathbf{w} : we obtained nonlinear decision boundaries in the Laplace and Metropolis-Hastings cases in Chapter 4, but only by averaging over many different straight lines.

5.2.2.3 Nonparametric models - the Gaussian process

Throughout this book, we have restricted ourselves to models of the form $\mathbf{w}^\top \mathbf{x}$. This model has a set of parameters \mathbf{w} in which it is linear. For any particular expansion of \mathbf{x} (e.g. adding squared terms), this function belongs to a particular family of functions. For example, if we had squared terms, it is a member of the quadratic family. The choice of a family places restrictions on the flexibility of the function – if we choose $w_0 + w_1x$, we can only model things with straight lines. If we choose $w_0 + w_1x + w_2x^2 + w_3x^3$, we can only model things with cubic (3rd order) polynomials. Models such as these are known as *parametric*, as they belong to a particular parametric family and the particular function within that family is determined by a set of parameters \mathbf{w} .

It is worth briefly mentioning a very flexible alternative – nonparametric models. Rather than being defined as a function of some parameters (e.g. $f(\mathbf{x}; \mathbf{w})$), nonparametric models are defined in a more general manner. For example, a popular nonparametric model is the Gaussian process (GP). In parametric models, we place a prior distribution on the parameters \mathbf{w} which in turn implies a prior distribution over the output values of the function. With a GP we place the prior distribution *directly* onto the output values of the function. Note that nonparametric does not mean that the GP does not have *any* parameters, but that it does not assume a parametric form for the function.

The GP is characterised by two functions – a mean function $\mu(\mathbf{x})$ that describes the average function value as a function of the attribute x (x can be a scalar or a vector) and a covariance function $c(\mathbf{x}_n, \mathbf{x}_m)$ that defines how similar the function output at \mathbf{x}_n should be to that at \mathbf{x}_m . In practice, the mean function is often assumed to be 0.

For any finite set of N data points, the Gaussian process essentially becomes an N -dimensional Gaussian distribution with mean $\boldsymbol{\mu} = [\mu(\mathbf{x}_1), \dots, \mu(\mathbf{x}_N)]^\top$ and covariance matrix:

$$\mathbf{C} = \begin{bmatrix} c(\mathbf{x}_1, \mathbf{x}_1) & c(\mathbf{x}_1, \mathbf{x}_2) & \dots & c(\mathbf{x}_1, \mathbf{x}_N) \\ c(\mathbf{x}_2, \mathbf{x}_1) & c(\mathbf{x}_2, \mathbf{x}_2) & \dots & c(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ c(\mathbf{x}_N, \mathbf{x}_1) & c(\mathbf{x}_N, \mathbf{x}_2) & \dots & c(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}$$

Sampling a vector from this Gaussian gives a value for the output of the function at each of the N data objects.

GPs have become increasingly popular within Machine Learning due to their flexibility – a GP is not restricted to a particular parametric family. They can be thought of as a replacement for a parametric model in any particular algorithm. For example, we could have used a GP to model the Olympics data, or as a replacement for $\mathbf{w}^\top \mathbf{x}_n$ in the logistic regression algorithm. We will omit further discussion here, but the reader is encouraged to investigate the use of GPs in the Machine Learning literature.

5.3 Nonprobabilistic classifiers

We now turn our attention to nonprobabilistic classifiers. Rather than providing a probability of class membership, $P(T_{\text{new}} = c | \mathbf{x}_{\text{new}}, \mathbf{x}, t)$, their output is an assignment of an object to a class: $t_{\text{new}} = c$. We will look at two different algorithms – K -nearest neighbours and the Support Vector Machine. Both are very popular within Machine Learning due to their excellent empirical performance. The Support Vector Machine will also provide us with an introduction to the area of kernel methods.

5.3.1 K -nearest neighbours

Our first approach, K -nearest neighbours (KNN), is very popular due to simplicity and excellent empirical performance. It can handle both binary and multi-class data and makes no assumptions about the parametric form of the decision boundary. KNN does not have a training phase and is best described through the simple process used to classify new objects, \mathbf{x}_{new} .

Consider our normal scenario – we have N training objects, each of which is represented by a set of attributes \mathbf{x}_n and a label t_n . To classify \mathbf{x}_{new} with KNN, we first find the K training points that are closest to \mathbf{x}_{new} . t_{new} is then set to be the majority class amongst these neighbours. This is illustrated in Figure 5.8. The training data consists of data points belonging to one of two classes (grey circles and white squares). Two test points are indicated by black diamonds and in both cases, the dotted circles enclose their $K = 3$ nearest neighbours. The neighbours of test

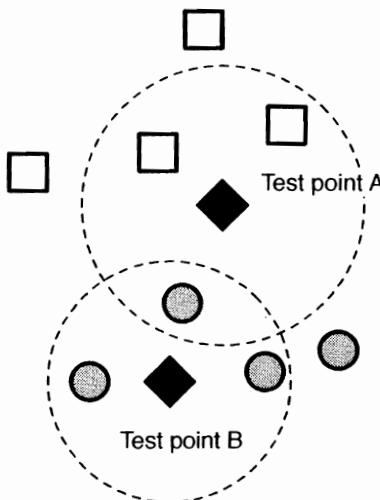


FIGURE 5.8: Cartoon depicting the operation of KNN ($K = 3$). Circles and squares denote the training points and diamonds the test points. Test point A will be assigned to the ‘square’ class and B to the circles.

point A include two from the square class and one from the circle class and so it will be classified as belonging to the square class. All of the neighbours of test point B belong to the circle class to which B is therefore assigned.

One drawback of the K -nearest-neighbours (KNN) approach is the issue of ties – two or more classes having an equal number of votes. For example, if $K = 8$ in Figure 5.8, we will always have four neighbours belonging to each class and no majority. One option is to assign the class randomly from the set of tied classes. This may not always be sensible as it means that the same \mathbf{x}_{new} may be assigned to different classes if it is tested more than once. For binary classification, a neater solution is to always use an odd number of neighbours. More generally we can weight the votes according to distance such that the votes from closer points have greater influence, making ties highly unlikely.

In Figure 5.8 we have used Euclidean distance to determine which points are neighbours of the test point. However, we are free to choose any distance measure we like. KNN is therefore very flexible – it can be used for any data type for which we can define a distance between the objects. Examples of other data types for which KNN has been used successfully include strings, graphs and images.

5.3.1.1 Choosing K

Once we have some data and have chosen a suitable distance measure, the only thing that remains is the choice of K . If K is too small, our classification can be heavily influenced by noise. This is demonstrated in Figure 5.9(a) (MATLAB script: `knnexample.m`) where we have plotted the decision boundary (consisting of points equidistant from a single neighbour in each class) for some binary data with $K = 1$. Whilst the majority of the boundary looks reasonable, there are three ‘islands’ that look like the result of over-fitting. Each island guards a large area of input space on what looks like the wrong side of the decision boundary. The three points in the centre of the islands are likely to be noise (i.e. mislabeled points). This problem is easily rectified by increasing K . Figure 5.9(b) shows the same data along with the $K = 5$ decision boundary. The inclusion of more neighbours has had the effect of regularising the boundary, removing the three islands.

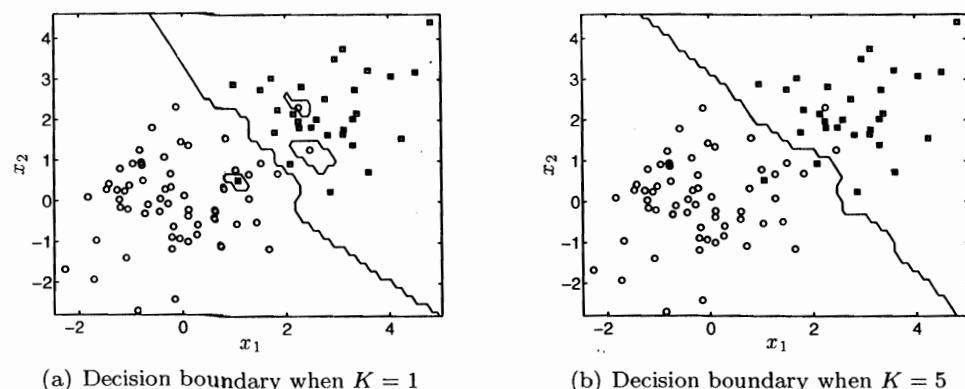


FIGURE 5.9: Binary classification dataset and decision boundaries for $K = 1$ and $K = 5$.

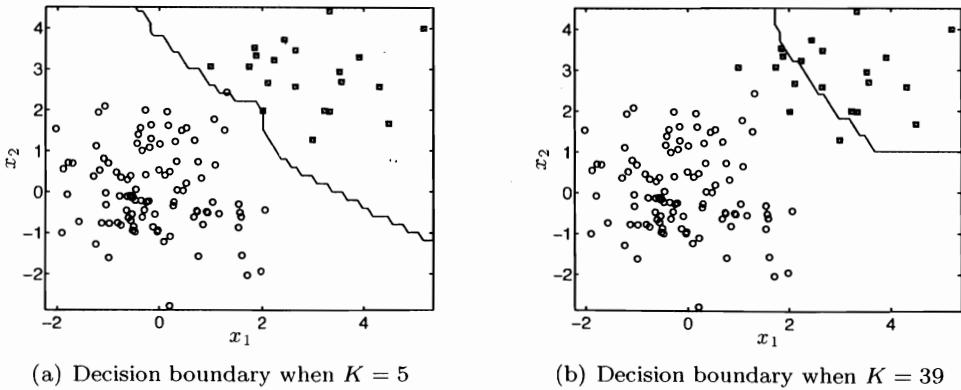
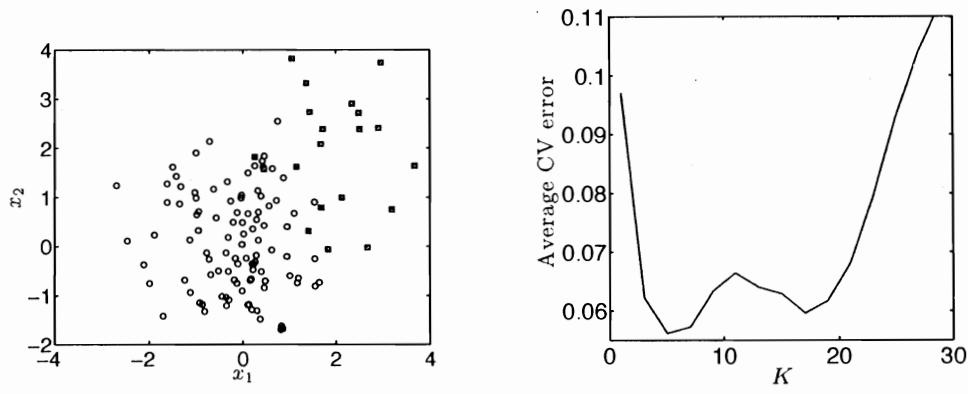


FIGURE 5.10: Second binary classification dataset and decision boundaries for $K = 5$ and $K = 39$.

We have seen that a very small value of K can be dangerous. What happens if K gets too big? As we increase K , we are using neighbours from further away from \mathbf{x}_{new} . Up to a point, this is useful. It has a regularising effect that reduces the chance of over-fitting. However, if we go too far, we will lose the true patterns in the data that we are attempting to model. Consider this extreme example: in some hypothetical training data there are $N_0 = 50$ and $N_1 = 10$ points from classes 0 and 1, respectively. Given that $N_1 = 10$, no test point can have more than 10 neighbours belonging to class 1. Therefore, if $K \geq 21$, \mathbf{x}_{new} can never be classified as belonging to class 1 – we have smoothed to such an extent that everywhere belongs to class 0! A less extreme example can be seen in Figures 5.10(a) and 5.10(b), where we show a dataset that has 50 points in class 0 (white circles) and only 20 points in class 1 (grey squares). The $K = 5$ decision boundary in Figure 5.10(a) looks reasonable whilst the $K = 39$ boundary in Figure 5.10(b) is being pushed up into the top right corner as the larger class exerts its influence.

Datasets with uneven numbers of objects in each class are known as *imbalanced* and are common in Machine Learning and something we must be aware of when we undertake any classification analysis. We will discuss this particular problem in more detail in Section 5.4.

The most popular method for choosing K is cross-validation (see Section 1.5.2). In previous sections when t was continuous, we used cross-validation to optimise the squared loss. We now need something suitable for our discrete (classification) t . We will discuss various other measures in Section 5.4 but for the moment we will use the simple measure that we used with the newsgroups data in Section 5.2.1.6 – the proportion of times the classifier makes a mistake. Figure 5.11(b) shows how the percentage error (0/1 loss) changes as K increases for the synthetic data given in Figure 5.11(a) (MATLAB script: `knncv.m`). Ten-fold cross-validation was used and, to remove the effect of any particular partitioning of the data into the 10 folds, the entire process was repeated 100 times. The errors plotted are therefore an average of $10 \times 100 = 1000$ fold errors. As K increases, the classification error drops to a minimum corresponding to $K = 5$ and then starts increasing (with a little bump at $K = 17$).



(a) Binary classification dataset. Note the class imbalance: the grey squares class has fewer members than the white circles

(b) Average cross-validation error as K is increased

FIGURE 5.11: Using cross-validation to find the best value of K . Ten-fold cross-validation was used and the reported error is averaged both over the folds and over 100 different partitions of the data into folds.

5.3.2 Support vector machines and other kernel methods

Our second nonprobabilistic classifier is the Support Vector Machine (SVM). SVMs are binary classifiers (although multi-class extensions have been proposed) and have been used successfully across a wide range of Machine Learning applications. Their success is due to their excellent empirical performance and for many applications they are hard to beat. They have been found to be particularly useful in applications where the number of attributes is much larger than the number of training objects. This is because the number of parameters that must be set for the SVM is related to the number of training objects and not the number of attributes.

The standard SVM uses a linear decision boundary, given by $\mathbf{w}^T \mathbf{x}_{\text{new}} + b$, to classify new data objects. Objects lying on one side of the line are put into class $t_{\text{new}} = 1$ and objects on the other side into $t_{\text{new}} = -1$ (note that the class labels are $\{1, -1\}$ rather than $\{0, 1\}$).

The SVM decision function for a test point \mathbf{x}_{new} is therefore given as:

$$t_{\text{new}} = \text{sign}(\mathbf{w}^T \mathbf{x}_{\text{new}} + b) \quad (5.11)$$

The learning task involves choosing the values of \mathbf{w} and b based on the training data. This is achieved by finding the parameters that maximise a quantity called the *margin*. This is much the same way that we minimised the loss in Chapter 1, maximised the likelihood in Chapter 2 and found the MAP solution in Chapter 3.

5.3.2.1 The margin

The margin is defined as the perpendicular distance from the decision boundary to the closest points on either side. This is illustrated in Figure 5.12, where the margin is denoted by γ .

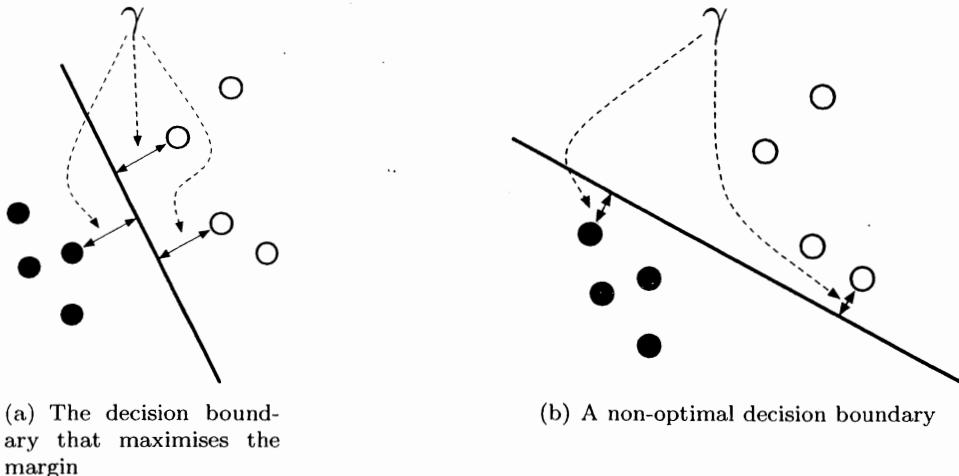


FIGURE 5.12: The classification margin γ , defined as the perpendicular distance from the decision boundary to the closest points on either side.

The toy examples in Figures 5.12(a) and 5.12(b) demonstrate why the margin is a sensible quantity to maximise. Intuitively, the boundary corresponding to the larger margin (Figure 5.12(a)) looks more sensible. In particular, the decision boundary in Figure 5.12(b) will classify points towards the top left and bottom right as belonging to the white and black classes, respectively, whereas common sense would suggest the opposite. Notice how in Figure 5.12(b) the margin is computed as the distance between the decision boundary and a *different* set of training points than in Figure 5.12(a). The margin is defined as the distance between the boundary and the *closest* points, the set of which will change as the boundary does.

5.3.2.2 Maximising the margin

It is easiest to compute the margin using one point from each class. Figure 5.13 shows how this is done. \mathbf{x}_1 and \mathbf{x}_2 are the closest points from the two classes. 2γ (i.e. double the boundary) is equal to the component of the vector joining \mathbf{x}_1 and \mathbf{x}_2 in the direction perpendicular to the boundary.

The vector joining \mathbf{x}_1 and \mathbf{x}_2 is given by $\mathbf{x}_1 - \mathbf{x}_2$ and the direction perpendicular to the decision boundary is given by $\mathbf{w}/\|\mathbf{w}\|$. The inner product between these two quantities gives us the quantity that we require:

$$2\gamma = \frac{1}{\|\mathbf{w}\|} \mathbf{w}^\top (\mathbf{x}_1 - \mathbf{x}_2).$$

Our decision function, $t_{\text{new}} = \text{sign}(\mathbf{w}^\top \mathbf{x}_{\text{new}} + b)$, is invariant to scaling its argument by a positive constant. This means that we can multiply $\mathbf{w}^\top \mathbf{x}_{\text{new}} + b$ by a positive constant λ and the output of the sign function will be unchanged. Therefore, we can decide to fix the scaling of \mathbf{w} and b such that $\mathbf{w}^\top \mathbf{x} + b = \pm 1$ for the closest points

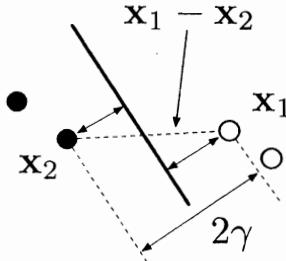


FIGURE 5.13: Illustrating the steps taken to compute the margin, γ . 2γ is equal to the component of the vector $\mathbf{x}_1 - \mathbf{x}_2$ in the direction perpendicular to the boundary.

on either side. This restriction allows us to simplify our expression for γ :

$$\begin{aligned}
 2\gamma &= \frac{1}{\|\mathbf{w}\|} \mathbf{w}^\top (\mathbf{x}_1 - \mathbf{x}_2) \\
 &= \frac{1}{\|\mathbf{w}\|} (\mathbf{w}^\top \mathbf{x}_1 - \mathbf{w}^\top \mathbf{x}_2) \\
 &= \frac{1}{\|\mathbf{w}\|} (\mathbf{w}^\top \mathbf{x}_1 + b - \mathbf{w}^\top \mathbf{x}_2 - b) \\
 &= \frac{1}{\|\mathbf{w}\|} (1 + 1) \\
 \gamma &= \frac{1}{\|\mathbf{w}\|}. \tag{5.12}
 \end{aligned}$$

Comment 5.1 – Constrained optimisation with Lagrange multipliers: At various points in this book we will need to perform constrained optimisations – finding the values of a set of parameters that maximise (or minimise) an objective function but that also satisfy some constraints. This can be done using *Lagrange* multipliers. In particular, we make a new objective function which includes the original plus an additional term for each constraint. The form of these terms is chosen such that the optimum of the new function is equal to the optimum of the constrained problem.

For example, suppose we wish to minimise $f(w)$ subject to the constraint $g(w) \leq a$:

$$\begin{aligned}
 &\operatorname{argmin}_w f(w) \\
 &\text{subject to } g(w) \leq a
 \end{aligned}$$

The new objective function is produced by adding a Lagrangian term of the form $\lambda(a - g(w))$ and optimised over both w and the Lagrange multiplier λ :

$$\begin{aligned}
 &\operatorname{argmin}_{w,\lambda} f(w) - \lambda(g(w) - a) \\
 &\text{subject to } \lambda > 0.
 \end{aligned}$$

Comment 5.1 - Constrained optimisation with Lagrange multipliers (continued): We are not going to go into the details of how this works here. Whenever we perform constrained optimisation, we will state the necessary Lagrangian terms without any further details. For more details, see the suggested reading at the end of this chapter.

To maximise the margin we must therefore maximise $\frac{1}{\|\mathbf{w}\|}$. There are, however, some constraints. Recall that we decided that, for the closest points in class 1, $\mathbf{w}^\top \mathbf{x}_n + b = 1$. Therefore, \mathbf{w} has to be chosen to satisfy $\mathbf{w}^\top \mathbf{x}_n + b \geq 1$ for all points in class 1. Similarly, it must satisfy $\mathbf{w}^\top \mathbf{x}_n + b \leq -1$ for those in class -1. Defining the labels as ± 1 allows us to express these two sets of constraints succinctly as:

$$t_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1.$$

Therefore, our learning task is to find the largest value of $\gamma = \frac{1}{\|\mathbf{w}\|}$ that satisfies these N constraints (where N is the number of points in the training set). It will actually be easier to minimise $\frac{1}{2}\|\mathbf{w}\|^2$ and so we shall do this instead. Formally, our optimisation problem has become:

$$\begin{aligned} & \underset{\mathbf{w}}{\operatorname{argmin}} \quad \frac{1}{2}\|\mathbf{w}\|^2 \\ & \text{subject to } t_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1, \text{ for all } n. \end{aligned}$$

This is the first time we have come across a constrained optimisation problem. To solve it, we need to incorporate the constraints into the objective function through a set of Lagrange multipliers. Lagrange multipliers add a new term in our objective function for each constraint such that the optimum of the new objective function corresponds to the optimum of the original, constrained problem. In our case, we need N Lagrangian terms. Each has an associated Lagrange multiplier, which is itself constrained to be positive. Without going into any more details of Lagrange multipliers, our new objective function is:

$$\begin{aligned} & \underset{\mathbf{w}, \alpha}{\operatorname{argmin}} \quad \frac{1}{2}\mathbf{w}^\top \mathbf{w} - \sum_{n=1}^N \alpha_n(t_n(\mathbf{w}^\top \mathbf{x}_n + b) - 1) \\ & \text{subject to } \alpha_n \geq 0, \text{ for all } n, \end{aligned}$$

where we've used the fact that $\|\mathbf{w}\|^2 = \mathbf{w}^\top \mathbf{w}$. At an optimum of this new objective function, the partial derivatives of the objective function with respect to \mathbf{w} and b must be zero. These derivatives are:

$$\begin{aligned} \frac{\partial}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{n=1}^N \alpha_n t_n \mathbf{x}_n \\ \frac{\partial}{\partial b} &= - \sum_{n=1}^N \alpha_n t_n. \end{aligned}$$

Equating these two expressions to zero gives us the following two identities that

must be satisfied at the optimum:

$$\mathbf{w} = \sum_{n=1}^N \alpha_n t_n \mathbf{x}_n \quad (5.13)$$

$$\sum_{n=1}^N \alpha_n t_n = 0. \quad (5.14)$$

Substituting the first of these identities back into the objective function gives us a new objective function which must be *maximised* with respect to the α_n rather than \mathbf{w} :

$$\begin{aligned} & \frac{1}{2} \mathbf{w}^\top \mathbf{w} - \sum_{n=1}^N \alpha_n (t_n (\mathbf{w}^\top \mathbf{x}_n + b) - 1) \\ &= \frac{1}{2} \left(\sum_{m=1}^N \alpha_m t_m \mathbf{x}_m^\top \right) \left(\sum_{n=1}^N \alpha_n t_n \mathbf{x}_n \right) - \sum_{n=1}^N \alpha_n \left(t_n \left(\sum_{m=1}^N \alpha_m t_m \mathbf{x}_m^\top \mathbf{x}_n + b \right) - 1 \right) \\ &= \frac{1}{2} \sum_{n,m=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^\top \mathbf{x}_n - \sum_{n,m=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^\top \mathbf{x}_n - \sum_{n=1}^N \alpha_n t_n b + \sum_{n=1}^N \alpha_n \\ &= \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n,m=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^\top \mathbf{x}_n \end{aligned}$$

where we used the fact that $\sum_{n=1}^N \alpha_n t_n = 0$ to remove the third term in the penultimate line. This expression is known as the dual optimisation problem and has to be maximised subject to the following constraints:

$$\alpha_n \geq 0, \quad \sum_{n=1}^N \alpha_n t_n = 0,$$

the second of which comes from Equation 5.14. Notice that \mathbf{w} doesn't feature at all in this optimisation problem.

This optimisation problem is a constrained quadratic programming task, quadratic because of the $\alpha_m \alpha_n$ term. There is no analytical solution but it is reasonably straightforward to solve numerically. For example, the Matlab function `quadprog` solves problems such as these.

5.3.2.3 Making predictions

Given a set of optimal α_n , how do we go about making predictions? Our decision function, $t_{\text{new}} = \text{sign}(\mathbf{w}^\top \mathbf{x}_{\text{new}} + b)$, is based on \mathbf{w} and b , not α_n . To convert it into a function of α_n , we substitute the expression for \mathbf{w} given in Equation 5.13, resulting in:

$$t_{\text{new}} = \text{sign} \left(\sum_{n=1}^N \alpha_n t_n \mathbf{x}_n^\top \mathbf{x}_{\text{new}} + b \right). \quad (5.15)$$

To find b , we will use the fact that for the closest points, $t_n (\mathbf{w}^\top \mathbf{x}_n + b) = 1$. Substituting Equation 5.13 into this expression and re-arranging allows us to calculate b

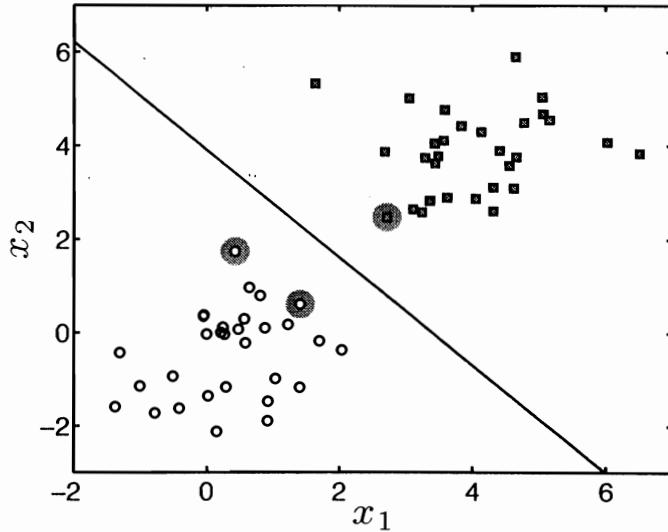


FIGURE 5.14: Decision boundary and support vectors for a linear SVM.

(note that $1/t_n = t_n$):

$$b = t_n - \sum_{m=1}^N \alpha_m t_m \mathbf{x}_m^\top \mathbf{x}_n \quad (5.16)$$

where x_n is any one of the closest points. This gives us everything we need to be able to classify any \mathbf{x}_{new} .

5.3.2.4 Support vectors

The set of points closest to the maximum margin decision boundary are known collectively as the support vectors. The name comes from the fact that they define, or support, the decision boundary. As the decision boundary is found by maximising the margin and as the margin only depends on the closest points, we could discard all of the other data and end up with just the same decision boundary. This is reflected by the fact that, at the optimum, all of the α_n that do not correspond to support vectors will be zero. If they were non-zero, they would have an influence on the decision function (see Equation 5.15). In many applications, this will lead to a sparse solution – the decision is a function of only a small subset of the training examples. For large problems, this can be a very useful feature. Consider classifying a test point using KNN when the training set consists of several thousand objects. To find the set of neighbours, distances must be computed between the new object and all of the training objects. For an SVM trained on the same data, the decision function might just involve a small subset of the training data.

Figure 5.14 shows a binary dataset and the resulting decision boundary ($\mathbf{w}^\top \mathbf{x} + b = 0$, where $\mathbf{w} = \sum_{n=1}^N \alpha_n t_n \mathbf{x}_n$) along with the three support vectors (large grey circles) (MATLAB script: `svmhard.m`). These are the only points for which $\alpha_n > 0$ and hence the only points that need to be used when classifying new data.

Although it could be considered efficient to base our decision on (in this case)

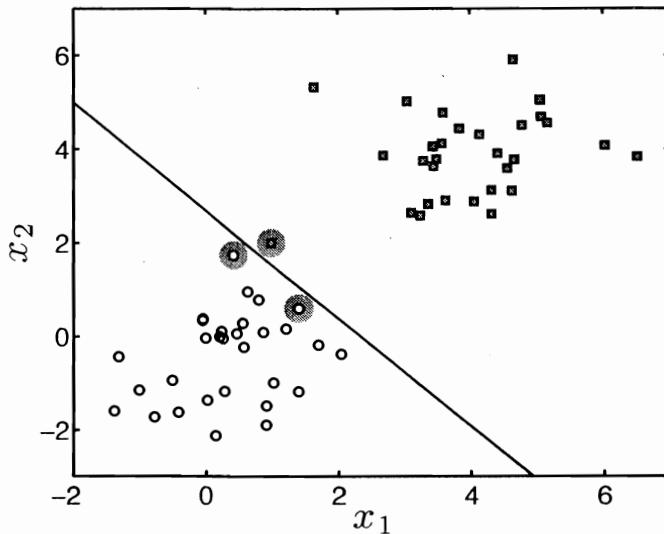


FIGURE 5.15: Decision boundary and support vectors for a linear SVM. The support vector from the grey square class appears to be exerting too much influence.

only three of the training points, it will not always be a good thing. To illustrate why, consider Figure 5.15 (MATLAB script: `svmhard.m`). This is the same data that we saw in Figure 5.14 with one difference – the support vector from the class denoted by grey squares has been moved closer to the other class. Moving this single data point has had a large effect on the position of the decision boundary. This is another example of over-fitting – we are allowing the data to have too much influence. To see why this happens, we need to look at our original constraints:

$$t_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1. \quad (5.17)$$

This means that all training points *have* to sit on the correct side of the decision boundary. This type of SVM is known as a *hard margin* SVM. It will sometimes be sensible (and lead to better generalisation performance) to relax this constraint. Fortunately, this is straightforward using a *soft margin*.

5.3.2.5 Soft margins

To allow points to potentially lie on the wrong side of the boundary, we need to slacken the constraints in our original formulation. In particular, we need to adapt Equation 5.17 so that it admits the possibility of some points lying closer to (or on the wrong side of) the decision boundary. To achieve this, the constraint becomes:

$$t_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1 - \xi_n, \quad (5.18)$$

where $\xi_n \geq 0$. If $0 \leq \xi_n \leq 1$, the point lies on the correct side of the boundary but within the boundary of the margin. If $\xi_n > 1$, the point lies on the wrong side of the

boundary. Our optimisation task becomes:

$$\begin{aligned} \operatorname{argmin}_{\mathbf{w}} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{n=1}^N \xi_n \\ \text{subject to } & \xi_n \geq 0 \text{ and } t_n(\mathbf{w}^T \mathbf{x}_n + b) \geq 1 - \xi_n \text{ for all } n. \end{aligned}$$

The new parameter C controls to what extent we are willing to allow points to sit within the margin band or on the wrong side of the decision boundary. If we follow the same steps that we took for the hard margin class we find that this change in the model has only a very small effect on the maximisation problem. Omitting the details (see Exercise EX 5.8), we now need to find the maximum of the following quadratic programming problem:

$$\begin{aligned} \operatorname{argmax}_{\mathbf{w}} \quad & \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n,m=1}^N \alpha_n \alpha_m t_n t_m \mathbf{x}_n^T \mathbf{x}_m \\ \text{subject to } & \sum_{n=1}^N \alpha_n t_n = 0 \text{ and } 0 \leq \alpha_n \leq C, \text{ for all } n. \end{aligned}$$

The only difference is an upper bound (C) on α_n . The influence of each training point in our decision function is proportional to α_n . We are therefore imposing an upper bound on the influence that any one training point can have. For the example in Figure 5.15, the support vector from the grey class had $\alpha_n = 5.45$. Setting C to 1 would result in a change in the decision boundary (some other α_n will have to become non-zero from the grey square class), moving it back towards the other objects in the grey square class. This is exactly what happens, as we can see from Figure 5.16, where we plot the decision boundary and support vectors for $C = 1$ and $C = 0.01$ (MATLAB script: `svnsoft.m`). As C decreases, the maximum potential influence of each training point is eroded and so more and more of them become active in the decision function.

Using a soft margin gives us a free parameter (C) that needs to be fixed. As with K for KNN, we can set this using cross-validation. The procedure and error measure are identical to those for KNN and so we omit details here. A final practical point is the computation of b . We can no longer use any support vector to compute it, as they will not all satisfy $t_n(\mathbf{w}^T \mathbf{x}_n + b) = 1$. Support vectors within the margin band (or on the wrong side) will have $t_n(\mathbf{w}^T \mathbf{x}_n + b) < 1$. To overcome this problem, find the support vector with the highest value of $\mathbf{w}^T \mathbf{x}_n$ (or $\sum_m \alpha_m t_m \mathbf{x}_m^T \mathbf{x}_n$), and compute b from Equation 5.16.

5.3.2.6 Kernels

Our study of SVMs thus far has been restricted to linear decision boundaries. The soft margin allows training points to reside on the wrong side of the decision boundary but this will not help if the data, like that shown in Figure 5.17, is more complex. When we wanted a nonlinear function from linear regression, we added some terms to \mathbf{x} and extended \mathbf{w} . With SVMs, we take a very different approach. The model remains the same (a linear decision boundary) but for data that has been transformed into some new space. The transformation is done in such a way as to make the transformed data classifiable with a linear decision boundary.

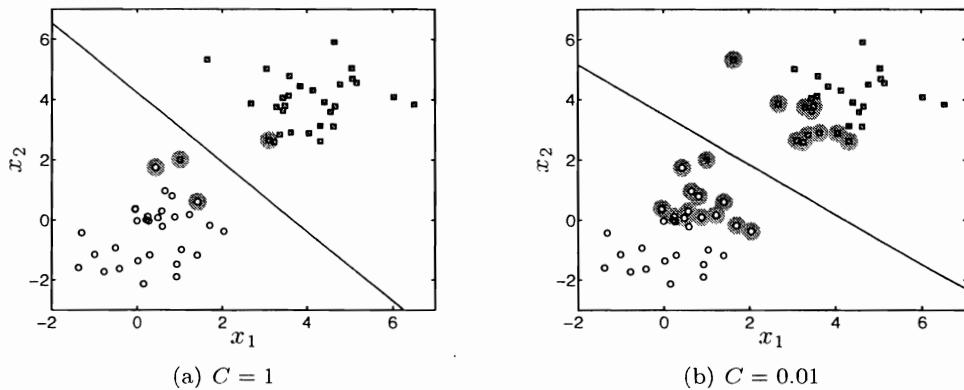


FIGURE 5.16: Decision boundary and support vectors for a linear SVM with a soft margin for two values of the margin parameter C . The influence of the stray support vector has been reduced.

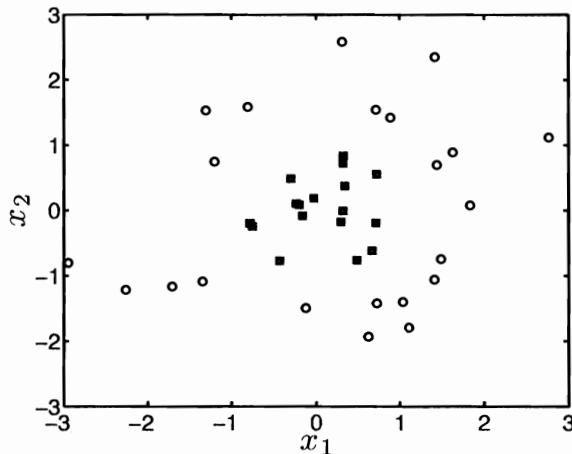


FIGURE 5.17: A binary dataset for which a linear decision boundary would not be appropriate.

To illustrate this idea, consider the data in Figure 5.17. The data cannot be separated by a straight line. However, if, instead of representing each data point by $\mathbf{x}_n = [x_{n1}, x_{n2}]^\top$, we represented them by their distance from the origin, $z_n = x_{n1}^2 + x_{n2}^2$, we could separate them with a straight line: the points in the circle class look consistently further from the origin than those in the square class. Using z_n instead of \mathbf{x}_n in the SVM, we do not need to modify the algorithm at all. When we have a test point, \mathbf{x}_{new} , we compute z_{new} and then classify it in the normal manner. In general, we will use $\phi(\mathbf{x}_n)$ to denote a transformation of the n th training object.

Perhaps the most important characteristic of the SVM framework is that we never actually have to perform the transformations. In our objective and decision function, the data $\mathbf{x}_n, \mathbf{x}_m, \mathbf{x}_{\text{new}}$ appear exclusively within inner (or dot) products:

$\mathbf{x}_n^T \mathbf{x}_m$, $\mathbf{x}_n^T \mathbf{x}_{\text{new}}$ etc. We never see an \mathbf{x} on its own. After applying the transformation, we need to calculate these inner products in the new space: $\phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$. We could explicitly transform each data point and compute the inner products of the transformed space. However, we do not actually need to think in terms of transformations at all. Instead, if we can show that some function $k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$ for some transformation $\phi(\cdot)$, we are free to use $k(\mathbf{x}_n, \mathbf{x}_m)$ in our expression in place of any inner product in the original space. Functions that correspond to inner products in some space are known as *kernel* functions.

Our optimisation and decision function (soft margin version) re-written to include kernel functions are:

$$\begin{aligned} \underset{\mathbf{w}}{\operatorname{argmax}} \quad & \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{n=1}^N \alpha_n \alpha_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \\ \text{subject to} \quad & \sum_{n=1}^N \alpha_n t_n = 0 \text{ and } 0 \leq \alpha_n \leq C, \text{ for all } n. \\ t_{\text{new}} = \operatorname{sign} \left(\sum_{n=1}^N \alpha_n t_n k(\mathbf{x}_n, \mathbf{x}_{\text{new}}) + b \right). \end{aligned}$$

There are plenty of off-the-shelf kernel functions (each equivalent to an inner product after some transformation) that we can use. The following are probably the three most popular:

$$\begin{aligned} \text{linear} \quad k(\mathbf{x}_n, \mathbf{x}_m) &= \mathbf{x}_n^T \mathbf{x}_m \\ \text{Gaussian} \quad k(\mathbf{x}_n, \mathbf{x}_m) &= \exp \left\{ -\gamma (\mathbf{x}_n - \mathbf{x}_m)^T (\mathbf{x}_n - \mathbf{x}_m) \right\} \\ \text{polynomial} \quad k(\mathbf{x}_n, \mathbf{x}_m) &= (1 + \mathbf{x}_n^T \mathbf{x}_m)^\gamma. \end{aligned} \quad (5.19)$$

The linear kernel is equivalent to the SVM that we have been using thus far. The Gaussian and polynomial kernels are more flexible and both have additional parameters (γ) that must be set by the user – normally via cross-validation.

The results of using a Gaussian kernel for the data shown in Figure 5.17 can be seen in Figure 5.18 for $\gamma = 1$ and $C = 10$ (all $\alpha_n < C$ so this is effectively a hard margin) (MATLAB script: `svmgauss.m`). The decision boundary looks reasonable. For the original SVM, we could compute the decision boundary exactly as it consisted of the values of \mathbf{x} that satisfied:

$$\mathbf{w}^T \mathbf{x} + b = 0.$$

We can no longer compute \mathbf{w} as it would be given by $\sum_n \alpha_n t_n \phi(\mathbf{x}_n)$ and we do not necessarily know $\phi(\mathbf{x}_n)$ (we only know $k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$). Therefore, to draw this decision boundary, we have had to evaluate $\sum_n \alpha_n t_n k(\mathbf{x}_n, \mathbf{x}_{\text{new}})$ over a grid of \mathbf{x}_{new} values and then use the Matlab `contour` function to draw the contour corresponding to $\sum_n \alpha_n t_n k(\mathbf{x}_n, \mathbf{x}_{\text{new}}) = 0$.

What happens if we change γ ? Modifying γ changes the (implicit) transformation $\phi(\mathbf{x}_n)$ which will in turn change the kind of decision boundaries we might expect to see when we view them in the original space (remember that they will be linear in the transformed space). For the Gaussian kernel, increasing γ has the effect of

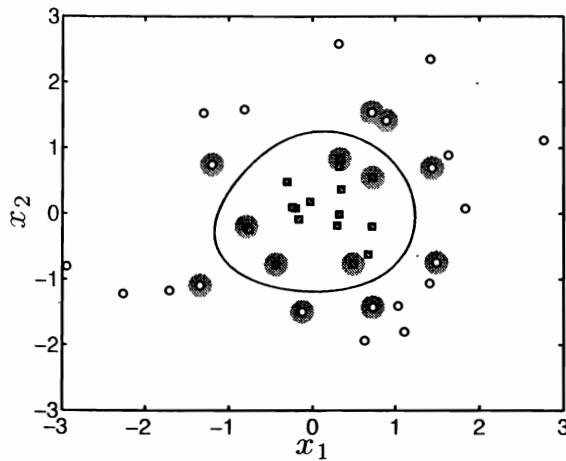


FIGURE 5.18: Decision boundary and support vectors for the dataset in Figure 5.17 using a Gaussian kernel with the kernel parameter $\gamma = 1$ and $C = 10$.

increasing the complexity of the decision boundaries in the original space. This is clear when we compare Figures 5.19(a) and 5.19(b), where we have used $\gamma = 0.01$ and $\gamma = 50$, respectively (MATLAB script: `svmgauss.m`). In Figure 5.19(a), the decision boundary is too simple – it is not able to curve rapidly enough in the original space to surround just the data from the square class. Conversely, when $\gamma = 50$ (Figure 5.19(b)) the decision boundary has too much flexibility, resulting in a decision boundary that looks far too complex. In both cases, it is also worth noticing that the number of support vectors has increased dramatically (c.f. Figure 5.18) and the solution can no longer be considered sparse.

This model complexity problem is exactly the same as the one we encountered in Chapter 1. There we found that increasing the polynomial order beyond a certain point resulted in poor predictions for our Olympics 100 m model. Here, models that are too simple (Figure 5.19(a)) or too complex (Figure 5.19(b)) will also produce bad predictions. In the too simple case, it looks like the model will predict a grey square too often and in the too complex case, the opposite. Just as in Chapter 1, we must be careful to set γ such that the complexity is just right, using, for example, cross-validation. To make matters worse, the parameters C and γ will affect the model in a coupled manner. We cannot optimise one and then optimise the other; we must do both at the same time. This is particularly problematic if our training dataset is large (N is high). The SVM solves an N -dimensional optimisation problem. For large N this might be very time consuming and a cross-validation based search over two parameters (C, γ) will result in performing this optimisation many times.

The SVM is not the only algorithm that can be kernelised. Many Machine Learning algorithms can be expressed in such a way that the data only appear inside inner products. This means that in a large number of algorithms, we can solve complex problems (e.g. fit highly nonlinear decision boundaries) without any additional algorithmic complexity. We will see another example when we look at clustering in the next chapter. We can also kernelise our other nonprobabilistic classifier, KNN. KNN

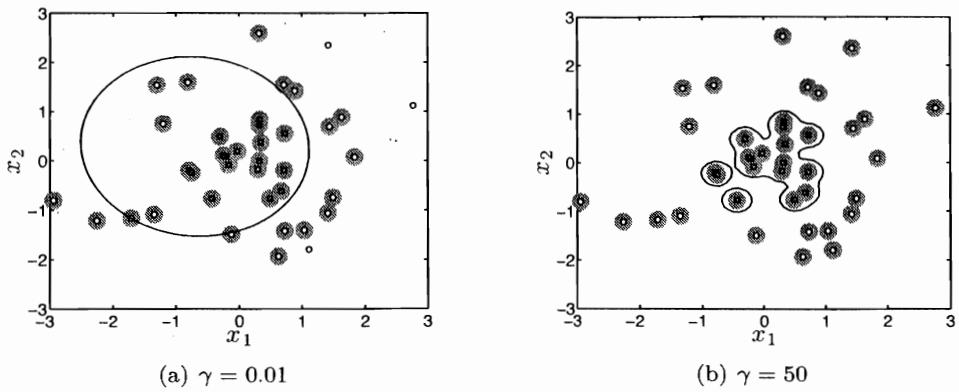


FIGURE 5.19: Decision boundary and support vectors for the dataset in Figure 5.17 using a Gaussian kernel with different values of the kernel parameter γ and $C = 10$.

requires the computation of distances between \mathbf{x}_{new} and each \mathbf{x}_n . This distance can be expressed as:

$$(\mathbf{x}_{\text{new}} - \mathbf{x}_n)^T (\mathbf{x}_{\text{new}} - \mathbf{x}_n).$$

If we multiply this out, we obtain just inner products:

$$\mathbf{x}_{\text{new}}^T \mathbf{x}_{\text{new}} - 2\mathbf{x}_{\text{new}}^T \mathbf{x}_n + \mathbf{x}_n^T \mathbf{x}_n.$$

Replacing this with its kernelised equivalent,

$$k(\mathbf{x}_{\text{new}}, \mathbf{x}_{\text{new}}) - 2k(\mathbf{x}_{\text{new}}, \mathbf{x}_n) + k(\mathbf{x}_n, \mathbf{x}_n),$$

gives us kernelised KNN.

5.3.3 Summary

In the previous sections we have described four popular classification algorithms and described how each can be used. These four algorithms provide a solid base from which one can experiment with data and explore the literature of other classification techniques.

Being able to apply a particular algorithm is only part of a classification analysis. It is also crucial to be able to reliably assess how well a particular classification is performing and that is the focus of the final section in this chapter.

5.4 Assessing classification performance

In the following discussion we shall assume that we are interested in assessing performance based on the predictions for some set of N independent test examples, $\mathbf{x}_1, \dots, \mathbf{x}_N$, with known labels t_1^*, \dots, t_N^* , to distinguish them from the labels predicted by the classifier – t_1, \dots, t_N . These could be from a completely independent dataset or could be the data held out in a particular cross-validation fold.

5.4.1 Accuracy – 0/1 loss

When we have needed to express a measure of performance we have used raw classification accuracy, also known as 0/1 loss. It is given this name as, for a particular test point, the loss is either 0 or 1, depending on whether the prediction is correct ($t_n = t_n^*$) or incorrect ($t_n \neq t_n^*$). When averaged over the N objects in the test set, this quantity gives the proportion of objects for which the classifier is *wrong*. This could be interpreted as an estimate of the probability that some random test point is incorrectly classified. Clearly, the lower this value the better. Although widely used, this measure does have some drawbacks. In particular, it is not always easy to place this quantity in context – i.e. how good is 0.2? Consider two hypothetical binary classification problems: In the first we observe roughly the same number of objects from each class and in the second, 80% of the objects we see come from class 1 and 20% from class 2. In the former case, an average 0/1 loss of 0.2 might represent very good performance. In the latter, it does not; we could always classify objects as belonging to class 1 and get an average loss of 0.2. We should therefore be very careful using 0/1 loss in applications where the classes are *imbalanced*. We will now introduce an alternative that overcomes this problem.

5.4.2 Sensitivity and specificity

Imagine a binary classification task that involves detecting disease. $t = 0$ corresponds to a healthy patient (\mathbf{x}) and $t = 1$ to a diseased patient. If we are attempting to detect a rare disease, 0/1 loss is a bad idea – diagnosing everyone as healthy will give us a very low 0/1 loss. Analysing two quantities known as sensitivity and specificity is a better idea. To compute sensitivity and specificity we need to extract four summary values from our classification results. These are the numbers of:

- True positives (TP) – the number of objects with $t_n^* = 1$ that are classified as $t_n = 1$ (diseased people diagnosed as diseased).
- True negatives (TN) – the number of objects with $t_n^* = 0$ that are classified as $t_n = 0$ (healthy people diagnosed as healthy).
- False positives (FP) – the number of objects with $t_n^* = 0$ that are classified as $t_n = 1$ (healthy people diagnosed as diseased).
- False negatives (FN) – the number of objects with $t_n^* = 1$ that are classified as $t_n = 0$ (diseased people diagnosed as healthy).

Given these values, we compute sensitivity as:

$$S_e = \frac{TP}{TP + FN} \quad (5.20)$$

and specificity as:

$$S_p = \frac{TN}{TN + FP}. \quad (5.21)$$

Both values lie between 0 and 1.

Broadly speaking, these two quantities tell us how good we are at detecting diseased and healthy people, respectively. The sensitivity is the proportion of the diseased people ($TP + FN$) that we correctly classify as being diseased (TP). Specificity is the proportion of all of the healthy people ($TN + FP$) that we correctly classify as being healthy (TN).

Considering our rare disease example, if we diagnosed everyone as healthy we would have a specificity of 1 (very good – we diagnose all healthy people correctly) but a sensitivity of 0 (we diagnose all unhealthy people incorrectly), which is very bad. Ideally we would like $S_e = S_p = 1$ – perfect sensitivity and specificity. This is unrealistic in all but the most trivial applications and we need a way to define how optimal a pair of sensitivity/specificity values are. For example, is $S_p = 0.9, S_e = 0.8$ better or worse than $S_p = 0.8, S_e = 0.9$? The answer will be application dependent. In our rare disease diagnosis, we do not want to mis-diagnose any diseased people but are probably happy to tolerate diagnosing some healthy people as diseased (they are likely to be subject to more tests and later discovered to be healthy). As such, we might be happy to reduce S_p in order to increase S_e . In other applications, we may have completely the opposite pressures.

It is often convenient to be able to combine sensitivity and specificity into a single value. This can be achieved through evaluating the area under the *receiver operating characteristic* (ROC) curve.

5.4.3 The area under the ROC curve

In many classification algorithms we are provided with a real valued output that is then thresholded to give a classification. For example, in (binary) Bayesian classification and logistic regression we are provided with $P(T_{\text{new}} = 1 | \mathbf{x}_{\text{new}}, \mathbf{X}, \mathbf{t})$ – a value between 0 and 1. In the SVM, we are provided with a real value that is then thresholded at 0 (passed through a sign function). We could use any threshold we liked for any of these algorithms to obtain a hard classification. For example, we may decide that if $P(T_n = 1 | \mathbf{x}_n, \mathbf{X}, \mathbf{t}) > 0.7$ then \mathbf{x}_n should belong to class 1. For the SVM we might decide to threshold at 0.2 rather than 0, making it slightly less likely that \mathbf{x}_n will be classified as belonging to class 1.

The receiver operating characteristic (ROC) curve lets us examine how the performance varies as we change this threshold. The sensitivity and specificity are calculated for a range of threshold values and sensitivity is plotted against the complementary specificity or false positive rate ($1 - S_p$), giving a curve that will typically look like those shown in Figure 5.20 (MATLAB script: `svmroc.m`). These curves are for the too simple and too complex models shown in Figures 5.19(a) and 5.19(b) evaluated on an independent test set of 1000 objects. We know that we want to make S_e and S_p as high as possible. Therefore, the closer the curve gets to the top left of the plot ($S_e = 1, 1 - S_p = 0$), the better. If the curve hits the top left corner,

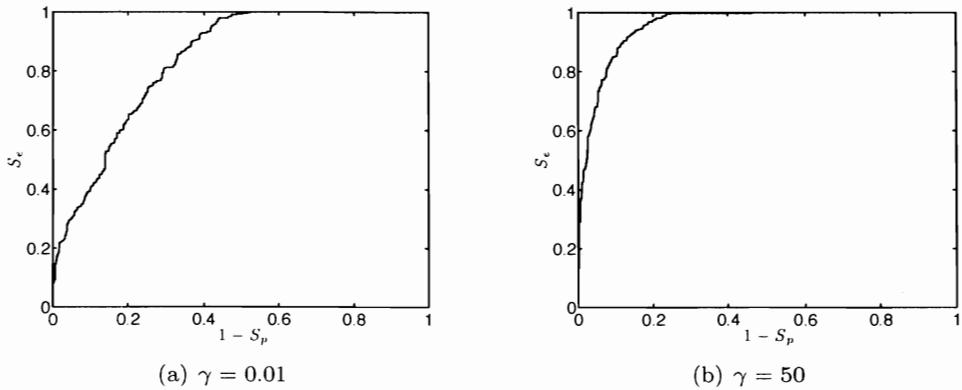


FIGURE 5.20: ROC curves for the SVMs shown in Figures 5.19(a) and 5.19(b).

it tells us that there is a threshold we can choose that will classify the data perfectly. The curve will always start at $S_e = 0, 1 - S_p = 0$, corresponding to a threshold that never classifies anything as belonging to class 1, and finish at $S_e = 1, 1 - S_p = 1$, corresponding to a classifier that never classifies anything as belonging to class 0 (-1 in the SVM case). As the classifier gets worse, the curve will get closer to a straight line from $(0, 0)$ to $(1, 1)$. This is equivalent to randomly guessing the class.

Based on the plots in Figure 5.20, the SVM with $\gamma = 50$ gets closer to the top left corner and so is better than the one with $\gamma = 0.01$. We can quantify this performance by computing the area under the ROC curve (known as the AUC). A classifier that is able to perfectly classify the data will have an AUC of 1 (the curve will go straight up the left hand side and then straight across the top); a classifier that is guessing randomly will have an AUC of 0.5 (the curve will be approximately a straight line from $(0, 0)$ to $(1, 1)$, as mentioned in the previous paragraph). The two curves in Figure 5.20 have AUCs of 0.8348 and 0.9551, respectively. In Figure 5.21, we show the ROC curve for $\gamma = 1$ (the SVM plotted in Figure 5.18). The AUC in this case is 0.9936 – the best of the three, as we would expect.

The AUC is, in most applications, a better way of evaluating performance than the 0/1 loss. It takes class imbalance into account through its use of sensitivity and specificity. One drawback is that it does not generalise to the multi-class setting. One way of using it within a multi-class problem is to analyse the results of the classification as several binary problems. For example, if we have three classes, we might do three ROC analyses where each one looks at the binary problem created by considering class c against the rest. This would provide useful information about each classifier but it is not clear how the three AUC values could be combined. We will now look at one final performance analysis tool that is easily generalised to (and very useful for) multi-class classification.

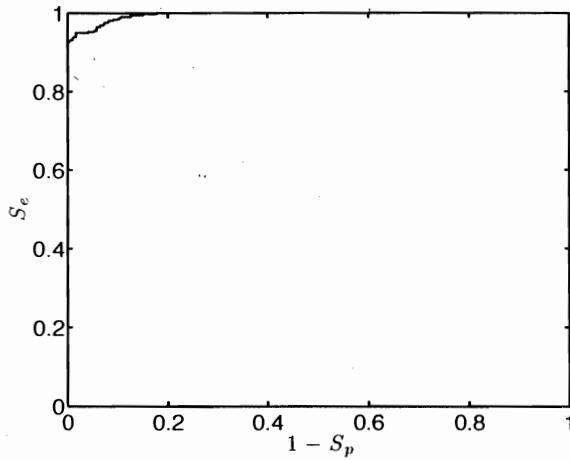


FIGURE 5.21: ROC curve for the SVM shown in Figure 5.18. The curve is hard to see because either $1 - S_p = 0$ or $S_e = 1$ for most threshold values.

TABLE 5.2: A binary confusion matrix.

		True class	
		1	0
Predicted class	1	TP	FP
	0	FN	TN

5.4.4 Confusion matrices

The four quantities (TP, TN, FP, FN) introduced in Section 5.4.2 are often usefully visualised in a table. In the case of two classes, this table will have two rows and two columns. The rows correspond to predicted class (t) and the columns to true class (t^*). The structure of this table is illustrated in Table 5.2 and it is known as a *confusion matrix*. Confusion matrices for binary problems just summarise these four values. However, where confusion matrices really come into their own is in multi-class problems. A confusion matrix for the 20 class newsgroup data would have 20 rows and 20 columns and would let us explore in detail what the algorithm was getting right and what it was getting wrong. This table can be seen in Table 5.3. At first glance, it looks rather unwieldy, but it is reasonably straightforward to extract useful information. For example, the high values on the diagonals tell us that on the whole, the classifier is doing pretty well. High off-diagonal elements tell us about mistakes that are being made regularly within the data. For example, 68 documents belonging to class 20 are incorrectly classified as belonging to class 16 – a phenomenon we already discussed in Section 5.2.1.6. Similarly, a large number of documents belonging to class 19 are incorrectly classified as belonging to class 17. This analysis not only lets us uncover where the mistakes are being made (details not available if we simply compute the 0/1 loss) but they also provide suggestions for how to improve performance. In Section 5.2.1.6 we saw that classes 20 and 16 were very similar, as were classes 19 and 17. Perhaps it is too difficult to distinguish

TABLE 5.3: Confusion matrix for the 20 class newsgroup data.

		True class																			
		Predicted class																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	242	3	3	0	1	0	0	1	0	4	2	0	2	18	7	8	2	0	1	1	3
2	0	296	33	8	8	42	9	1	1	0	0	0	1	0	1	0	0	0	0	0	0
3	0	6	209	15	9	8	4	0	0	0	0	0	1	28	3	0	0	0	0	0	0
4	0	12	60	303	36	12	46	2	0	1	0	0	2	7	0	0	1	0	0	0	0
5	0	8	10	22	277	2	21	0	0	1	0	0	1	3	0	1	2	0	0	1	0
6	1	21	30	2	2	304	0	1	0	3	0	1	1	0	0	0	1	0	0	0	0
7	0	1	0	5	5	1	235	5	1	2	0	1	1	0	9	4	0	0	2	1	0
8	0	3	1	6	4	0	31	356	25	3	1	0	0	9	4	0	0	1	1	0	1
9	0	2	2	0	1	2	5	4	353	1	0	0	2	0	1	0	0	1	1	0	0
10	0	0	2	0	1	1	0	0	2	2	348	4	0	0	1	0	0	1	1	0	0
11	1	0	1	1	0	0	1	0	0	16	382	0	0	1	0	0	1	0	1	1	0
12	1	16	16	5	4	10	3	1	1	2	0	360	45	0	4	1	3	4	3	1	0
13	1	4	1	24	16	0	9	5	1	2	0	3	260	3	4	0	0	0	0	0	0
14	2	3	4	0	8	0	2	0	1	0	2	2	6	324	4	1	1	0	3	3	3
15	3	7	4	1	2	3	3	2	0	0	1	0	4	3	336	0	2	0	7	5	68
16	39	4	5	0	0	1	3	1	1	3	2	2	5	17	4	3	3	7	2	95	19
17	4	0	0	0	3	1	1	5	4	1	0	9	0	3	1	3	325	3	325	4	5
18	7	1	0	0	0	1	3	1	2	2	1	0	2	6	2	1	2	1	2	325	4
19	7	2	9	0	6	2	5	8	5	8	4	8	0	10	21	1	16	19	185	7	92
20	10	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	2	4	0	1	0

between them. Class 20 appears to be the class that is misclassified most often and so if we wanted to improve performance perhaps we should concentrate here – maybe we could collect more data or consider combining it with another class.

5.5 Discriminative and generative classifiers

In our discussion, we subdivided classifiers into probabilistic and nonprobabilistic approaches. Another common way of partitioning classifiers is to split them depending on whether they are generative or **discriminative**. Generative classifiers define a model for each class and then assign new objects to the model that suits them best. On the other hand, discriminative classifiers explicitly define a decision boundary between classes. The Bayesian classifier (Section 5.2.1) is an example of a generative classifier and the SVM (Section 5.3.2) and logistic regression (Section 5.2.2) examples of discriminative classifiers.

5.6 Summary

In this chapter we have introduced four popular classification algorithms – two that provide probabilistic outputs and two that provide hard classifications. In the space available to us it is impossible to do them all justice – whole books have been written on SVMs and other kernel methods alone. However, the material presented here should be enough for the reader to be able to implement and experiment with these algorithms. In addition, it should provide enough background knowledge about the general problem of classification and the various *types* of classification algorithm that the reader can explore other algorithms and place them into some kind of context.

In addition to describing algorithms, we have also looked at how we might evaluate whether a classifier is doing well and some of the problems we might come up against. Again, we have only scraped the surface. There are many other performance measures favoured by different application areas and plenty of other problems that we might come up against.

5.7 Exercises

- EX 5.1. Assuming $\Sigma_c = \mathbf{I}$ for all classes, compute the posterior density $p(\boldsymbol{\mu}_c | \mathbf{X}^c)$ for the parameter $\boldsymbol{\mu}_c$ of a Bayesian classifier where the set of training objects in class c is given by $\mathbf{x}_1, \dots, \mathbf{x}_{N_c}$. Assume a Gaussian prior on $p(\boldsymbol{\mu}_c)$.

EX 5.2. Using the posterior computed in the previous exercise, compute the expected likelihood

$$p(\mathbf{x}_{\text{new}} | T_{\text{new}} = c, \mathbf{X}, \mathbf{t}) = \mathbf{E}_{p(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c | \mathbf{x}^c)} \{ p(\mathbf{x}_{\text{new}} | \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \}$$

EX 5.3. Compute the maximum likelihood estimates of $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ for class c of a Bayesian classifier with Gaussian class-conditionals and a set of N_c objects belonging to class c : $\mathbf{x}_1, \dots, \mathbf{x}_{N_c}$.

EX 5.4. Compute the maximum likelihood estimates of q_{mc} for class c of a Bayesian classifier with multinomial class-conditionals and a set of N_c , M -dimensional objects belonging to class c : $\mathbf{x}_1, \dots, \mathbf{x}_{N_c}$.

EX 5.5. For a Bayesian classifier with multinomial class-conditionals with M -dimensional parameters \mathbf{q}_c , compute the posterior Dirichlet for class c when the prior over \mathbf{q}_c is a Dirichlet with constant parameter α and the observations belonging to class c are the N_c observations $\mathbf{x}_1, \dots, \mathbf{x}_{N_c}$.

EX 5.6. Using the posterior computed in the previous exercise, compute the expected likelihood

$$p(\mathbf{x}_{\text{new}} | T_{\text{new}} = c, \mathbf{X}, \mathbf{t}) = \mathbf{E}_{p(\mathbf{q}_c | \mathbf{x}^c)} \{ p(\mathbf{x}_{\text{new}} | \mathbf{q}_c) \}$$

EX 5.7. Compute the MAP estimate of q_{cm} for the setup described in Exercise EX 5.4.

EX 5.8. Derive the dual optimisation problem for a soft margin SVM.

Further reading

- [1] Ken Binmore and Joan Davies. *Calculus: Concepts and Methods*. Cambridge University Press, 2002.
 - Includes good descriptions of the use of Lagrangian terms in optimisation.
- [2] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.
 - A comprehensive introduction to Support Vector Machines and other kernel techniques.
- [3] Richard Duda, Peter Hart, and David Stork. *Pattern Classification*. Wiley-Interscience, second edition, 2000.
 - A comprehensive textbook on the subject of classification.
- [4] T. Furey et al. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.
 - One of the first papers to apply the Support Vector Machine to very high dimension microarray data. Also describes a simple feature selection technique.
- [5] Brian Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.
 - A classic pattern recognition textbook from the late 1990s.
- [6] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
 - An accessible introduction to kernel techniques with examples of their use in many applications. This is a broader and more practically minded text than the other by the same authors.

Chapter 6

Clustering

Thus far we have been concerned with **supervised learning**. In all tasks, we have been provided with a set of data objects $\mathbf{x}_1, \dots, \mathbf{x}_N$ and their associated labels (or targets) t_1, \dots, t_N . For example, objects consisting of Olympics years and targets corresponding to 100 m winning times; objects consisting of documents and targets consisting of document categories. It is the presence of the targets, t_n , that makes the tasks supervised.

Sometimes we will just be supplied with objects \mathbf{x}_n without labels. The analysis of this kind of data requires **unsupervised** Machine Learning techniques. At first glance it is perhaps hard to understand what can be done with such data. There is certainly not much that can be done if we were told just the years of the modern Olympics.

In this and the following chapter, we will see two families of techniques that are used extensively in Machine Learning for exactly these unsupervised scenarios. In this chapter, we will look at the first: clustering. In particular we will consider two clustering methods – K -means and mixture models.

6.1 The general problem

The aim of cluster analysis is to create a grouping of objects such that objects within a group are similar and objects in different groups are not similar. There are many ways of defining what it means for two objects to be similar and many ways of performing the grouping once similarity is defined. Before we look at some in more detail, we shall first motivate cluster analysis with some examples.

Customer preference: Imagine you run a large online store and would like to personalise users' shopping experiences. Your motives are not entirely altruistic – you hope that by improving their shopping experience, users will buy more. One way to do this is to provide each user with a set of unique recommendations that they see when they access your site. You do not *directly* know each user's personal preferences and tastes but you do have lots of data – records of all purchases made by each user. This is classic machine learning territory – no fundamental model but lots of data.

Assuming that we can define a measure of similarity between customers based on their purchasing history, we could use cluster analysis to group customers into K groups. Within each group, customers have similar shopping patterns. Differences

between customers in the same group could form the basis of a recommender system. For example, customers A and B are in the same cluster because they share a number of purchases – perhaps they both have an interest in a particular sport. However, customer A has additionally bought several items that customer B has not. On the strength of their similarity, it might make sense for customer B to be recommended these items.

A recommender system could also be created by clustering the items based on the customers they were bought by. If items 1 and 2 were both bought by customers A, D, E and G then they could be considered similar. Customers could then be recommended items that were similar (in this sense) to items they had already bought.

Gene function prediction: A large proportion of research effort in molecular biology involves categorising genes into particular functional classes – i.e. what role does a particular gene play; what is its purpose? One potential source of information is mRNA microarray data – numerical values describing how active each gene is in a particular biological sample. For a collection of genes, this activity can be measured over time. If genes are clustered based on this representation, we obtain a grouping of genes such that genes in a particular group exhibit similar behaviour over time. Consider one such group (cluster) consisting of 10 genes. Half of the genes have known function, whereas the function of the other half is unknown. Given no additional evidence, it might be reasonable to assume that the unknown half have the same or similar function as those for which function is known. This will not always give the correct function, but it is a good starting point for additional analysis.

In this example, the structure present in the data exposed by cluster analysis has allowed us to make some prediction about the objects. It is interesting to note that this problem could alternatively be considered as a supervised classification problem where the genes with known function form the training set (the class labels consist of the different functions) and the unknown as a test set to be labeled by the algorithm.

6.2 *K*-means clustering

Consider the data shown in Figure 6.1. It consists of 100 objects, $\mathbf{x}_1, \dots, \mathbf{x}_{100}$, each represented by two attributes: $\mathbf{x} = [x_1, x_2]^\top$. When we plotted classification data, objects belonging to different classes were plotted with different symbols. Now we have no class information – all of the dots look the same.

If you were to partition these objects into groups by hand such that groups contained similar objects, you might come to the conclusion that there were three groups. Most objects fall quite obviously into one of these three groups, although there are a few that are more ambiguous (e.g. the point at $\mathbf{x} \approx [2.5, -1]^\top$).

By clustering the data in this manner, we have implicitly defined what ‘similar’ means – similar objects are those that are close to one another in terms of squared distance (i and j are similar if $(\mathbf{x}_i - \mathbf{x}_j)^\top(\mathbf{x}_i - \mathbf{x}_j) = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2$ is low). Provided with no additional information about the data or the purpose of the cluster analysis, this is a reasonable measure of similarity. There are other ways of

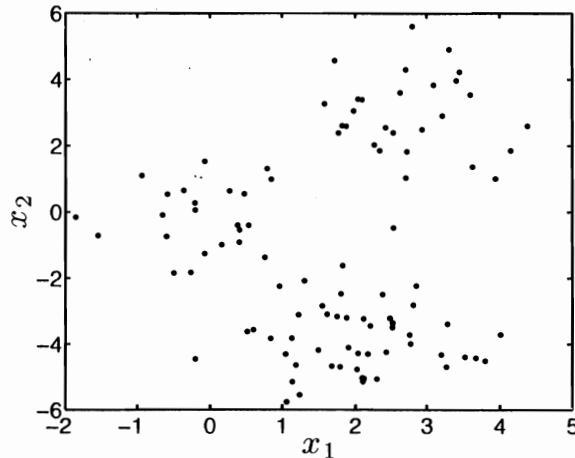


FIGURE 6.1: Synthetic dataset for clustering examples.

defining similarity that may be more appropriate, for example, the **Mahalanobis distance**, $(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j)$. Both of these distances are suitable for real-valued data. For data of other types (for example, text), different distance measures would be required.

To develop an algorithm that can perform this grouping automatically, we need to define what a cluster is more formally. K -means defines a cluster as a representative point, just like one of the data objects. The point is defined as the mean of the objects that are assigned to the cluster (hence the name K -means). We will use μ_k to define the mean point for the k th cluster and z_{nk} as a binary indicator variable that is 1 if object n is assigned to cluster k and zero otherwise. Each object has to be assigned to one, and only one cluster, i.e. $\sum_k z_{nk} = 1$. This leads us to the following expression for μ_k :

$$\mu_k = \frac{\sum_n z_{nk} \mathbf{x}_n}{\sum_n z_{nk}} \quad (6.1)$$

Each object is assigned to the cluster to which it is closest, i.e. the cluster k that gives the minimum value of $(\mathbf{x}_n - \mu_k)^T (\mathbf{x}_n - \mu_k)$ (or some other suitable distance).

This is a circular argument: the clusters are defined as the centres of the points assigned to them, and the points are assigned to their closest clusters. If we know the clusters, μ_1, \dots, μ_K , we can compute the assignments, but without the assignments we cannot compute the clusters. K -means clustering overcomes this problem with an iterative scheme. Starting with initial (random) values for the cluster means, μ_1, \dots, μ_K :

1. For each data object, \mathbf{x}_n , find k that minimises $(\mathbf{x}_n - \mu_k)^T (\mathbf{x}_n - \mu_k)$ (i.e. find the closest cluster mean) and set $z_{nk} = 1$ and $z_{nj} = 0$ for all $j \neq k$.
2. If all of the assignments (z_{nk}) are unchanged from the previous iteration, stop.
3. Update each μ_k with Equation 6.1.
4. Return to 1.

Figure 6.2 illustrates the operation of the algorithm for the data shown in Figure 6.1 (MATLAB script: `kmeansexample.m`). Figure 6.2(a) shows the initial guesses for the means (large symbols) with the data objects given the symbol corresponding to their closest mean. The means are now updated according to Equation 6.1 and Figure 6.2(b) shows the means moving to their new locations. Now that the means have changed, the objects must be re-assigned and the new assignments as well as the resulting change in the means can be seen in Figure 6.2(c). Figures 6.2(d) and 6.2(e) show the status after three and five iterations, respectively. After eight iterations, the algorithm has converged and the final assignments (the clustering) can be seen in Figure 6.2(f). The point at $\mathbf{x}_n = [2.5, -1]^T$ appears to be incorrectly assigned – this is simply due to the scaling of the axis.

This iterative scheme is guaranteed to converge to a local minimum of the following quantity:

$$D = \sum_{n=1}^N \sum_{k=1}^K z_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)^T (\mathbf{x}_n - \boldsymbol{\mu}_k), \quad (6.2)$$

which can be interpreted as the total distance between the objects and their respective cluster centres. However, it is not guaranteed to reach the lowest possible value (the global minimum). Whether it does or not will depend on the initial guesses of the cluster means. For K -means, this problem can never be totally overcome unless we evaluate every possible way of assigning all N points to K clusters, which is infeasible for even small values of N and K . A more common way to partly overcome this limitation is to run the algorithm from several random starting points and use the solution that gives the lowest value of the total distance.

6.2.1 Choosing the number of clusters

In order to use K -means, we need to choose a value of K – the number of clusters. Determining the number of clusters is a common problem in cluster analysis. Recall that K -means produces a clustering that corresponds to a local minima of Equation 6.2. Unfortunately, in much the same way that likelihood turned out to be a poor model selection criteria (recall that it monotonically increases as models become more complex, e.g. Figure 2.11(a)), D is no good either. Figure 6.3 shows $\log D$ as K is increased (MATLAB script: `kmeansK.m`). For each value of K we used 50 random initialisations of the algorithm and the boxplots show the median value, the 25th and 75th percentiles and any outliers. It is clear that $\log D$ (and hence D) decreases as K is increased. As K increases, large clusters will be broken down into smaller and smaller parts. The smaller each cluster, the closer each point will get (on average) to its cluster mean, reducing its contribution to D . Taken to the extreme case of $K = N$, it is possible to get $D = 0$ when each cluster contains just one object and $\boldsymbol{\mu}_k = \mathbf{x}_n$.

There is no straightforward solution to this model selection problem. To overcome it, it is often useful to look beyond the clustering to the overall aim of the analysis. For example, in the introduction to this chapter, we mentioned a recommendation system that clustered customers. The grouping is done in order to obtain a compact representation of the data and provide customer-product recommendations. Therefore, it makes sense to choose the number of clusters that produces the

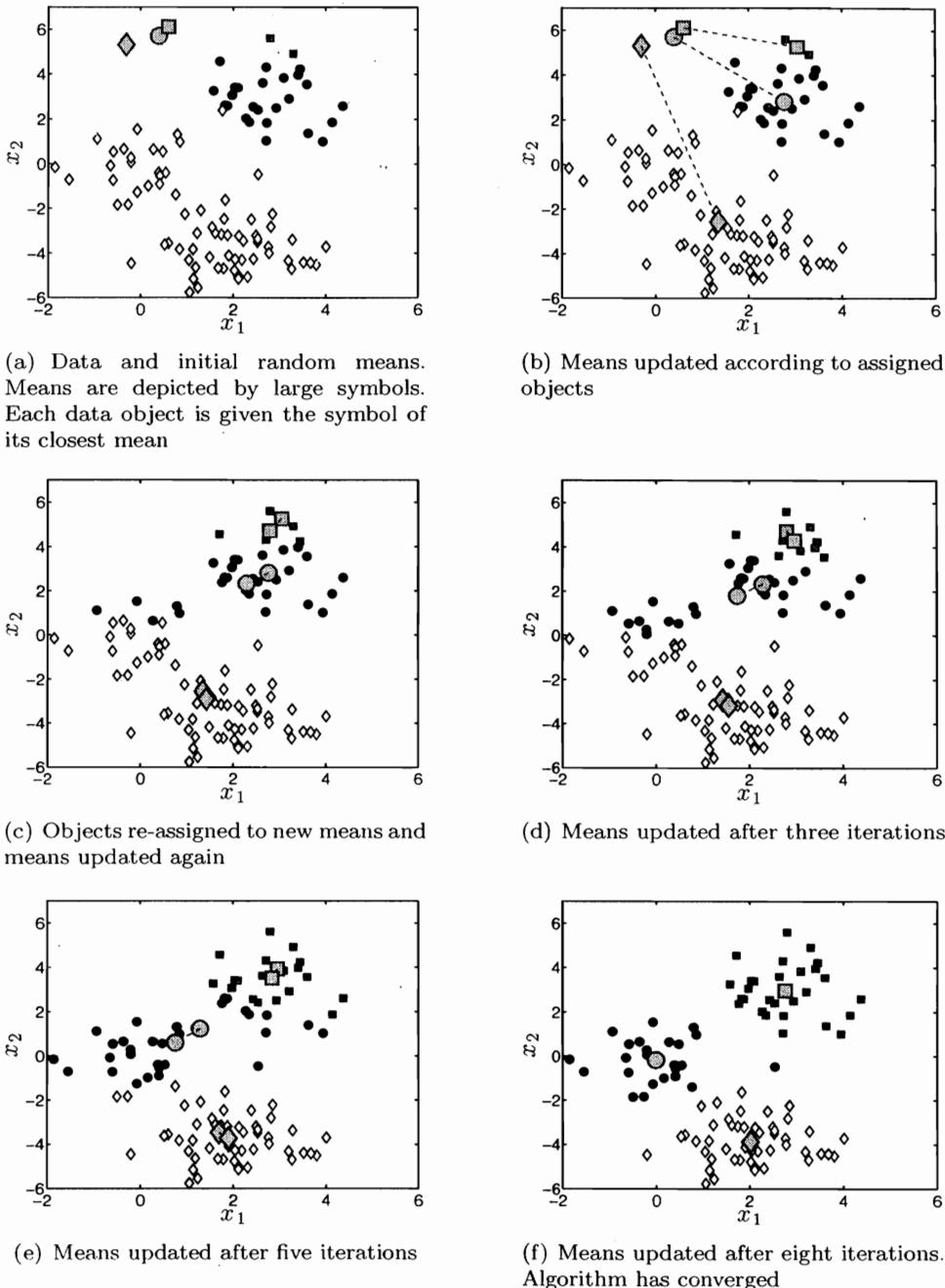


FIGURE 6.2: Illustration of the K -means algorithm. Data objects are represented as small symbols, means as large symbols. Objects are given the symbol of the mean to which they're assigned.

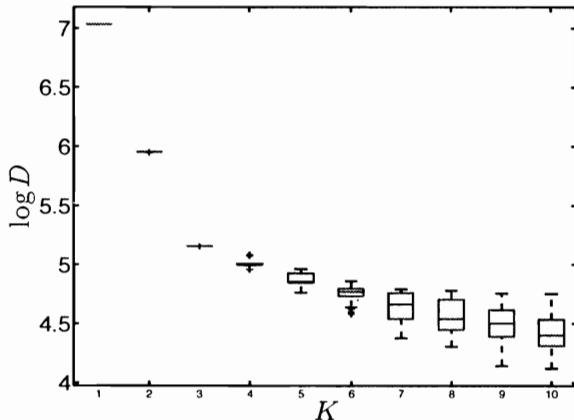


FIGURE 6.3: $\log D$ (where D is defined in Equation 6.2) as K increases for the data shown in Figure 6.1. Each boxplot is the result of 50 random initialisations of the K -means algorithm.

best recommendations, perhaps on some validation data. Similarly, clustering is a popular choice for **feature selection** in classification – clustering features based on their values across objects rather than clustering objects (\mathbf{X}^\top rather than \mathbf{X}). In this instance, K should be chosen to give the best classification performance.

6.2.2 Where K -means fails

Figure 6.4 shows two datasets on which K -means has failed to extract what looks like the true cluster structure. In both cases, the objects in the true clusters do not necessarily conform to our current notion of similarity (distance). In the first example, Figure 6.4(a), the data exist in concentric circles. It is clear that standard K -means can never work in this setting, as the means of both circles are in the same place. In the second example, Figure 6.4(b), the clusters are stretched in such a way (check the scaling of the axes) that objects at the top of the right hand cluster are closer to the mean of the left hand cluster (the means are shown in this plot as large symbols).

In the next section, we will cluster the data in Figure 6.4(a) by kernelising the K -means algorithm. For the data in Figure 6.4(b) we will turn, from Section 6.3 onwards, to an alternative clustering method: mixture models.

6.2.3 Kernelised K -means

We can extend K -means using the kernel substitution trick that we introduced in Chapter 5. At an abstract level, the idea is the same: rather than making the algorithm more complex, we will transform the data into a space in which our simple algorithm works. We shall highlight this approach using the data shown in Figure 6.4(a).

We have seen that rather than actually performing the transformation of the data, kernel methods use kernel functions to directly compute inner (dot) products

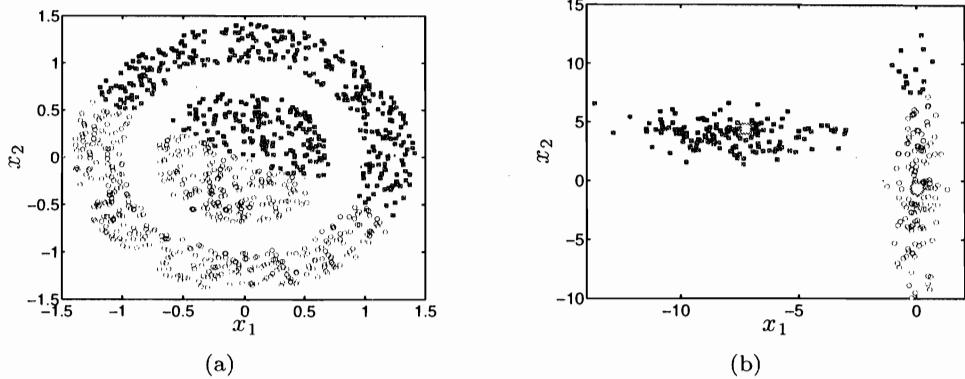


FIGURE 6.4: Two datasets in which K -means fails to capture the clear cluster structure.

in the transformed space. As such, any algorithm where the data objects, $\mathbf{x}_1, \dots, \mathbf{x}_N$, only appear as inner products ($\mathbf{x}_i^\top \mathbf{x}_j$, etc.) can be given the kernel treatment, making it more powerful without any significant additional cost. Key to the operation of K -means is the computation of the distance between the n th object and the k th mean:

$$d_{nk} = (\mathbf{x}_n - \boldsymbol{\mu}_k)^\top (\mathbf{x}_n - \boldsymbol{\mu}_k),$$

where the mean, $\boldsymbol{\mu}_k$, is calculated according to Equation 6.1. Substituting this into the expression for d_{nk} gives:

$$d_{nk} = \left(\mathbf{x}_n - \frac{1}{N_k} \sum_{m=1}^N z_{mk} \mathbf{x}_m \right)^\top \left(\mathbf{x}_n - \frac{1}{N_k} \sum_{r=1}^N z_{rk} \mathbf{x}_r \right),$$

where $N_k = \sum_{n=1}^N z_{nk}$, the number of objects assigned to cluster k .

Multiplying out this expression results in the data (\mathbf{x}_n) only appearing in product terms:

$$d_{nk} = \mathbf{x}_n^\top \mathbf{x}_n - \frac{2}{N_k} \sum_{m=1}^N z_{mk} \mathbf{x}_n^\top \mathbf{x}_m + \frac{1}{N_k^2} \sum_{m=1}^N \sum_{r=1}^N z_{mk} z_{rk} \mathbf{x}_m^\top \mathbf{x}_r.$$

All that remains is to replace the inner products with kernel functions to give a kernelised distance:

$$d_{nk} = K(\mathbf{x}_n, \mathbf{x}_n) - \frac{2}{N_k} \sum_{m=1}^N z_{mk} K(\mathbf{x}_n, \mathbf{x}_m) + \frac{1}{N_k^2} \sum_{m=1}^N \sum_{r=1}^N z_{mk} z_{rk} K(\mathbf{x}_m, \mathbf{x}_r). \quad (6.3)$$

This distance is purely a function of the data and the current assignments; the cluster means do not appear. In fact, it is not, in general, possible to actually compute the cluster means in the transformed space. The original expression for the mean of cluster k is:

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^N z_{nk} \mathbf{x}_n}{\sum_{n=1}^N z_{nk}},$$

and the kernelised version is:

$$\mu_k = \frac{\sum_{n=1}^N z_{nk} \phi(\mathbf{x}_n)}{\sum_{n=1}^N z_{nk}}.$$

Within this expression, data objects appear on their own and not as inner products. In Chapter 5 we discussed how, for most kernel functions, we cannot compute the transformation ($\mathbf{x}_n \rightarrow \phi(\mathbf{x}_n)$); we can only compute inner products in the transformed space ($(\phi(\mathbf{x}_n))^T \phi(\mathbf{x}_m)$). If we are unable to compute the transformation, we cannot compute μ_k .

Equation 6.3 suggests the following procedure for kernelised K -means:

1. Randomly initialise z_{nk} for each n (see below).
2. Compute d_{n1}, \dots, d_{nK} for each object using Equation 6.3.
3. Assign each object to the cluster with the lowest d_{nk} .
4. If assignments have changed, return to step 2, otherwise stop.

In standard K -means, we initialised the algorithm by randomly setting the means μ_1, \dots, μ_K . In kernel K -means we do not have access to the means and we therefore initialise the algorithm via the object-cluster assignments, z_{nk} . We could do this completely randomly – for each n set one z_{nk} to 1 and all of the others (z_{nl} , $l \neq k$) to zero but, given that we know K -means to be sensitive to initial conditions, it might be better to be more careful. Alternatively, we could run standard K -means and use the values of z_{nk} at convergence. This has the advantage that we can be sure that objects within the same cluster will be reasonably close to one another (something that we cannot guarantee if we set them randomly). A second alternative would be to assign $N - K + 1$ objects to cluster 1 and the remaining $K - 1$ objects to their own individual clusters. The performance of each iteration scheme will depend on the particular characteristics of the data being clustered.

Figure 6.5 shows the result of applying the kernel K -means algorithm to the data shown in Figure 6.4(a) (MATLAB script: `kernelkmeans.m`). In this case, we have initialised by assigning all but one object to the ‘circle’ cluster and the remaining object to the ‘square’ cluster. A Gaussian kernel was used with $\gamma = 1$ (see Equation 5.19). Figure 6.5(a) shows the assignments one iteration after initialisation. As the algorithm progresses through 5, 10 and 30 iterations (Figures 6.5(b), (c) and 6.5(d), respectively) the smaller cluster grows to take up the central circle. At convergence (Figure 6.5(d)), we can see that the algorithm has captured the interesting structure in the data.

Not only does kernel K -means allow us to find clusters that do not conform to our original idea of similarity, it also opens the door to performing analysis on other data types. We can cluster any type of data for which a kernel function exists and it is hard to find a data type for which there does not. Obvious examples are kernels for text (each object is a document) and kernels for graphs or networks. The latter is used widely in computational biology.

6.2.4 Summary

In the previous sections we introduced the K -means algorithm and showed how it could be kernelised. One of the great advantages of K -means is its simplicity

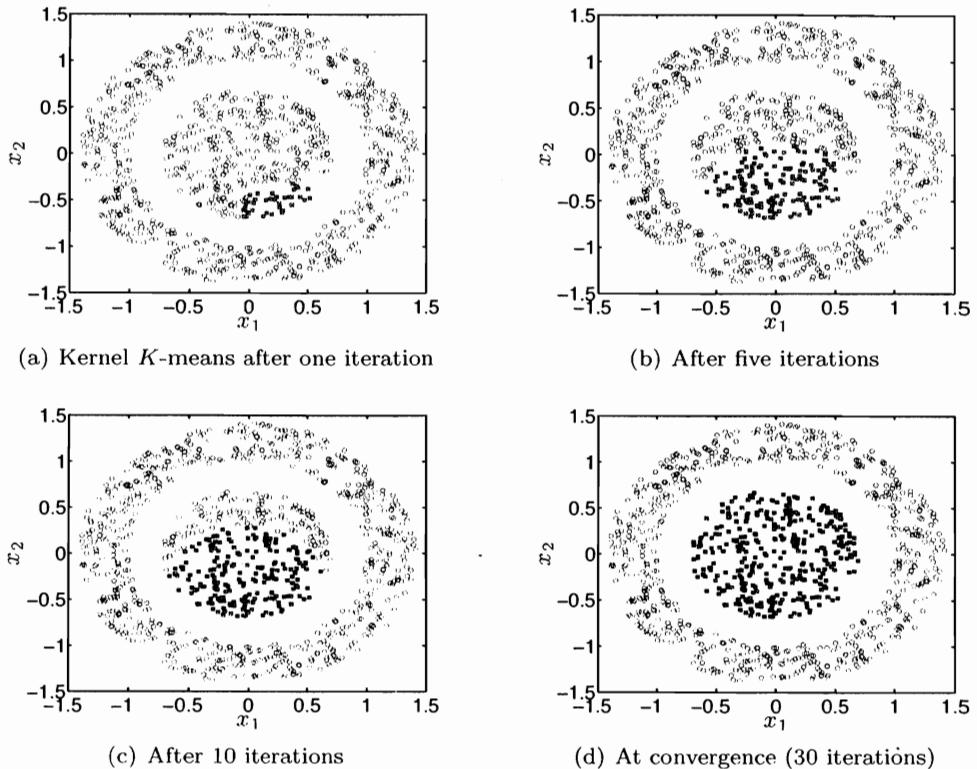


FIGURE 6.5: Result of applying kernelised K -means to the data shown in Figure 6.4(a).

– it is very easy to use and poses no great computational challenge. However, its simplicity is also a drawback: assuming that a cluster can be represented by a single point will often be too crude. In addition, there is no objective way to determine the number of clusters if our aim is just to cluster (remember that we mentioned how the number of clusters could be chosen as the one that gave best performance in some later task like classification). To overcome some of these drawbacks, we will now describe clustering with statistical mixture models. These models share some similarities with K -means but offer far richer representations of the data.

6.3 Mixture models

In Figure 6.4(b) we showed a dataset for which the original K -means failed. The two clusters were stretched in such a way that some objects that should have belonged to one were in fact closer to the centre of the other. The problem our K -means algorithm had here was that its definition of a cluster was too crude. The

characteristics of these stretched clusters cannot be represented by a single point and the squared distance. We need to be able to incorporate a notion of shape. Statistical mixture represents each cluster as a probability density. This generalisation leads to a powerful approach as we can model clusters with a wide variety of shapes in almost any type of data.

6.3.1 A generative process

In Section 2.1.1 we motivated a probabilistic treatment of the linear model described in Chapter 1 by creating a process by which the data could have been generated. In that case, we combined (by adding them together) a deterministic function of the form $\mathbf{w}^\top \mathbf{x}_n$ with a Gaussian random variable with zero mean and variance σ^2 . Data generated in this way was qualitatively similar to our real data. Note that we never tried to claim that this *was* the process by which the data was generated; it was merely an abstraction that would allow us to build a better model. We will use much the same motivation to move from K -means to statistical mixture models.

Our synthetic clustering dataset is reproduced in Figure 6.6. How could we generate data that *looks* like this? The data in Figure 6.6 does not look like samples from any density function that we have encountered. There appear to be three disjoint regions in which data are concentrated. None of the density functions that we have seen can produce data with this complex structure. However, each of the three regions looks simple enough to generate on its own. In fact, they all look a bit like samples from two-dimensional Gaussians.

Assuming that the data was generated by three separate Gaussians suggests a two step procedure for sampling the n th data object, \mathbf{x}_n :

1. Select one of the three Gaussians.
2. Sample \mathbf{x}_n from this Gaussian.

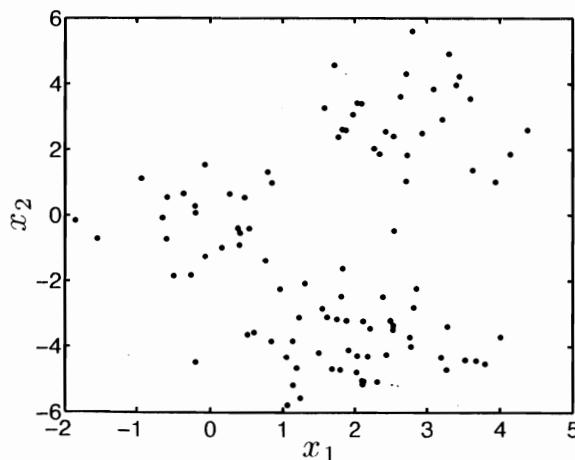


FIGURE 6.6: Synthetic dataset for clustering examples.

Both of these steps are straightforward. Step 1 involves choosing one value from a discrete set, like rolling a die. To do this, we just need to define the probability of each outcome, π_k , subject to the constraint $\sum_k \pi_k = 1$. Having chosen which Gaussian to sample from, the second step is straightforward.

To illustrate this process, we will sample some data from a setup with $K = 2$ Gaussians. As in K -means, we will use z_{nk} as an indicator variable. If we choose the k th component as the source of the n th object, we set $z_{nk} = 1$, and $z_{nj} = 0$ for all $j \neq k$. We will use μ_k and Σ_k to denote the parameters of the k th Gaussian.

The density function for \mathbf{x}_n , given that it was produced by the k th component ($z_{nk} = 1$), is a Gaussian with mean and covariance μ_k and Σ_k , respectively:

$$p(\mathbf{x}_n | z_{nk} = 1, \mu_k, \Sigma_k) = \mathcal{N}(\mu_k, \Sigma_k)$$

For our example, we will use the following means and covariances for the two components:

$$\mu_1 = [3, 3]^\top, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad \mu_2 = [1, -3]^\top, \Sigma_2 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad (6.4)$$

Finally, we need to define π_k . Assuming that component 1 is more likely than component 2, we will use $\pi_1 = 0.7$, $\pi_2 = 0.3$. Figure 6.7 shows the first 50 generated data object and the density functions of the two Gaussians (MATLAB script: `mixgen.m`). For the first point, $k = 2$ is chosen and the object sampled from the second (lower) component – Figure 6.7(a). Figure 6.7(b) shows the first five objects (the most recent is always denoted as a larger circle). We notice that all but the first one have come from the first component. This is not surprising as the first component is more likely than the second: $\pi_1 > \pi_2$. Figures 6.7(c) and 6.7(d) show the first 10 and 50 data objects, respectively. If we compare Figure 6.7(d) with Figure 6.6, we can see that, although the datasets are different, they share certain qualities. In particular, it looks like the data in Figure 6.6 could have been generated in a similar manner to the generative procedure shown in Figure 6.7.

The generative procedure that we have described is the generative procedure for a mixture model – the data are assumed to have been sampled from a *mixture* of several individual density functions. Mixture models find a wide variety of uses in data modelling as fitting a set of simple distributions is often more straightforward than fitting one more complex one. Within the context of clustering, each individual component can be viewed as a cluster – all objects for which $z_{nk} = 1$ are in the k th cluster. Our learning task is to infer, from the observed data, the component parameters (μ_k, Σ_k) and the assignments of objects to components. As with K -means, this is a circular argument: the component parameters would be easy to compute if we knew the assignments and the assignments would be easy to compute if we knew the component parameters. Without either, it is hard to know where to start. The answer comes in the form of the Expectation-Maximisation (EM) algorithm – an iterative maximum likelihood technique that is used for a wide range of models and has parallels with the K -means algorithm we introduced earlier in the chapter.

6.3.2 Mixture model likelihood

To derive the steps required in the EM algorithm, we need an expression for the likelihood. To keep this as general as possible, we will work with $p(\mathbf{x}_n | z_{nk} = 1, \Delta_k)$

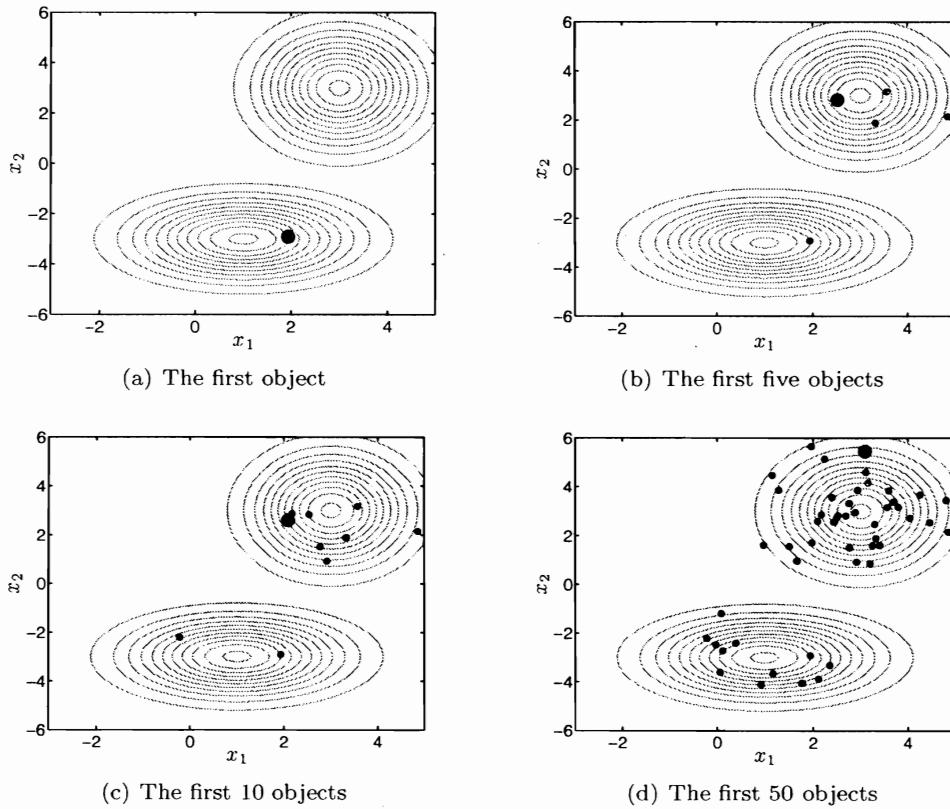


FIGURE 6.7: Generating data from two Gaussians.

where Δ_k denotes the parameters of the k th density (not necessarily Gaussian). In addition, Δ will denote the collection of the parameters of all of the mixture components $\Delta = \{\Delta_1, \dots, \Delta_K\}$ and we will collect all of the π_k together into a vector, $\pi = \{\pi_1, \dots, \pi_K\}$.

We require the likelihood of the data objects \mathbf{x}_n under the whole model: $p(\mathbf{x}_n | \Delta, \pi)$. To obtain this expression, we start with the likelihood of a particular data object conditioned on $z_{nk} = 1$:

$$p(\mathbf{x}_n | z_{nk} = 1, \Delta) = p(\mathbf{x}_n | \Delta_k).$$

To obtain $p(\mathbf{x}_n | \Delta, \pi)$, we need to get rid of z_{nk} . To do this, we first multiply both sides by $p(z_{nk} = 1)$, which we have defined as π_k :

$$\begin{aligned} p(\mathbf{x}_n | z_{nk} = 1, \Delta)p(z_{nk} = 1) &= p(\mathbf{x}_n | \Delta_k)p(z_{nk} = 1) \\ p(\mathbf{x}_n, z_{nk} = 1 | \Delta, \pi) &= p(\mathbf{x}_n | \Delta_k)\pi_k. \end{aligned}$$

Summing both sides over k (marginalising over the individual components) yields

the likelihood:

$$\sum_{k=1}^K p(\mathbf{x}_n, z_{nk} = 1 | \Delta, \pi) = \sum_{k=1}^K p(\mathbf{x}_n | \Delta_k) \pi_k$$

$$p(\mathbf{x}_n | \Delta, \pi) = \sum_{k=1}^K \pi_k p(\mathbf{x}_n | \Delta_k).$$

Making the standard independence assumption, we can extend this to the likelihood of all N data objects:

$$p(\mathbf{X} | \Delta, \pi) = \prod_{n=1}^N \sum_{k=1}^K \pi_k p(\mathbf{x}_n | \Delta_k). \quad (6.5)$$

6.3.3 The EM algorithm

We shall now demonstrate the use of the EM algorithm to maximise the likelihood given in Equation 6.5. It is normally easier to work with the logarithm of the likelihood and so taking the natural logarithm of Equation 6.5 gives:

$$L = \log p(\mathbf{X} | \Delta, \pi) = \sum_{n=1}^N \log \sum_{k=1}^K \pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \quad (6.6)$$

The summation inside the logarithm makes finding the optimal parameter values, $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi$, challenging. The EM algorithm overcomes this problem by deriving a *lower bound* on this likelihood (a function of \mathbf{X}, Δ and π that is *always* lower than or equal to L). Instead of maximising L directly, we instead maximise the lower bound.

To obtain a lower bound on L we can use the following relationship between logs of expectations and expectations of logs, which is known as Jensen's inequality:

$$\log \mathbf{E}_{p(z)} \{f(z)\} \geq \mathbf{E}_{p(z)} \{\log f(z)\}, \quad (6.7)$$

i.e. the log of the expected value of $f(z)$ is always greater than or equal to the expected value of $\log f(z)$.

In order to use Jensen's inequality to lower bound our likelihood, we need to make the right hand side of Equation 6.6 look like the log of an expectation. To do this, we multiply and divide the expression inside the summation over k by a new variable, q_{nk} :

$$L = \sum_{n=1}^N \log \sum_{k=1}^K \pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \frac{q_{nk}}{q_{nk}}.$$

If we restrict q_{nk} to be positive and satisfy the summation constraint $\sum_{k=1}^K q_{nk} = 1$ (i.e., q_{nk} is some probability distribution over the K components for the n th object), we can re-write this as an expectation with respect to q_{nk} :

$$L = \sum_{n=1}^N \log \sum_{k=1}^K q_{nk} \frac{\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{q_{nk}}$$

$$= \sum_{n=1}^N \log \mathbf{E}_{q_{nk}} \left\{ \frac{\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{q_{nk}} \right\}.$$

Applying Jensen's inequality, we can lower bound this expression:

$$L = \sum_{n=1}^N \log \mathbf{E}_{q_{nk}} \left\{ \frac{\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{q_{nk}} \right\} \geq \sum_{n=1}^N \mathbf{E}_{q_{nk}} \left\{ \log \frac{\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{q_{nk}} \right\}.$$

The right hand side of this expression is the bound (we will denote it \mathcal{B}) that we shall optimise. Expanding the expression gives us something more manageable:

$$\begin{aligned} \mathcal{B} &= \sum_{n=1}^N \mathbf{E}_{q_{nk}} \left\{ \log \frac{\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{q_{nk}} \right\} \\ &= \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log \left(\frac{\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{q_{nk}} \right) \\ &= \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log \pi_k + \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) - \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log q_{nk}. \end{aligned} \tag{6.8}$$

Values of $q_{nk}, \pi, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ that correspond to a local maxima of this bound will also correspond to a local maxima of the log likelihood, L .

As we mentioned earlier, the EM algorithm produces an iterative procedure. This will involve updates for each of the quantities in the model that we will repeat until convergence. To obtain each update we will take the partial derivative of the bound \mathcal{B} with respect to the relevant parameter, set it to zero and solve. We will now do this for each parameter in turn.

6.3.3.1 Updating π_k

Only the first term of \mathcal{B} contains π_k (the partial derivative of all other terms with respect to π_k is zero). π_k is a probability and therefore $\sum_k \pi_k = 1$. Hence, the optimisation with respect to π_k is constrained. As we saw for the SVM in Section 5.3.2.2, we can use Lagrangian terms to incorporate constraints into our objective function (in this case, \mathcal{B}). The relevant part of \mathcal{B} along with the suitable Lagrangian term (and associated Lagrange multiplier λ ; see Comment 5.1) is:

$$\mathcal{B} = \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log \pi_k - \lambda \left(\sum_{k=1}^K \pi_k - 1 \right) + \dots$$

Taking partial derivatives with respect to π_k , setting to zero and re-arranging results in:

$$\begin{aligned} \frac{\partial \mathcal{B}}{\partial \pi_k} &= \frac{\sum_{n=1}^N q_{nk}}{\pi_k} - \lambda = 0 \\ \sum_{n=1}^N q_{nk} &= \lambda \pi_k. \end{aligned} \tag{6.9}$$

The final step requires computing λ . To do this, we sum both sides over k :

$$\begin{aligned} \sum_{k=1}^K \sum_{n=1}^N q_{nk} &= \lambda \sum_{k=1}^K \pi_k \\ \sum_{n=1}^N 1 &= \lambda \\ \lambda &= N \end{aligned}$$

where we have used the fact that $\sum_{k=1}^K q_{nk} = 1$ and $\sum_{k=1}^K \pi_k = 1$ by definition. Substituting $\lambda = N$ into Equation 6.9 gives us the expression for π_k :

$$\pi_k = \frac{1}{N} \sum_{n=1}^N q_{nk}.$$

We will discuss the intuition behind this and the other expression in Section 6.3.3.5.

6.3.3.2 Updating μ_k

Next, we will look at μ_k . Only the second term of \mathcal{B} includes μ_k . If we explicitly write $p(\mathbf{x}_n | \mu_k, \Sigma_k)$ as a multivariate Gaussian (e.g. Equation 2.28) and expand we obtain:

$$\begin{aligned} \mathcal{B} &= \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log \left(\frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} \exp \left(-\frac{1}{2} (\mathbf{x}_n - \mu_k)^\top \Sigma_k^{-1} (\mathbf{x}_n - \mu_k) \right) \right) + \dots \\ &= -\frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log \left((2\pi)^d |\Sigma_k| \right) - \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K q_{nk} (\mathbf{x}_n - \mu_k)^\top \Sigma_k^{-1} (\mathbf{x}_n - \mu_k) + \dots \end{aligned}$$

The first term does not involve μ_k and can therefore be ignored. Making use of the following identity (see Table 1.4):

$$f(\mathbf{w}) = \mathbf{w}^\top \mathbf{C} \mathbf{w}, \quad \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} = 2\mathbf{C}\mathbf{w},$$

and using the chain rule, we can take the partial derivative of \mathcal{B} with respect to μ_k :

$$\begin{aligned} \frac{\partial \mathcal{B}}{\partial \mu_k} &= -\frac{1}{2} \sum_{n=1}^N q_{nk} \times \frac{\partial (\mathbf{x}_n - \mu_k)^\top \Sigma_k^{-1} (\mathbf{x}_n - \mu_k)}{\partial (\mathbf{x}_n - \mu_k)} \times \frac{\partial (\mathbf{x}_n - \mu_k)}{\partial \mu_k} \\ &= \sum_{n=1}^N q_{nk} \Sigma_k^{-1} (\mathbf{x}_n - \mu_k). \end{aligned}$$

Equating to zero and re-arranging gives us an expression for μ_k :

$$\begin{aligned}
 \sum_{n=1}^N q_{nk} \Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) &= 0 \\
 \sum_{n=1}^N q_{nk} \Sigma_k^{-1} \mathbf{x}_n &= \sum_{n=1}^N q_{nk} \Sigma_k^{-1} \boldsymbol{\mu}_k \\
 \sum_{n=1}^N q_{nk} \mathbf{x}_n &= \boldsymbol{\mu}_k \sum_{n=1}^N q_{nk} \\
 \boldsymbol{\mu}_k &= \frac{\sum_{n=1}^N q_{nk} \mathbf{x}_n}{\sum_{n=1}^N q_{nk}}. \tag{6.10}
 \end{aligned}$$

6.3.3.3 Updating Σ_k

Third, we will look at Σ_k . As with $\boldsymbol{\mu}_k$, we only need to look at the $p(\mathbf{x}_n | \boldsymbol{\mu}_k, \Sigma_k)$ term of \mathcal{B} . We have already seen this term expanded to:

$$\mathcal{B} = -\frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log \left((2\pi)^d |\Sigma_k| \right) - \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K q_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)^\top \Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) + \dots$$

Ignoring the constant (2π) part of the first term, we are left with:

$$\mathcal{B} = -\frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log (|\Sigma_k|) - \frac{1}{2} \sum_{n=1}^N \sum_{k=1}^K q_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)^\top \Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) + \dots$$

To take partial derivatives with respect to the matrix Σ_k , we need two more useful identities. First:

$$\frac{\partial \log |\mathbf{C}|}{\partial \mathbf{C}} = (\mathbf{C}^\top)^{-1},$$

and

$$\frac{\partial \mathbf{a}^\top \mathbf{C}^{-1} \mathbf{b}}{\partial \mathbf{C}} = -(\mathbf{C}^\top)^{-1} \mathbf{a} \mathbf{b}^\top (\mathbf{C}^\top)^{-1}.$$

Using these two identities, we take partial derivatives with respect to Σ_k :

$$\frac{\partial \mathcal{B}}{\partial \Sigma_k} = -\frac{1}{2} \sum_{n=1}^N q_{nk} \Sigma_k^{-1} + \frac{1}{2} \sum_{n=1}^N q_{nk} \Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) (\mathbf{x}_n - \boldsymbol{\mu}_k)^\top \Sigma_k^{-1}.$$

Note that as it is a covariance matrix, Σ_k is symmetric and therefore $\Sigma_k^\top = \Sigma_k$. Equating this expression to zero and re-arranging gives:

$$-\frac{1}{2} \sum_{n=1}^N q_{nk} \Sigma_k^{-1} + \frac{1}{2} \sum_{n=1}^N q_{nk} \Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) (\mathbf{x}_n - \boldsymbol{\mu}_k)^\top \Sigma_k^{-1} = 0$$

$$\frac{1}{2} \sum_{n=1}^N q_{nk} \Sigma_k^{-1} = \frac{1}{2} \sum_{n=1}^N q_{nk} \Sigma_k^{-1} (\mathbf{x}_n - \boldsymbol{\mu}_k) (\mathbf{x}_n - \boldsymbol{\mu}_k)^\top \Sigma_k^{-1}$$

Pre- and post-multiplying both sides by Σ_k allows us to cancel all of the Σ_k^{-1} :

$$\begin{aligned} \Sigma_k \sum_{n=1}^N q_{nk} \Sigma_k^{-1} \Sigma_k &= \Sigma_k \Sigma_k^{-1} \sum_{n=1}^N q_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k) (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \Sigma_k^{-1} \Sigma_k \\ \Sigma_k \sum_{n=1}^N q_{nk} &= \sum_{n=1}^N q_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k) (\mathbf{x}_n - \boldsymbol{\mu}_k)^T \\ \Sigma_k &= \frac{\sum_{n=1}^N q_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k) (\mathbf{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^N q_{nk}}. \end{aligned} \quad (6.11)$$

6.3.3.4 Updating q_{nk}

Finally, we need to derive an update for q_{nk} . This appears in all three terms in \mathcal{B} . In addition, it is subject to the constraint $\sum_{k=1}^K q_{nk} = 1$ and so, like the update for π_k , we will need a Lagrangian term (see Comment 5.1). The bound, complete with Lagrangian term is:

$$\begin{aligned} \mathcal{B} &= \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log \pi_k + \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log p(\mathbf{x}_n | \boldsymbol{\mu}_k, \Sigma_K) - \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log q_{nk} \\ &\quad - \lambda \left(\sum_{k=1}^K q_{nk} - 1 \right). \end{aligned}$$

Taking partial derivatives with respect to q_{nk} gives:

$$\frac{\partial \mathcal{B}}{\partial q_{nk}} = \log \pi_k + \log p(\mathbf{x}_n | \boldsymbol{\mu}_k, \Sigma_k) - (1 + \log q_{nk}) - \lambda,$$

Comment 6.1 – The product rule of differentiation: The product rule is used when we need to differentiate a product of two functions of the same variable with respect to the variable. For example, if

$$f(a) = g(a)h(a),$$

the product rule states that

$$\frac{\partial f(a)}{\partial a} = g(a) \frac{\partial h(a)}{\partial a} + \frac{\partial g(a)}{\partial a} h(a).$$

For example, to differentiate $a \log a$ with respect to a , this gives

$$a \times \frac{1}{a} + 1 \times \log(a) = 1 + \log(a).$$

where we have used the product rule (see Comment 6.1) to differentiate the $q_{nk} \log q_{nk}$ term. Setting to zero, re-arranging and exponentiating gives us an expression for q_{nk} :

$$\begin{aligned} 1 + \log q_{nk} + \lambda &= \log \pi_k + \log p(\mathbf{x}_n | \boldsymbol{\mu}_k, \Sigma_k) \\ \exp(\log q_{nk} + (\lambda + 1)) &= \exp(\log \pi_k + \log p(\mathbf{x}_n | \boldsymbol{\mu}_k, \Sigma_k)) \\ q_{nk} \exp(\lambda + 1) &= \pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \Sigma_k). \end{aligned} \quad (6.12)$$

As with the update for π_k , to find the constant term (in this case, $\exp(\lambda + 1)$), we sum both sides over k :

$$\begin{aligned} \exp(\lambda + 1) \sum_{k=1}^K q_{nk} &= \sum_{k=1}^K \pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \\ \exp(\lambda + 1) &= \sum_{k=1}^K \pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \end{aligned} \quad (6.13)$$

Substituting Equation 6.13 into Equation 6.12 gives us our expression for q_{nk} :

$$q_{nk} = \frac{\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j p(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \quad (6.14)$$

6.3.3.5 Some intuition

The four update equations are:

$$\pi_k = \frac{1}{N} \sum_{n=1}^N q_{nk} \quad (6.15)$$

$$\boldsymbol{\mu}_k = \frac{\sum_{n=1}^N q_{nk} \mathbf{x}_n}{\sum_{n=1}^N q_{nk}} \quad (6.16)$$

$$\boldsymbol{\Sigma}_k = \frac{\sum_{n=1}^N q_{nk} (\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^T}{\sum_{n=1}^N q_{nk}} \quad (6.17)$$

$$q_{nk} = \frac{\pi_k p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j p(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \quad (6.18)$$

The first three expressions rely heavily on q_{nk} : π_k is the mean value of q_{nk} for a particular k , $\boldsymbol{\mu}_k$ is the average of the data objects weighted by q_{nk} and $\boldsymbol{\Sigma}_k$ is a weighted covariance. What does q_{nk} represent? Equation 6.18 can provide some intuition. At first glance, it looks a lot like Bayes' rule with a prior π_k , a likelihood $p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ and a normalising constant obtained by averaging over the k components. In fact, it could be interpreted as computing a posterior probability of object n belonging to class k (it looks very similar to the Bayesian classification version of Bayes' rule given in Equation 5.2). In particular,

$$p(z_{nk} = 1 | \mathbf{x}_n, \boldsymbol{\pi}, \Delta) = \frac{p(z_{nk} = 1 | \pi_k) p(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K p(z_{nj} = 1 | \pi_j) p(\mathbf{x}_n | \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} = q_{nk}. \quad (6.19)$$

For particular values of the model parameters $\boldsymbol{\pi}, \boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_1, \dots, \boldsymbol{\Sigma}_K$, q_{nk} tells us the posterior probability of object n belonging to component k . In light of this, Equations 6.15, 6.16 and 6.17 make sense. Equation 6.15 is the average of all posterior probabilities of belonging to class k or, in other words, the expected proportion of the data belonging to class k . Imagine a scenario where the components are so distinct that the posterior probabilities are all either 1 or 0. In this case, π_k is just the proportion of the data assigned to component k . $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the mean and variance of the data objects where each object is weighted by the posterior probability of belonging to component k – objects that have a high probability of

belonging to component k have a strong influence on the mean and variance of component k .

Keeping the previous discussion in mind, we can split the four updates into two sets. The first set consists of updating our current estimates of the model components π_k , μ_k and Σ_k with q_{nk} , the assignment probabilities, fixed. In the second step, we update the assignments q_{nk} to reflect the new values of the model parameters. This procedure is very similar to the K -means algorithm introduced earlier. Updating q_{nk} is analogous to updating z_{nk} in K -means and updating μ_k , Σ_k , π is analogous to updating μ_k in K -means. The key difference is that we are computing posterior probabilities of cluster memberships rather than making hard assignments and the fact that we are inferring the component covariances (although this is a design choice – we could simply make the assumption that $\Sigma_k = \mathbf{I}$). Replacing q_{nk} with z_{nk} in Equation 6.16 gives us exactly the mean update from K -means, Equation 6.1.

The four update equations make up an example of the EM algorithm. The first three updates, π_k , μ_k , Σ_k , make up the so-called ‘M’ (maximisation) step where the bound is maximised conditioned on the values of q_{nk} . The update of q_{nk} is known as the ‘E’ (expectation) step as it actually involves computing the expected value of the unknown assignments, z_{nk} , although we have not derived them in this way. The reader is encouraged to explore other uses of EM in the literature to see alternative derivations.

6.3.4 Example

The synthetic data we have used throughout this chapter is reproduced in Figure 6.8 and we will use it to illustrate the operation of the EM algorithm we derived in the previous section (MATLAB script: `gmix.m`). Much like K -means, we must specify the number of components we expect to see *a priori* and in this case we will use $K = 3$. Unlike K -means, there is a useful measure that we can use to infer this from the data and we shall come to this in due course.

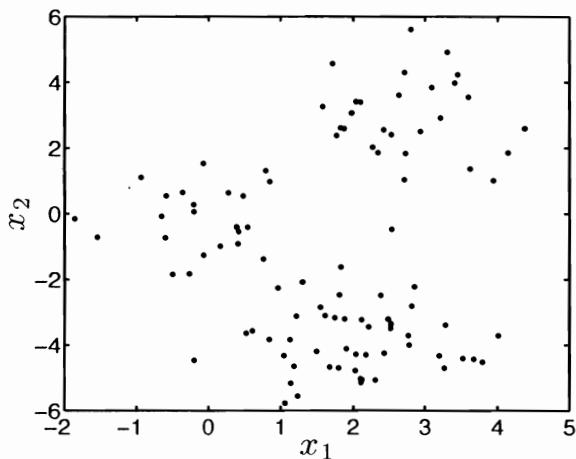


FIGURE 6.8: The synthetic clustering data encountered earlier in the chapter.

Before we can start performing the updates provided in Equations 6.15 to 6.18 we need to initialise some of the parameters. We do this by randomly choosing the means and covariances of the three mixture components. The three resulting Gaussian pdfs are plotted in Figure 6.9(a). In addition, before we can compute q_{nk} using Equation 6.18, we need to initialise π_k . We do this by assuming a uniform prior distribution over the three components: $\pi_k = 1/K$. We now have all we need to compute q_{nk} through Equation 6.18 (the ‘E’ step) and then subsequently update π_k, μ_k, Σ_k using Equations 6.15, 6.16 and 6.17, respectively (the ‘M’ step). The resulting Gaussians can be seen in Figure 6.9(b). We notice that after only one iteration, the Gaussians are beginning to reflect the cluster structure in the data. After this first large step, progress becomes a little slower. In Figure 6.9(c) we see the Gaussians after five EM iterations – the top right component has become more distinct (completely separated from the other two) whilst the other two are gradually diverging. Two iterations later, these two components have moved apart, as can be seen in Figure 6.9(d) and from here it is only a few iterations until the algorithm converges – updating q_{nk} and the model parameters causes no change in their values. The converged solution can be seen in Figure 6.9(e) where the distinct cluster structure is clearly visible. Finally, in Figure 6.9(f), we can see the evolution of the bound \mathcal{B} and the log likelihood L . Both increase as required.

Often we are not interested in the Gaussians themselves but the assignments of objects to components – the *clustering*. These are provided by the values of q_{nk} – the posterior probability of objects belonging to components. If we want a single assignment of objects to components, we can assign each object to the component for which it has the highest posterior probability. It is worth pointing out that hard assignment like this might not always be sensible. Consider an object (object n) that has the following values of q_{nk} at convergence:

$$q_{n1} = 0.53, \quad q_{n2} = 0.45, \quad q_{n3} = 0.02$$

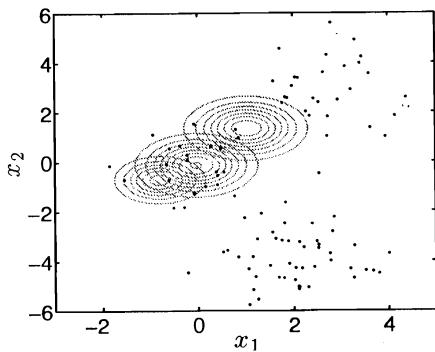
If we must assign it to a particular component, number 1 is most appropriate but in doing so, we are throwing away useful information about the relationship object n has with component 2.

At this point, you could be forgiven for wondering why we have bothered deriving this rather complex way of doing something that K -means seemed to do in a much more straightforward manner – the clusterings produced by K -means and the mixture model are almost identical and K -means can be kernelised. In the next two sections, we will see that mixture models have some key advantages over K -means due, predominantly, to their probabilistic nature.

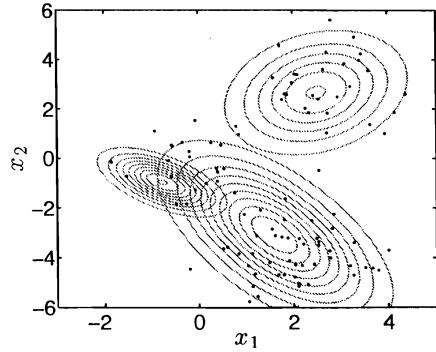
Before we move on we will revisit the data that motivated our move from K -means to the mixture model (Figure 6.10(a)). Using $K = 2$ and the update equations derived in this section, we can apply a mixture model to this data and the result is seen in Figure 6.10(b). It is clear that the mixture model has successfully extracted the interesting cluster structure.

6.3.5 EM finds local optima

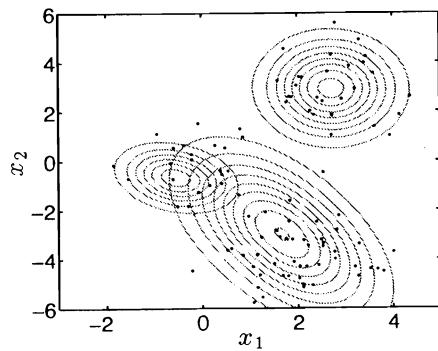
As with K -means, the solution to which the EM algorithm will converge will depend upon the specific initialisation. It is only guaranteed to reach a local maximum of the likelihood and not necessarily the global maximum. In fact, there will always



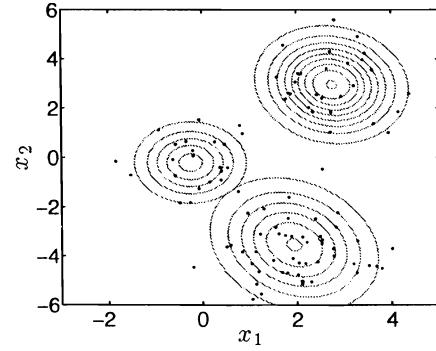
(a) The three randomly initialised Gaussian mixture components



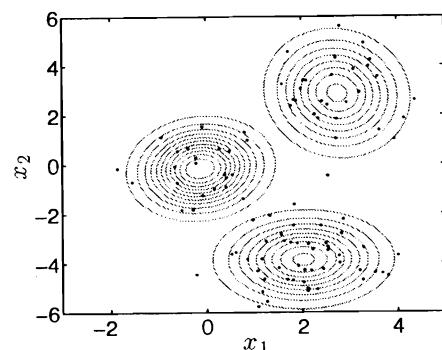
(b) The three components after one iteration of the EM algorithm



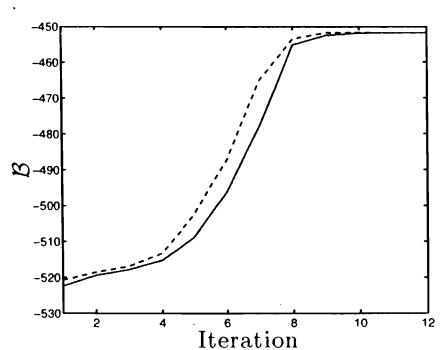
(c) The three components after five iterations of the EM algorithm



(d) The three components after seven iterations of the EM algorithm



(e) The three components at convergence of the EM algorithm



(f) The evolution of the bound \mathcal{B} (solid line, Equation 6.8) and log likelihood L (dashed line, Equation 6.5)

FIGURE 6.9: Example of the Gaussian mixture algorithm in action.

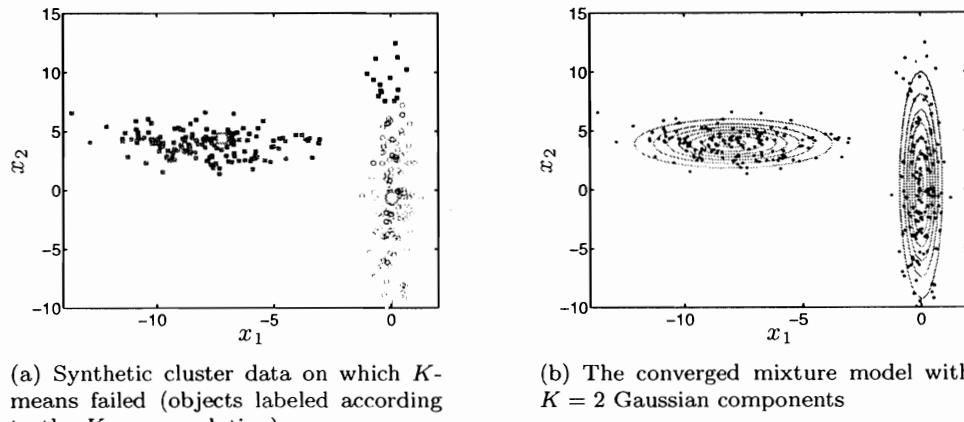


FIGURE 6.10: The data on which K -means failed and the successful mixture model solution.

be more than one global maximum as redefining the component labels must result in the same likelihood (renaming μ_k, Σ_k as, say, μ_j, Σ_j). As for K -means, we cannot solve this problem analytically and have to resort to running the algorithm from many starting points. We can use the likelihood (Equation 6.5) to evaluate which of the converged solutions is better (as we did with D , Equation 6.2, in K -means).

6.3.6 Choosing the number of components

As with K -means, we have to specify the number of clusters by choosing the number of components. We saw earlier that this choice, within the context of K -means, was nontrivial – the only quantity at our disposal was the total distance between objects and their cluster centres and this kept decreasing as the number of components increased. The same problem eliminates the use of the log likelihood L (and the bound B) for the mixture model. Figure 6.11(a) shows how the log likelihood L increases with the number of mixture components K . To understand why this is the case, consider the clustering with $K = 10$ shown in Figure 6.11(b). Each of the three original components (Figure 6.9(e)) is now represented by several smaller components. Imagine these Gaussians plotted in 3D (as we did in Figure 2.8). Because their volume must equal 1 (they are densities), the smaller the area they occupy in the input space (i.e. the smaller the ellipses in Figure 6.11(b)) the higher they must be. The likelihood for the dataset, which is the product of the heights at each of the data objects (or the sum of the log of the heights for the log likelihood, L), will be higher. As we add more and more components, the area they need to take up decreases and the likelihood increases still further.

Fortunately, we can overcome this problem by computing the likelihood on a validation set using, for example, cross-validation. The results of performing a 10-fold cross-validation can be seen in Figure 6.12 (MATLAB script: `gmixcv.m`). The line and bars show the mean and standard deviation of the likelihood on the held-out data. The results are not conclusive in the sense that they strongly suggest a

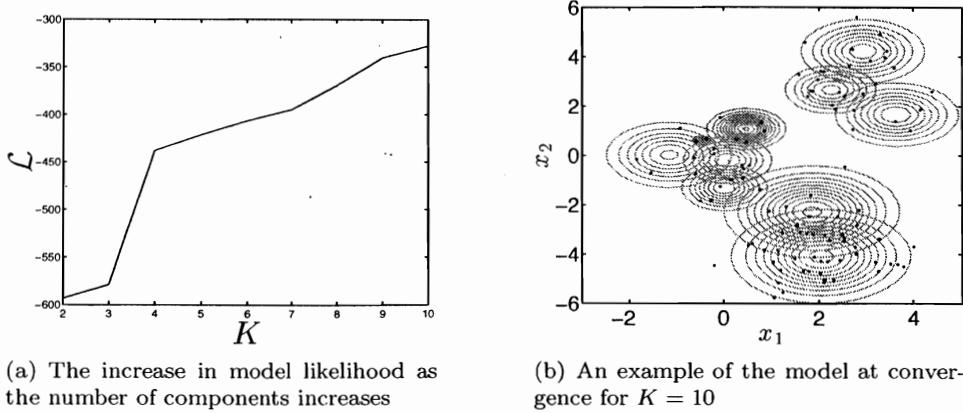


FIGURE 6.11: The log likelihood L increases with the number of components, K .

particular number of components but they do give us an indication that the likely number lies somewhere between, say, three and eight. In our experience, this is about as much precision as one can expect with this quantity of data but it offers a considerable advantage over K -means, where it is hard to get any indication of how many clusters are present.

Of course, if the clustering is just one step in a larger analysis, we can use some other figure of merit (classification accuracy, for example) to choose the number of clusters. In addition, recently developed nonparametric techniques enable the number of components to be sampled within a Markov-chain Monte Carlo scheme (like the Metropolis–Hastings method described in Chapter 4). Such techniques are

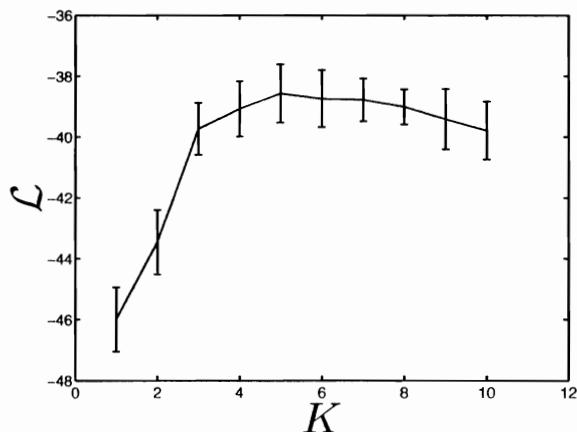


FIGURE 6.12: Result of 10-fold cross-validation for a Gaussian mixture model on the data shown in Figure 6.8.

beyond the scope of this book but the interested reader is referred to the growing body of literature in this area given in the reading list at the end of the chapter.

6.3.7 Other forms of mixture components

The second advantage of mixture models over K -means is their immense flexibility. In particular, $p(\mathbf{x}_n | \dots)$ can take the form of any probability density. In the previous example we used (and derived updates for) a Gaussian. Many other forms of components are regularly used. We will demonstrate this with a binary dataset but before we do that, it is worth spending a little more time with the Gaussian as it often appears in slightly different forms.

In particular, it is often necessary to put restrictions on the mixture component covariance matrix as there is not enough data to reliably estimate a full covariance matrix. For example, if we had 10-dimensional data rather than 2-dimensional data we would need a lot more data to be able to estimate the 55 parameters required in each covariance matrix. To overcome this, it is common to assume that the covariance matrix has only diagonal elements. You should recall from Chapter 2 that this is equivalent to assuming that the dimensions are independent. The only difference to the EM algorithm is in the update for Σ_k , which can now be separated into an update for the variance of each dimension d , σ_{kd}^2 (see Exercise EX 6.1). An even more extreme case is when the covariance is assumed to be *isotropic* (diagonal, with the same value on each diagonal element), $\Sigma_k = \sigma_k^2 \mathbf{I}$. Once again, the only difference to the algorithm is in the update for Σ_k (see Exercise EX 6.2).

We will now briefly describe a mixture model for binary data. Each data object, \mathbf{x}_n , is a collection of D binary values. For example, in $D = 10$ dimensions, an example data object might be:

$$\mathbf{x}_n = [0, 1, 0, 1, 1, 1, 0, 0, 0, 1].$$

An example 10-dimensional dataset is shown in Figure 6.13. Each row represents one data object. Assuming that the dimensions are independent within a particular component, $p(\mathbf{x}_n | \dots)$ could be represented as a product of Bernoulli distributions (see Section 2.3.1):

$$p(\mathbf{x}_n | \mathbf{p}_k) = \prod_{d=1}^D p_{kd}^{x_{nd}} (1 - p_{kd})^{1-x_{nd}}, \quad (6.20)$$

where $\mathbf{p}_k = [p_{k1}, \dots, p_{kD}]^\top$ is a vector of dimension-specific probabilities for the k th component (i.e. $0 \leq p_{kd} \leq 1$). There will be two differences to our EM algorithm. First, when updating q_{nk} Equation 6.18 becomes:

$$q_{nk} = \frac{\pi_k p(\mathbf{x}_n | \mathbf{p}_k)}{\sum_{j=1}^K \pi_j p(\mathbf{x}_n | \mathbf{p}_j)} \quad (6.21)$$

where $p(\mathbf{x}_n | \mathbf{p}_k)$ is given by Equation 6.20. Second, an update for \mathbf{p}_k will replace the updates for μ_k and Σ_k (Equations 6.16 and 6.17).

To derive this update, we can extract the data-dependent term from the bound

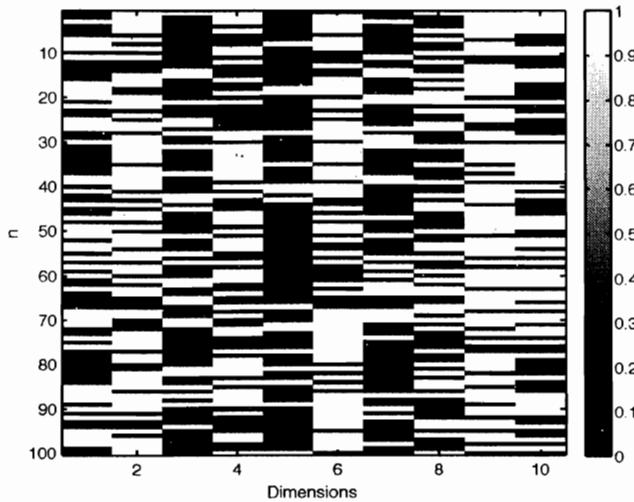


FIGURE 6.13: An example binary dataset with $N = 100$ objects and $D = 10$ dimensions. Each row represents one data object.

B (Equation 6.8). This term becomes:

$$\begin{aligned}\mathcal{B} &= \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log p(\mathbf{x}_n | \mathbf{p}_k) + \dots \\ &= \sum_{n=1}^N \sum_{k=1}^K q_{nk} \log \prod_{d=1}^D p_{kd}^{x_{nd}} (1 - p_{kd})^{1-x_{nd}} + \dots \\ &= \sum_{n=1}^N \sum_{k=1}^K q_{nk} \sum_{d=1}^D (x_{nd} \log p_{kd} + (1 - x_{nd}) \log(1 - p_{kd})) + \dots\end{aligned}$$

Keeping only p_{kd} terms results in:

$$\mathcal{B} = \sum_{n=1}^N q_{nk} (x_{nd} \log p_{kd} + (1 - x_{nd}) \log(1 - p_{kd})) + \dots$$

Taking partial derivatives with respect to p_{kd} gives us:

$$\frac{\partial \mathcal{B}}{\partial p_{kd}} = \sum_{n=1}^N q_{nk} \left(\frac{x_{nd}}{p_{kd}} - \frac{1 - x_{nd}}{1 - p_{kd}} \right).$$

Setting to zero and re-arranging gives us an update for p_{kd} ; setting to zero and solving gives (see Exercise EX 6.3):

$$p_{kd} = \frac{\sum_{n=1}^N q_{nk} x_{nd}}{\sum_{n=1}^N q_{nk}} \quad (6.22)$$

which is the weighted average of the d th data dimension, much like the update for μ_k

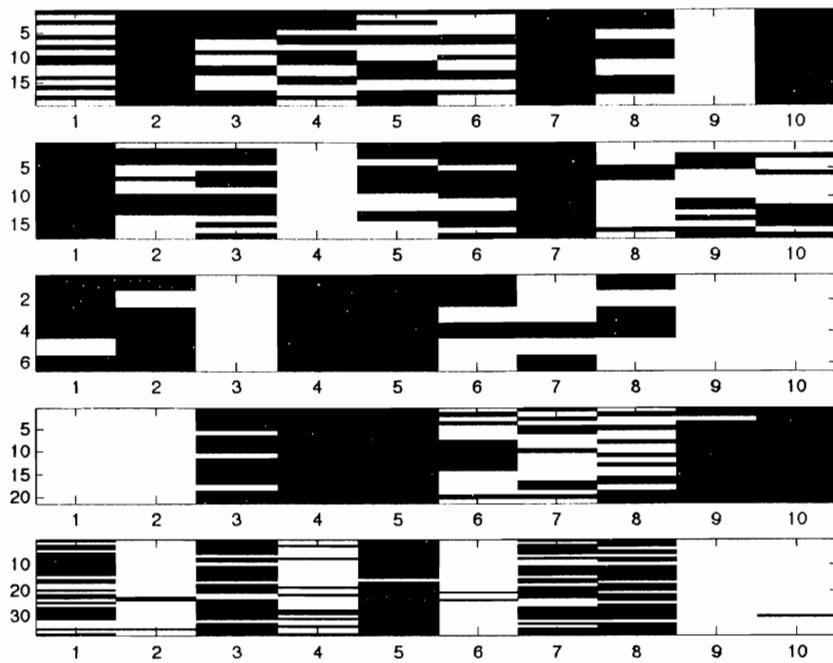


FIGURE 6.14: $K = 5$ clusters extracted from the data shown in Figure 6.13 using the mixture model with binary components.

in the Gaussian mixture (Equation 6.16). Our new EM algorithm involves iterating between updating q_{nk} from Equation 6.21 (the ‘E’ step) and updating \mathbf{p}_k and π_k using Equations 6.22 and 6.15, respectively. Just as with the Gaussian example, we need to initialise π_k and the component parameters which, once again, we do by setting $\pi_k = 1/K$ and randomly setting each p_{kd} to a value between 0 and 1. Using $K = 5$ and running the algorithm until convergence gives the clusters shown in Figure 6.14, where each block is one cluster (MATLAB script: `binmix.m`). We can see clear cluster structure – for example, in cluster 1 (top), all objects have a 1 in dimension 9 and an 0 in dimensions 10, 7 and 2.

In much the same way, we could derive an EM algorithm for many other component densities (see Exercise EX 6.6).

6.3.8 MAP estimates with EM

If we have a limited quantity of data, it might be useful to be able to regularise the parameter estimates obtained via EM. A straightforward way to do this is by multiplying the likelihood by suitable prior densities for the parameters and obtaining the MAP estimate (the value of the parameters that maximise the posterior; see Section 4.3). For example, in the binary example described above, we might use

independent beta priors (see Section 2.5.2) for each of the parameters p_{kd} :

$$p(\mathbf{p}_1, \dots, \mathbf{p}_K | \alpha, \beta) = \prod_{k=1}^K \prod_{d=1}^D \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} p_{kd}^{\alpha-1} (1 - p_{kd})^{\beta-1}.$$

This adds an extra p_{kd} term in the bound \mathcal{B} . The relevant terms are now:

$$\begin{aligned} \mathcal{B} &= (\alpha - 1) \log p_{kd} + (\beta - 1) \log(1 - p_{kd}) \\ &\quad + \sum_{n=1}^N q_{nk} (x_{nd} \log p_{kd} + (1 - x_{nd}) \log(1 - p_{kd})) + \dots \end{aligned}$$

Taking partial derivatives, setting to zero and solving in the normal way gives (see Exercise EX 6.4):

$$p_{kd} = \frac{\alpha - 1 + \sum_{n=1}^N q_{nk} x_{nd}}{\alpha + \beta - 2 + \sum_{n=1}^N q_{nk}}. \quad (6.23)$$

Note that $\alpha = \beta = 1$ recovers Equation 6.22. The regularising effect is clear. If $x_{nd} = 1$ for all n or $x_{nd} = 0$ for all n , Equation 6.22 would give $p_{kd} = 1$ and $p_{kd} = 0$. If a new data object did not have $x_{nd} = 1$ (or 0), it would have a likelihood of 0 of belonging to this cluster regardless of its values in the other $D - 1$ dimensions. Equation 6.23 overcomes this problem by effectively bounding p_{kd} to minimum and maximum values of:

$$\frac{\alpha - 1}{\alpha + \beta - 2},$$

and

$$\frac{\alpha - 1 + N}{\alpha + \beta - 2 + N}$$

respectively.

Map solutions can be obtained via EM for many prior and likelihood combinations. See Exercise EX 6.5 for another example.

6.3.9 Bayesian mixture models

Obtaining point estimates that correspond to the maximum likelihood or MAP solutions via EM is not the only way to cluster with a mixture model. In particular, it is possible to use a Markov chain Monte Carlo scheme to sample cluster assignments and the associated component parameters. This has various advantages, not least the fact that it is possible to get around the problem of fixing the number of components (as mentioned at the end of Section 6.3.6). The result is not a single clustering but many samples from a *distribution* over clusterings. In a pure modelling sense, this is a good thing – we are explicitly acknowledging the fact that there is uncertainty present in the number of clusters (components) and the associated assignments. In the presence of such uncertainty, insisting on a single clustering comes with all the pitfalls of any other point estimate. However, it is comes with problems of interpretability. For many applications it is hard to imagine how we can use a distribution over clusterings and often people resort to picking the sampled clustering that has maximum likelihood. MCMC approaches to mixture models are useful when the desired end result can be expressed as an expectation with respect to the distribution over clusterings. For example, if we want to compute the probability that two objects, \mathbf{x}_n and \mathbf{x}_m , are in the same cluster, we can simply count

the number of samples in which they are and divide this by the total number of samples. We could not calculate this quantity using a maximum likelihood or MAP solution via EM.

6.4 Summary

In this chapter we have provided an introduction to clustering through two families of algorithms: K -means (including kernel K -means) and mixture models. The simplicity of K -means (and the flexibility of kernel K -means) makes it a popular approach. The diverse range of different component models available means that mixture models (and subtle variants) are appearing in more and more applications. These techniques also have drawbacks – in particular, the K -means algorithm and the EM algorithm for mixture models are both only guaranteed to reach local optima. In other words, they will reach an optimum of their respective objective function but it will not necessarily be the global optima (the overall best solution). In both cases, the particular solution that is reached depends on the initialisation – different random values of μ_k and Σ_k will lead to different clusterings.

It is also important to remember that there are many other approaches available that we could never have covered in a single chapter and the reader is strongly encouraged to investigate other popular methods, for example: hierarchical clustering (widely used in computational biology), spectral clustering and functional clustering.

6.5 Exercises

- EX 6.1. Derive the EM update for the variance of the d th dimension and the k th component, σ_{kd}^2 , when the cluster components have a diagonal Gaussian likelihood:

$$p(\mathbf{x}_n | z_{nk} = 1, \mu_{k1}, \dots, \mu_{KD}, \sigma_{k1}^2, \dots, \sigma_{kD}^2) = \prod_{d=1}^D \mathcal{N}(\mu_{kd}, \sigma_{kd}^2)$$

- EX 6.2. Repeat Exercise EX 6.1 with isotropic Gaussian components:

$$p(\mathbf{x}_n | z_{nk} = 1, \boldsymbol{\mu}_k, \sigma_k^2) = \prod_{d=1}^D \mathcal{N}(\mu_{kd}, \sigma_k^2)$$

- EX 6.3. Derive the EM update expression for the parameter p_{kd} given in Equation 6.22.

- EX 6.4. Derive the MAP EM update expression for the parameter p_{kd} given in Equation 6.22. Assume a Beta prior with parameters α and β .

EX 6.5. Derive the MAP update for a mixture model with Gaussian components that are independent over the D dimensions:

$$p(\mathbf{x}_n | z_{nk} = 1, \mu_{k1}, \dots, \mu_{kD}, \sigma_{k1}^2, \dots, \sigma_{kD}^2) = \prod_{d=1}^D \mathcal{N}(\mu_{kd}, \sigma_{kd}^2),$$

assuming an independent Gaussian prior on each μ_{kd} with mean m and variance s^2 .

EX 6.6. Derive an EM algorithm for fitting a mixture of Poisson distributions. Assume you observe N integer counts, x_1, \dots, x_N . The likelihood is:

$$p(\mathbf{x}|\Delta) = \prod_{n=1}^N \sum_{k=1}^K \pi_k \frac{\lambda_k^{x_n} \exp\{-\lambda_k\}}{x_n!}$$

Further reading

- [1] David Blei, Andrew Ng, and Michael Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

This paper describes a complex mixture-type model for text data. The model is based on a more complex generative process than that described in this book. This particular model has proved to be very popular in the Machine Learning and Information Retrieval literature.

- [2] Igor Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. Model-based clustering and visualization of navigation patterns on a web site. *Data Mining and Knowledge Discovery*, pages 399–424, 2003.

Mixture models can be defined with any type of component density. Here, a model is developed that uses Markov chains as the component densities, parameterised by a set of transition probabilities. The model is used to analyse internet browsing behaviour.

- [3] Guojun Gan, Chaogun Ma, and Jianhong Wu. *Data Clustering: Theory, Algorithms, and Applications*. Society for Industrial Mathematics, 2007.

- [4] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31:651–666, 2010.

A recent tutorial paper providing an overview of the clustering problem and various clustering algorithms.

- [5] Anil K. Jain and R.C. Dubes. *Algorithms For Clustering Data*. Prentice Hall, 1988.

A clustering textbook that is now out of print but available free from the authors' website: http://www.cse.msu.edu/~jain/Clustering_Jain_Dubes.pdf

- [6] Anil K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Reviews*, pages 264–323, 1999.

A review of clustering techniques with discussions of applications in areas such as information retrieval, image segmentation and object recognition.

- [7] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, 2000.

A comprehensive description of statistical mixture models.

- [8] Carl Rasmussen. The infinite Gaussian mixture model. In *In Advances in Neural Information Processing Systems 12*, pages 554–560, 2000.

One of the first papers to describe the use of a Dirichlet Process for overcoming the problem of fixing the number of components in a mixture model.

Chapter 7

Principal Components Analysis and Latent Variable Models

In the previous chapter we introduced two unsupervised methods that could be used to perform clustering – the partitioning of data objects into a finite number of disjoint groups such that objects in the same group share some similarity. We now turn our attention to a second class of unsupervised methods that could broadly be classed as **projection** techniques.

We will see how these methods can be used to take datasets in very high dimensions and project them down to a smaller number of dimensions for, for example, visualisation and feature selection. These techniques fall within the larger scope of latent variable models and we shall use the visualisation example to help provide an introduction to this area.

7.1 The general problem

Our starting point is a dataset of N objects, \mathbf{y}_n . Each object is an M -dimensional vector. The number of parameters in many models increases with the number of dimensions, M . Therefore, if M is large, it can make parameter estimation challenging. Also, data in many dimensions is difficult to visualise. For these reasons, it is often useful to transform the M -dimensional representation \mathbf{y}_n into a D -dimensional representation \mathbf{x}_n . This process is known as projection. We are projecting M -dimensional data into D -dimensions in a manner that will hopefully preserve the properties of interest.

Figure 7.1 illustrates this problem in a more familiar setting. Both Figure 7.1(a) and Figure 7.1(b) show projections (shadows) of a three-dimensional object (a hand) onto a two-dimensional surface. In Figure 7.1(c) we see the idea of projection in a more mathematical setting. Here, some two-dimensional data (\mathbf{y}_n) has been projected onto one dimension. The projected dimension happens to be one of the original two dimensions, but this is not necessary. To draw an analogy with Figures 7.1(a) and 7.1(b), the original objects \mathbf{y}_n correspond to the hand, and \mathbf{x}_n to the shadow.

7.1.1 Variance as a proxy for interest

When performing the projection, we would like to retain as much of the interesting structure in our data as possible. What do we mean by *interesting*? Figures 7.1(a)

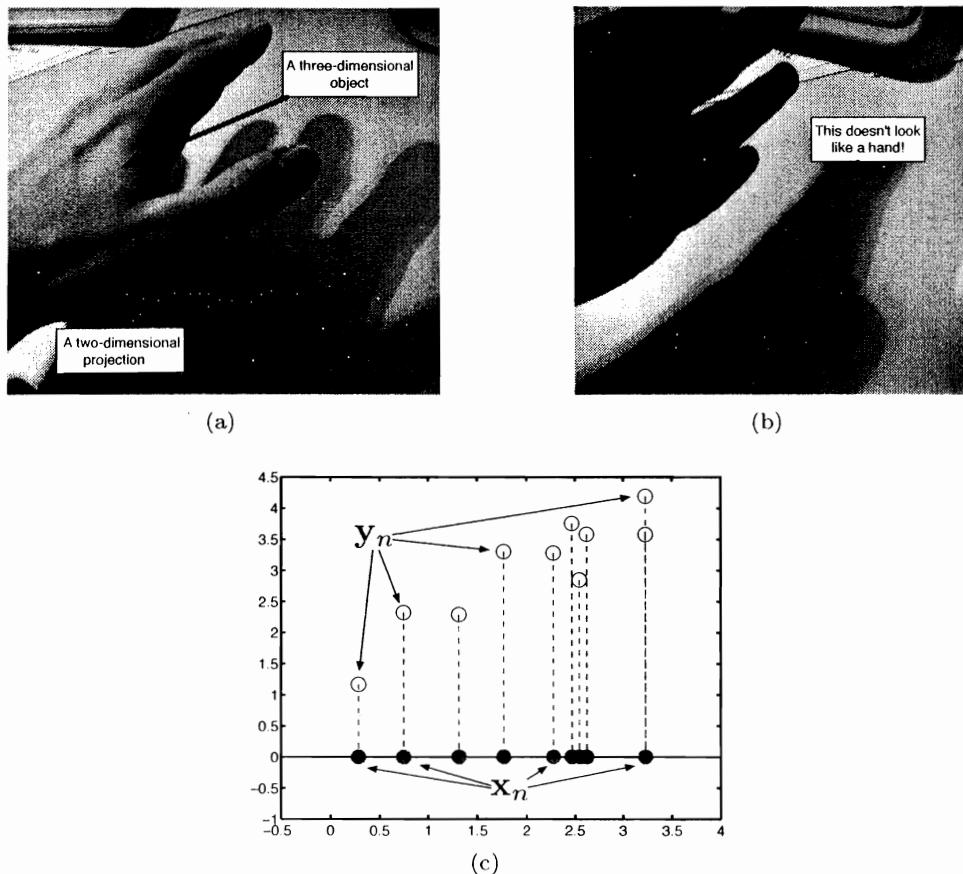


FIGURE 7.1: The idea of projection. (a) and (b) A hand (three-dimensions) being projected onto a table (two-dimensions) by a light. (c) Two-dimensional data y_n being projected into one dimension x_n . In this case, the projection is aligned with one of the original axes. This will not necessarily be the case.

and 7.1(b) are both projections of the same ‘data’. It is fairly clear in this case that the projection in Figure 7.1(a) maintains more of the characteristics of the original object (the hand) than Figure 7.1(b). In general, however, we will not be aware of the structure in the original representation and so cannot use this to optimise our projection.

In Figure 7.2(a) we can see a cloud of data points that have been generated from a single Gaussian distribution. The data has been projected onto two lines, A and B. Each line gives a different one-dimensional representation of the two-dimensional data. Note that, unlike in Figure 7.1(c), the lines do not correspond to either of the original dimensions. The representation in either one-dimensional space (the position on either line) is given by a linear combination of the original dimensions. In particular, $x_n = w_1 y_{n1} + w_2 y_{n2}$ (where $\mathbf{y}_n = [y_{n1}, y_{n2}]^\top$) or, in vector notation, $x_n = \mathbf{w}^\top \mathbf{y}_n$, where $\mathbf{w} = [w_1, w_2]^\top$.

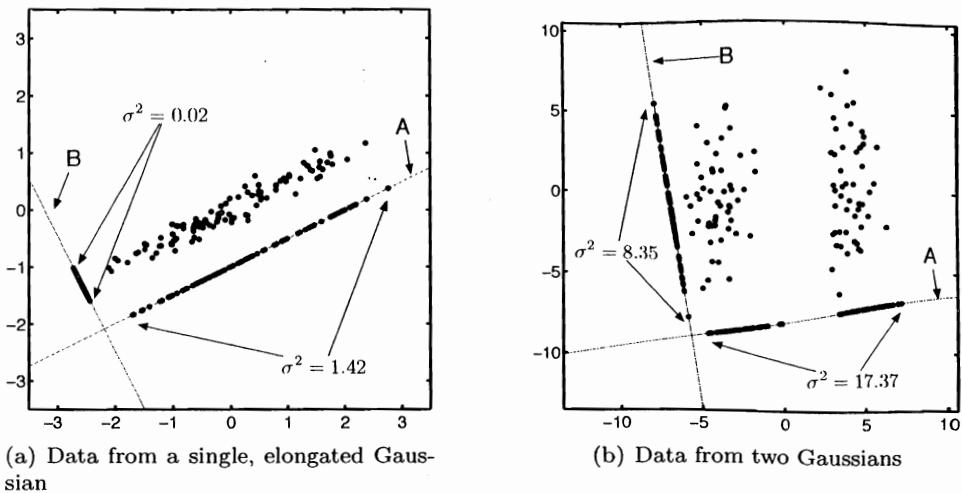


FIGURE 7.2: Examples showing the variance of different projections of two synthetic two-dimensional datasets. In both cases, two different one-dimensional projections are shown (labeled A and B) as well as the variance of the data in each projection (σ^2).

We can compute the variance of the data in each one-dimensional space as:

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \mu_x)^2,$$

and it is obvious that will be higher for projection A than for projection B. Given no other information, if we must project into one dimension, we should probably choose A. Put another way, if we are forced to throw one of A and B away, it feels safer to throw away the information present in B.

Figure 7.2(b) gives a more interesting example. We now have data that exhibits cluster structure. Projecting onto A preserves this cluster structure whilst projecting onto B does not. Cluster structure is an interesting property and so it looks like the projection onto A will be of more interest than that onto B. The variance of the data after projection A is more than double that for projection B. This is due to the cluster structure – all points are a large distance from the mean. If cluster structure is present in the data, using the projection with the highest variance is likely to preserve this structure.

For this reason, variance is seen as a good quantity to maximise when deciding on projection directions. It is the quantity that is maximised in the most popular projection technique, Principal Components Analysis.

7.2 Principal components analysis

Principal Components Analysis (PCA) is perhaps the most widely used statistical technique for projecting data into a lower-dimensional space. It is very popular within Machine Learning for visualisation and feature selection. PCA defines a linear projection: each of the projected dimensions is a linear combination of the original dimensions. That is, if we are projecting from M to D dimensions, PCA will define D vectors, \mathbf{w}_d , each of which is N -dimensional. The d th element of the projection, x_{nd} (where $\mathbf{x}_n = [x_{n1}, \dots, x_{nD}]^\top$), is computed as:

$$x_{nd} = \mathbf{w}_d^\top \mathbf{y}_n.$$

The learning task is therefore to choose how many dimensions we want to project into (D) and then pick a projection vector, \mathbf{w}_d , for each.

PCA uses variance in the projected space as the criteria to choose \mathbf{w}_d . In particular, \mathbf{w}_1 will be the projection that makes the variance in the x_{n1} as high as possible. The second projected dimension is also chosen to maximise the variance but \mathbf{w}_2 must be orthogonal to \mathbf{w}_1 ($\mathbf{w}_1^\top \mathbf{w}_2 = 0$). The third component, \mathbf{w}_3 , must maximise the variance and be orthogonal to both \mathbf{w}_1 and \mathbf{w}_2 , etc. In general:

$$\mathbf{w}_i^\top \mathbf{w}_j = 0, \quad \forall j \neq i.$$

This set of constraints tells us that if we set $D = M$, performing PCA amounts to rotating a rotation of the original data, without any loss of information.

In addition, PCA imposes the constraint that each \mathbf{w}_i must have a length of 1, $\mathbf{w}_i^\top \mathbf{w}_i = 1$. This does not restrict the technique, as it is only the direction of each \mathbf{w} that is important.

The problem that PCA solves in order to find the projections, $\mathbf{w}_1, \dots, \mathbf{w}_D$, can be derived in a number of ways. We are going to do it by deriving an expression for the variance of x_{n1} , as this is perhaps the most intuitive. The reader is encouraged to explore other approaches within the Statistics and Machine Learning literature.

Before we start the derivation it is useful to make the assumption that each of the original dimensions has zero mean:

$$\bar{\mathbf{y}} = \frac{1}{N} \sum_{n=1}^N \mathbf{y}_n = 0.$$

This can be enforced by subtracting the mean, $\bar{\mathbf{y}}$, from each \mathbf{y}_n .

We shall start by finding a projection into $D = 1$ dimension. In other words, we are only interested in finding one \mathbf{w} vector. In this case, the projection results in a scalar value, x_n , for each observation given by:

$$x_n = \mathbf{w}^\top \mathbf{y}_n.$$

The variance, σ_x^2 , is given by:

$$\sigma_x^2 = \frac{1}{N} \sum_{n=1}^N (x_n - \bar{x})^2 \tag{7.1}$$

We can simplify this expression due to our assumption that $\bar{\mathbf{y}} = 0$:

$$\begin{aligned}\bar{x} &= \frac{1}{N} \sum_{n=1}^N \mathbf{w}^\top \mathbf{y}_n \\ &\equiv \mathbf{w}^\top \left(\frac{1}{N} \sum_{n=1}^N \mathbf{y}_n \right) \\ &= \mathbf{w}^\top \bar{\mathbf{y}} = 0.\end{aligned}$$

Equation 7.1 becomes:

$$\sigma_x^2 = \frac{1}{N} \sum_{n=1}^N x_n^2.$$

Substituting the definition of x_n gives:

$$\begin{aligned}\sigma_x^2 &= \frac{1}{N} \sum_{n=1}^N (\mathbf{w}^\top \mathbf{y}_n)^2 \\ &= \frac{1}{N} \sum_{n=1}^N \mathbf{w}^\top \mathbf{y}_n \mathbf{y}_n^\top \mathbf{w} \\ &= \mathbf{w}^\top \left(\frac{1}{N} \sum_{n=1}^N \mathbf{y}_n \mathbf{y}_n^\top \right) \mathbf{w} \\ \sigma_x^2 &= \mathbf{w}^\top \mathbf{C} \mathbf{w},\end{aligned}\tag{7.2}$$

where \mathbf{C} is the sample covariance matrix, defined as:

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^N (\mathbf{y}_n - \bar{\mathbf{y}})(\mathbf{y}_n - \bar{\mathbf{y}})^\top,$$

but where $\bar{\mathbf{y}} = 0$ in our case. Note that this expression tells us that we didn't lose anything by transforming our data to force $\bar{\mathbf{y}}$ to be 0. \mathbf{C} would be the same whether we did this or not.

Our aim is to find the value of \mathbf{w} that maximises σ^2 and therefore also maximises $\mathbf{w}^\top \mathbf{C} \mathbf{w}$. We could keep increasing $\mathbf{w}^\top \mathbf{C} \mathbf{w}$ by increasing the value of the elements in \mathbf{w} and this is why \mathbf{w} is constrained to have a length of 1, $\mathbf{w}^\top \mathbf{w} = 1$. As with the constraints in the SVM optimisation in Chapter 5 and the EM derivation in Chapter 6, we can incorporate this constraint into our optimisation through the use of a Lagrangian term (see Comment 5.1). In particular, we wish to find the \mathbf{w} that maximises

$$L = \mathbf{w}^\top \mathbf{C} \mathbf{w} - \lambda(\mathbf{w}^\top \mathbf{w} - 1)$$

Taking partial derivatives with respect to \mathbf{w} , equating to zero and rearranging gives

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{w}} &= 2\mathbf{C}\mathbf{w} - \lambda\mathbf{w} = \mathbf{0} \\ \mathbf{C}\mathbf{w} &= \lambda\mathbf{w}\end{aligned}\tag{7.3}$$

(where we have incorporated the factor of 2 into the constant λ).

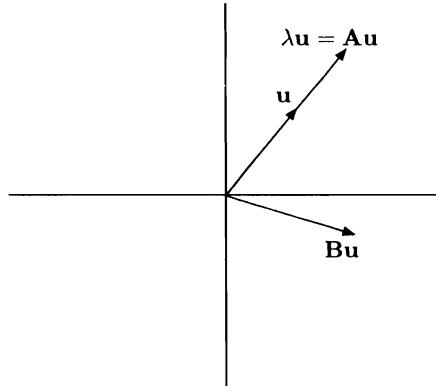
Comment 7.1 – Eigenvectors and eigen values: The eigenvector/eigenvalue equation for some square matrix \mathbf{A} is given as:

$$\lambda_i \mathbf{u}_i = \mathbf{A} \mathbf{u}_i. \quad (7.4)$$

The solutions to this equation are pairs of eigenvalues (λ_i) and eigenvectors (\mathbf{u}_i).

The figure on the right provides some intuition for this equation. Multiplying an M -dimensional vector \mathbf{u} by an $M \times M$ matrix \mathbf{B} results in another M -dimensional vector. Therefore, we can consider the matrix \mathbf{B} as defining a rotation of the vector \mathbf{u} . Different \mathbf{B} matrices will produce different rotations. The solutions to Equation 7.3 for a particular matrix \mathbf{A} are the vectors \mathbf{u} for which applying the rotation \mathbf{A} only results in a change in the length of \mathbf{u} . The magnitude of this change is given by the scalar λ .

In general, if the matrix \mathbf{A} has M rows and M columns, there are M eigenvector/eigenvalue pairs that solve Equation 7.4. The M eigenvectors will be orthogonal. We are not going to go into the detail of how to solve the eigenvalue/eigenvector equation. Routines for doing it are common, for example, the `eigs` function in Matlab.



Equation 7.3 is of a very common form, known as the eigenvector/eigenvalue equation (see Comment 7.1). Comparing Equation 7.3 with Equation 7.4, we can see that the projection \mathbf{w} that maximises the variance is one of the eigenvectors of the covariance matrix \mathbf{C} . However, there will be M of these; how do we know which one corresponds to the highest variance? Our expression for σ_x^2 is:

$$\sigma_x^2 = \mathbf{w}^\top \mathbf{C} \mathbf{w}.$$

Remember that $\mathbf{w}^\top \mathbf{w} = 1$, and we can therefore multiply the left hand side of this expression by $\mathbf{w}^\top \mathbf{w}$:

$$\sigma^2 \mathbf{w}^\top \mathbf{w} = \mathbf{w}^\top \mathbf{C} \mathbf{w}.$$

Removing a \mathbf{w}^\top from each side leaves us with something that looks very similar to 7.3:

$$\sigma^2 \mathbf{w} = \mathbf{C} \mathbf{w},$$

telling us that given an eigenvalue/eigenvector pair (λ, \mathbf{w}) , λ corresponds to the variance of the data in the projected space defined by \mathbf{w} . If we find the M eigenvector/eigenvalue pairs of the covariance matrix \mathbf{C} , the pair with the highest eigenvalue corresponds to the projection with maximal variance, \mathbf{w}_1 . The second highest eigenvalue corresponds to \mathbf{w}_2 , the third to \mathbf{w}_3 , etc.

To summarise, performing PCA on a set of data objects, $\mathbf{y}_1, \dots, \mathbf{y}_N$, requires

performing the following steps (the expressions in parentheses are the corresponding matrix operations if we define $\mathbf{Y} = [\mathbf{y}_1, \dots, \mathbf{y}_N]^\top$):

1. Transform the M -dimensional data to have zero mean by subtracting $\bar{\mathbf{y}}$ from each object where $\bar{\mathbf{y}} = \frac{1}{N} \sum_{n=1}^N \mathbf{y}_n$.
2. Compute the sample covariance matrix $\mathbf{C} = \frac{1}{N} \sum_{n=1}^N \mathbf{y}_n \mathbf{y}_n^\top$ (or $\mathbf{C} = \frac{1}{N} \mathbf{Y}^\top \mathbf{Y}$).
3. Find the M eigenvector/eigenvalue pairs of the covariance matrix. This can be done using, for example, the `eigs` function in Matlab.
4. Find the eigenvectors corresponding to the D highest eigenvalues, $\mathbf{w}_1, \dots, \mathbf{w}_D$.
5. Create the d th dimension for object n in the projection, $x_{nd} = \mathbf{w}_d^\top \mathbf{y}_n$ (or $\mathbf{X} = \mathbf{Y}\mathbf{W}$, where $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_D]$, i.e. the $M \times D$ matrix created by placing the D eigenvectors alongside one another and \mathbf{X} is the $N \times D$ matrix defined as $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^\top$).

To see a simple example of this, we can look back at Figure 7.2. In both plots, the directions we chose to project onto were the principal components. As we were working in two dimensions, there was a maximum of two components (a 2×2 covariance matrix has only two eigenvectors; it is impossible to have more than two orthogonal directions in two dimensions). When looking at these plots, remember that the procedure only defines the *direction* of the lines we have projected onto. We have moved the lines down a bit (for A) and left a bit (for B) to aid visualisation.

Figure 7.3 shows a more complex example (MATLAB script: `pcaexample.m`). Here, we have generated a dataset where each object is drawn from one of three clusters (see Figure 7.3(a)). We then make the data more complex by adding an additional five dimensions whose values are drawn from $\mathcal{N}(0, 1)$:

$$y_{nd} \sim \mathcal{N}(0, 1), \quad d = 3, \dots, 7, \quad n = 1, \dots, N.$$

In other words, there is structure in the first two dimensions and noise in the remainder. This might correspond to a real example where we have measured various

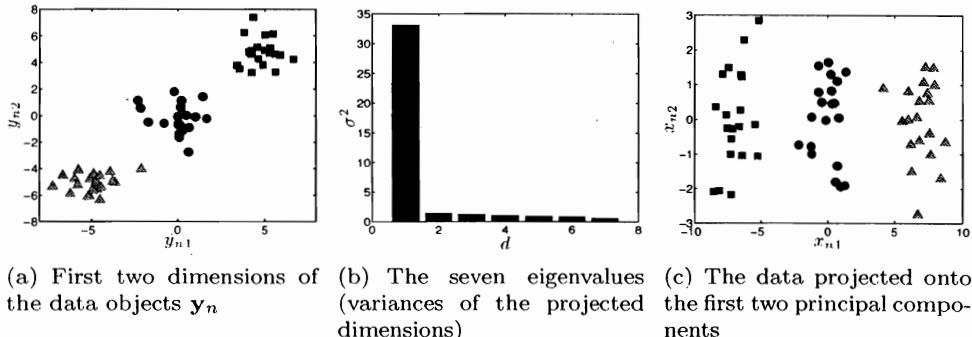


FIGURE 7.3: Synthetic PCA example where one projected dimension is all that is required. The data objects \mathbf{y}_n are seven-dimensional. The first two dimensions have the cluster structure shown in (a). The remainder are made up of values sampled from $\mathcal{N}(0, 1)$.

attributes of some objects, but do not know *a priori* which, if any, are interesting. After mean-centering the data, Figure 7.3(b) shows the values of the seven eigenvalues of the covariance matrix, $\mathbf{C} = \mathbf{Y}^T \mathbf{Y}$, ordered by magnitude. Recall that these values correspond to the variance in each of the D potential projection dimensions. We can see that the highest eigenvalue is far higher than any of the others – it looks like we could capture most of the variance in our original seven-dimensional space with just one projected dimension. This might seem strange given that our original cluster structure was in two dimensions. However, looking back at Figure 7.3(a), the cluster structure is really only one-dimensional, as the clusters all lie on the line $y_{n1} = y_{n2}$. One projected dimension will suffice. Plotting the data in the first two projected dimensions makes this clear (Figure 7.3(c)). The first projected dimension x_{n1} holds all the cluster structure.

This example shows us an important feature of PCA. The eigen-spectrum (magnitudes of the eigenvalues; Figure 7.3(b)) gives us some indication of how many *interesting* features there are in our data. In particular, Figure 7.3(b) tells us that we are unlikely to gain much by using two projected dimensions rather than one.

A second example can be seen in Figure 7.4 (MATLAB script: `pcaexample2.m`). Figure 7.4(a) shows a different cluster structure in the first two dimensions (the other five dimensions are constructed as in the previous example). We now have four clusters in an orientation that could not be explained using only one dimension. There is no single linear projection that would keep all of the clusters separate. We would therefore expect more than one large eigenvalue. Figure 7.4(b) shows that this is indeed the case – the first two eigenvalues are now both much higher than the remainder. The data projected onto these first two components can be seen in Figure 7.4(c) and it is clear that the cluster structure is preserved in the reduced space.

Before we proceed, it is worth reiterating a couple of important points from these examples. First, remember that in both examples we added five ‘random’ dimensions. Hence, the problem is not as trivial as Figures 7.3(a) and 7.4(a) suggest. Second, although we have labeled the data objects differently when plotting them (circles, squares, etc.) this information is not used by PCA – it is unsupervised. Finally, the fact that we happened to put the cluster structure into the first two dimensions is

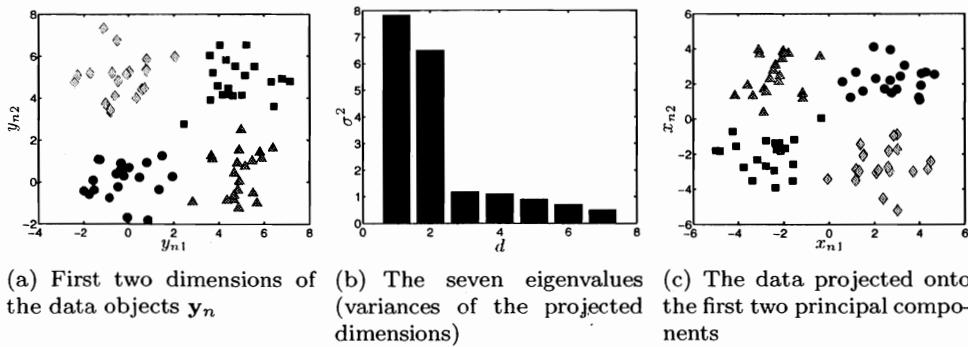


FIGURE 7.4: Synthetic PCA example where two projected dimensions are required.

irrelevant. If we shuffled up the columns of \mathbf{Y} (i.e. re-ordered the dimensions), the result would be exactly the same.

7.2.1 Choosing D

In the previous section we used the eigen-spectrum (and our knowledge of the data) to inform how many dimensions we should project into. In general, our choice of D will be very application specific. For example, if we are performing PCA as a visualisation step that lets us see our high-dimensional data, then we are restricted to the number of dimensions we can visualise in a practical way: normally a maximum of three.

For other uses, the eigen-spectrum provides some useful information but its interpretation is highly subjective (we will not always get plots that send as clear a message as Figures 7.3(b) and 7.4(b)). If PCA is being used as part of a larger system, it is important to consider more objective measures. For example, a common use of PCA is as a feature extraction technique prior to classification. If the data used in Figure 7.4 was actually a four class classification problem, consisting of a data matrix \mathbf{Y} and some labels \mathbf{t} , it might be sensible to perform the classification with the projected data \mathbf{X} rather than the original data \mathbf{Y} . In this case, D should really be chosen to be the value that gives the best classification performance via, perhaps, cross-validation.

7.2.2 Limitations of PCA

PCA has been successfully applied in many application areas but, like all models, it has clear limitations. In particular, it implicitly makes two assumptions about the data:

1. The data are real-valued.
2. There are no missing values in the data.

Many problems will give rise to data that fulfils both of these criteria but just as many will not. For example, missing values are a common occurrence in scientific data where quantities being measured go outside the dynamic range of the measurement equipment. Datasets of purchasing records (whether someone bought something or not) are binary and not real-valued. An obvious example that fails both criteria is a dataset of movie ratings. Imagine a matrix with a row for each viewer and a column for each movie. The value in the i th row and j th column is the rating that the i th viewer gives to the j th movie. Typically, this value will be an integer (0 to 5 stars; not real-valued) and it is highly unlikely that a single viewer would be able to watch and rate each movie, so many values will be missing.

In the remainder of this chapter, we shall use these limitations of classical PCA as an opportunity to introduce the general concept of *latent variable models* and how we can perform inference or learning within these models. We should stress that there are many types of and uses for latent variable models beyond the PCA-like domain – we’ve already seen one – mixture models – in Chapter 6. However, addressing the limitations of classical PCA gives us a good route in. We will also use this opportunity to introduce Variational Inference – a method of approximating an intractable posterior density that has become very popular throughout the

Machine Learning community in recent years due to its appealing combination of good empirical performance and low computational overhead.

7.3 Latent variable models

In many applications there will be characteristics of the objects of interest that are not provided in the data we are given. These *latent* variables (also known as hidden variables) can be placed into two categories:

1. Variables corresponding to a real feature of the object that have not been measured (e.g. maybe the technology to measure it is not available).
2. Abstract qualities that do not really exist but arise from our modelling assumptions and might be useful.

There are many examples of the former in the analysis of biological data. Consider a biological system consisting of three molecular species: A, B and C. A and C are easy to measure whilst B, for whatever reason, is not. In other words, A and C are observed whilst B is hidden. In this situation, we can explicitly model B as a hidden variable and use the data for A and C to learn something about B.

We are more interested in the second type of latent variable in this chapter. PCA is a good example – we observe some M -dimensional vectors, \mathbf{y}_n , and use these to construct a set of D -dimensional vectors, \mathbf{x}_n . The input vectors are likely to be measurements of something that actually exists in the world. The \mathbf{x}_n , however, are latent variables that we have created based on assumptions within our model – they do not necessarily exist ‘in the wild’. We created them in the hope that they might be more useful than the original variables for, for example, visualisation.

We shall return to PCA-like models in due course. First, it will be useful to take a model that we have already seen and place it into the latent variable framework.

7.3.1 Mixture models as latent variable models

In Section 6.3, we introduced mixture models as a powerful clustering technique. By means of an introduction, we described a procedure for generating data that involved, for each data object we wished to generate, choosing one of K possible components and then sampling the object from this component. We introduced a set of indicator variables, z_{nk} , where $z_{nk} = 1$ if the n th object was generated by the k th component. These indicator variables are latent variables – they do not necessarily exist in reality – that enabled us to build the mixture models. When deriving the algorithm for inferring the parameters of the mixture model, we did not explicitly use z_{nk} , although in Equation 6.19 we showed that the q_{nk} parameters could be interpreted as the posterior probability that object n was generated by component k : $p(z_{nk} = 1 | \mathbf{x}_n, \pi, \Delta)$. Our model definition implied the existence of a set of latent variables and we were able to learn something about the values they might have taken.

7.3.2 Summary

At the start of this chapter, we introduced PCA as a tool for *projection* of M -dimensional data into a D -dimensional space (where $D < M$). This can be useful for visualisation (when $D \leq 2$) or as a more generic unsupervised pre-processing tool before other analyses (classification, clustering, etc.). There are some drawbacks to PCA (ability to handle just real-valued data, inability to cope with missing data) which we will overcome in the remainder of this chapter. To set the scene for this, we have shown that PCA is one of a family of techniques known collectively as *latent variable models*, to which the mixture models introduced in Chapter 6 also belong. To be able to perform inference within a probabilistic PCA model, we will need to make some approximations. We have already seen ways of doing this in Chapter 4, and here we will introduce another technique, Variational Bayes.

7.4 Variational Bayes

Variational Bayes (VB) is an approximate inference technique that has become popular within Machine Learning due to its good empirical performance and relatively low computational cost. Like the Laplace approximation introduced in Section 4.4, it allows us to approximate an intractable posterior with something tractable. The parameters of the approximate posterior are optimised to make the approximation as close to the true posterior as possible.

Although VB is used to construct an approximate posterior, it is not its primary motivation. The posterior approximation appears when we attempt to maximise the log marginal likelihood.

Consider a very general case where we have some data \mathbf{Y} and a model that implies some parameters/latent variables $\boldsymbol{\theta}$. Note that we are lumping all model parameters and latent variables into the same symbol ($\boldsymbol{\theta}$). Within a Bayesian framework, the distinction between how we deal with latent variables and parameters becomes a little blurred: they are all things that we do not know so we treat them all as random variables. The marginal likelihood, $p(\mathbf{Y})$, is defined as:

$$p(\mathbf{Y}) = \int p(\mathbf{Y}, \boldsymbol{\theta}) d\boldsymbol{\theta}. \quad (7.5)$$

In this expression, we have omitted conditioning on all things that are constant. This could include the model type, prior parameters, hyper-parameters, etc. All of these things are model/problem specific, so we stick with this general expression but will be more precise when examining particular examples. An example of a more specific version of this equation is given in Section 3.4, where we first encountered marginal likelihoods.

Note that this expression is also commonly given with the joint density $p(\mathbf{Y}, \boldsymbol{\theta})$ broken up into its constituent parts:

$$p(\mathbf{Y}) = \int p(\mathbf{Y}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}.$$

The marginal likelihood computed in Equation 7.5 is therefore the result of averaging the likelihood ($p(\mathbf{Y}|\boldsymbol{\theta})$) over all values of the parameters (and latent variables), weighted by the prior, $p(\boldsymbol{\theta})$. This expression can be maximised with respect to all of the things on which the whole expression is conditioned (model structure, prior parameters). Unfortunately, maximising it is almost always very difficult due to the integral over the potentially high-dimensional parameter space. One way in which we can make progress is to lower bound the log marginal likelihood in a manner similar to that which we used in the EM derivation in Chapter 6. There, we used Jensen's inequality (see Equation 6.7), which we will repeat here:

$$\log \mathbf{E}_{p(z)} \{f(z)\} \geq \mathbf{E}_{p(z)} \{\log f(z)\}.$$

The log marginal likelihood is given by:

$$\log p(\mathbf{Y}) = \log \int p(\mathbf{Y}, \boldsymbol{\theta}) d\boldsymbol{\theta}.$$

We start by introducing an arbitrary distribution over $\boldsymbol{\theta}$, $Q(\boldsymbol{\theta})$ into the right hand side:

$$\log p(\mathbf{Y}) = \log \int Q(\boldsymbol{\theta}) \frac{p(\mathbf{Y}, \boldsymbol{\theta})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta}.$$

Recall from Equation 6.7 that Jensen's inequality tells us that the log of an expectation is always greater than the expectation of a log. The right hand side of our expression can be interpreted as an expectation (of $p(\mathbf{Y}, \boldsymbol{\theta})/Q(\boldsymbol{\theta})$) with respect to $Q(\boldsymbol{\theta})$, so we use Jensen's inequality to construct a lower bound, $\mathcal{L}(Q)$:

$$\begin{aligned} \log p(\mathbf{Y}) &= \log \int Q(\boldsymbol{\theta}) \frac{p(\mathbf{Y}, \boldsymbol{\theta})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta} \\ &\geq \int Q(\boldsymbol{\theta}) \log \frac{p(\mathbf{Y}, \boldsymbol{\theta})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta} = \mathcal{L}(Q). \end{aligned} \quad (7.6)$$

Computing the difference between the true log marginal likelihood and our new bound reveals how we can obtain an approximate posterior:

$$\begin{aligned} \log p(\mathbf{Y}) - \mathcal{L}(Q) &= \log p(\mathbf{Y}) - \int Q(\boldsymbol{\theta}) \log \frac{p(\mathbf{Y}, \boldsymbol{\theta})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta} \\ &= \log p(\mathbf{Y}) - \int Q(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta}|\mathbf{Y})p(\mathbf{Y})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta} \\ &= \log p(\mathbf{Y}) - \int Q(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta}|\mathbf{Y})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta} - \int Q(\boldsymbol{\theta}) \log p(\mathbf{Y}) d\boldsymbol{\theta} \\ &= \log p(\mathbf{Y}) - \int Q(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta}|\mathbf{Y})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta} - \log p(\mathbf{Y}) \int Q(\boldsymbol{\theta}) d\boldsymbol{\theta} \\ &= \log p(\mathbf{Y}) - \int Q(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta}|\mathbf{Y})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta} - \log p(\mathbf{Y}) \\ \log p(\mathbf{Y}) - \mathcal{L}(Q) &= - \int Q(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta}|\mathbf{Y})}{Q(\boldsymbol{\theta})} d\boldsymbol{\theta} \doteq -\text{KL}[Q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathbf{Y})] \end{aligned} \quad (7.7)$$

The final expression is known as the Kullback-Leibler (KL) divergence between the posterior, $p(\boldsymbol{\theta}|\mathbf{Y})$ and $Q(\boldsymbol{\theta})$; see Comment 7.2.

Comment 7.2 – Kullback-Leibler divergence: It is often important to be able to quantify the difference between two probability distributions. For example, if we are trying to find an approximate posterior that is similar to the true posterior, we need to define what we mean by similar! The Kullback-Leibler divergence is one such quantity that appears in the derivation of Variational Bayesian techniques. It is defined for discrete and continuous distributions as:

$$\text{KL}[q(x)||p(x)] = \int q(x) \log \frac{p(x)}{q(x)} dx \quad (\text{continuous})$$

$$\text{KL}[q(x)||p(x)] = \sum_x q(x) \log \frac{p(x)}{q(x)} \quad (\text{discrete})$$

For continuous distributions it is almost always intractable to compute due to the integral over a potentially high-dimensional space.

An important property of the KL divergence is its asymmetry — $\text{KL}[q(x)||p(x)] \neq \text{KL}[p(x)||q(x)]$. KL divergence is always less than or equal to zero with the maximum value of zero being reached when $p(x) = q(x)$.

The left hand side of Equation 7.7 must always be greater than or equal to zero (remember that $\mathcal{L}(Q)$ is a lower bound on $\log p(\mathbf{Y})$). The KL divergence is a measure of dissimilarity between two distributions that takes the value 0 if the two distributions are identical and is otherwise less than zero. Maximising $\mathcal{L}(Q)$ by varying Q reduces the negative of the KL divergence and therefore has the effect of making $Q(\boldsymbol{\theta})$ more and more similar to the true posterior $p(\boldsymbol{\theta}|\mathbf{Y})$. If $Q(\boldsymbol{\theta})$ and $p(\boldsymbol{\theta}|\mathbf{Y})$ are identical, the bound is equal to the true log marginal likelihood (see Exercise EX 7.1).

7.4.1 Choosing $Q(\boldsymbol{\theta})$

We have seen that if we maximise the bound with respect to $Q(\boldsymbol{\theta})$, we are making $Q(\boldsymbol{\theta})$ a better and better approximation to the posterior. We need to choose the form of $Q(\boldsymbol{\theta})$ and it makes sense to choose it in such a way that makes it relatively straightforward to maximise the bound given in Equation 7.6. There is a clear trade-off here – more complex forms for $Q(\boldsymbol{\theta})$ are likely to make the bound harder to optimise but will provide us with a better approximation. Simple forms for $Q(\boldsymbol{\theta})$ will potentially make optimisation easy but the resulting approximation will probably be poor. A popular assumption is to assume independence across the different parameters/latent variables with $\boldsymbol{\theta}$:

$$Q(\boldsymbol{\theta}) = \prod_{l=1}^L Q_l(\boldsymbol{\theta}_l) \quad (7.8)$$

where each $l = 1 \dots L$ is a different individual or set of parameters or latent variables. For example, a model may have M vectors of parameters \mathbf{w}_m and N latent variable vectors \mathbf{x}_n collectively known as \mathbf{W} and \mathbf{X} , respectively. We might decide to assume independence across these sets of parameters:

$$Q(\mathbf{W}, \mathbf{X}) = Q_{\mathbf{W}}(\mathbf{W})Q_{\mathbf{X}}(\mathbf{X}).$$

We could go one step further and assume that either or both of these distributions was independent over its M (or N) components:

$$Q_{\mathbf{W}}(\mathbf{W}) = \prod_{m=1}^M Q_{\mathbf{w}_m}(\mathbf{w}_m), \quad \text{and/or} \quad Q_{\mathbf{X}}(\mathbf{X}) = \prod_{n=1}^N Q_{\mathbf{x}_n}(\mathbf{x}_n).$$

Going even further, we could, for example, assume independence across the D -dimensions of \mathbf{x}_n :

$$Q_{\mathbf{X}}(\mathbf{X}) = \prod_{n=1}^N \prod_{d=1}^D Q_{x_{nd}}(x_{nd}).$$

Given that the parameters are likely to be dependent in the true posterior, the more independence assumptions we make, the worse our approximation is likely to become. This is an example of the tradeoff we just mentioned: greater independence assumptions will make the bound easier to optimise but will result in a poorer approximate posterior.

7.4.2 Optimising the bound

If we construct $Q(\boldsymbol{\theta})$ in the manner described by Equation 7.8, the bound is optimised by distributions of the form:

$$Q_l(\boldsymbol{\theta}_l) = \frac{\exp(\mathbf{E}_{k \neq l} \{\log p(\mathbf{Y}, \boldsymbol{\theta})\})}{\int \exp(\mathbf{E}_{k \neq l} \{\log p(\mathbf{Y}, \boldsymbol{\theta})\}) d\boldsymbol{\theta}_l} \quad (7.9)$$

where the expectation is over all of the individual distributions making up the product in Equation 7.8 except the l th one.

The expression is not as ominous as it appears at first glance. The denominator is simply a normalising constant that will often be defined by the form of the terms involving $\boldsymbol{\theta}_l$ in the numerator. For example, the presence of linear ($\boldsymbol{\theta}_l^T \mathbf{b}$) and quadratic ($\boldsymbol{\theta}_l^T \mathbf{A} \boldsymbol{\theta}_l$) terms suggests that $Q_l(\boldsymbol{\theta}_l)$ is Gaussian, for which we know the normalising constant.

Computing each $Q_l(\boldsymbol{\theta}_l)$ requires taking an expectation with respect to each other $Q_k(\boldsymbol{\theta}_k)$. Much like the EM algorithm in Chapter 6, this means that we will require an iterative procedure to optimise our approximate posterior.

7.5 A probabilistic model for PCA

To illustrate Variational Bayes, we will start with a probabilistic PCA-like model. Assume that we observe $n = 1 \dots N$ M -dimensional input vectors \mathbf{y}_n . We would like to find a D -dimensional representation \mathbf{x}_n (where $D < M$). We will link \mathbf{y}_n and \mathbf{x}_n with the following model:

$$\mathbf{y}_n = \mathbf{W}\mathbf{x}_n + \mathbf{v}$$

where \mathbf{W} is an $M \times D$ matrix and \mathbf{v} is an $M \times 1$ noise vector. The graphical

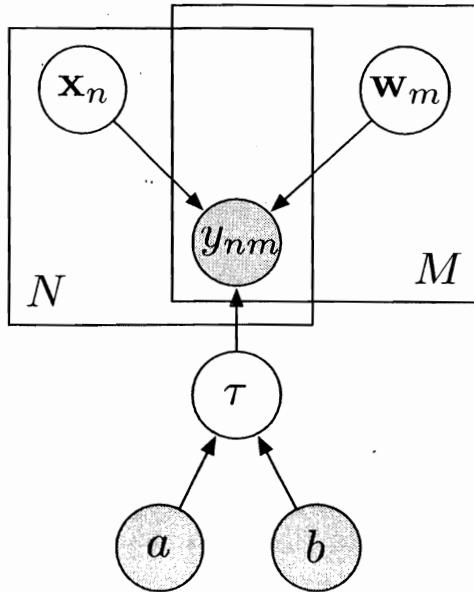


FIGURE 7.5: Graphical representation of the probabilistic PCA model.

representation (see Section 3.6) of this model can be seen in Figure 7.5. We will make the following prior assumptions:

$$\begin{aligned} p(\mathbf{x}_n) &= \mathcal{N}(\mathbf{0}, \mathbf{I}_D) \\ p(\mathbf{W}) &= \prod_{m=1}^M p(\mathbf{w}_m) \\ p(\mathbf{w}_m) &= \mathcal{N}(\mathbf{0}, \mathbf{I}_D) \\ p(y_{nm}) &= \mathcal{N}(\mathbf{w}_m^\top \mathbf{x}_n, \tau^{-1}) \\ p(\tau|a, b) &= \Gamma(a, b) = \frac{b^a \tau^{a-1} e^{-b\tau}}{\Gamma(a)} \end{aligned}$$

where $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_M]^\top$ and for convenience we have defined a **precision** rather than variance parameter for the noise: τ ($\tau^{-1} = \sigma^2$).

We will now use Variational Bayes to infer an approximate posterior over $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^\top$, \mathbf{W} and τ . The first step is to decide how we shall decompose $Q(\mathbf{W}, \mathbf{X}, \tau)$. We will do the following:

$$Q(\mathbf{W}, \mathbf{X}, \tau) = Q_\tau(\tau) \left[\prod_{n=1}^N Q_{\mathbf{x}_n}(\mathbf{x}_n) \right] \left[\prod_{m=1}^M Q_{\mathbf{w}_m}(\mathbf{w}_m) \right],$$

i.e. assume independence across the three sets of parameters (for brevity, we will refer to the latent variables \mathbf{x}_n as parameters from now on) and additionally independence across the various vector components of \mathbf{X} and \mathbf{W} .

To obtain expressions for each $Q_i(\theta_i)$, we need, from Equation 7.9, to take expec-

tations of $\log p(\mathbf{Y}, \boldsymbol{\theta})$. In our example, this is (making the standard IID assumption):

$$\begin{aligned} p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau) &= p(\tau|a, b) \left[\prod_{m=1}^M p(\mathbf{w}_m) \right] \left[\prod_{n=1}^N p(\mathbf{x}_n) p(\mathbf{y}_n|\mathbf{W}, \mathbf{x}_n, \tau) \right] \\ \log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau) &= \log p(\tau|a, b) + \sum_{m=1}^M \log p(\mathbf{w}_m) + \sum_{n=1}^N \log p(\mathbf{x}_n) \\ &\quad + \sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{W}, \mathbf{x}_n, \tau). \end{aligned}$$

Because of our assumption that the noise vector \mathbf{v} has diagonal covariance, the final term on the right hand side can be further expanded to each individual element of \mathbf{y}_n :

$$\begin{aligned} \log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau) &= \log p(\tau|a, b) \\ &\quad + \sum_{m=1}^M \log p(\mathbf{w}_m) \\ &\quad + \sum_{n=1}^N \log p(\mathbf{x}_n) \\ &\quad + \sum_{n=1}^N \sum_{m=1}^M \log p(y_{nm}|\mathbf{w}_m, \mathbf{x}_n, \tau), \end{aligned} \tag{7.10}$$

where

$$p(y_{nm}|\mathbf{w}_m, \mathbf{x}_n, \tau) = \mathcal{N}(\mathbf{w}_m^\top \mathbf{x}_n, \tau^{-1}).$$

We will now define each component of the approximate posterior in turn.

7.5.1 $Q_\tau(\tau)$

From Equation 7.9 we know that:

$$Q_\tau(\tau) \propto \exp \left(\mathbf{E}_{Q_X(\mathbf{x})Q_W(\mathbf{w})} \{ \log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau) \} \right).$$

Any terms within the expectation that do not include τ can be ignored as they will be swallowed up by the normalisation constant. The only terms that do depend on τ in Equation 7.10 are the first and the last. Writing these terms in full gives us:

$$\begin{aligned} \log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau) &\propto a \log b + (a-1) \log \tau - b\tau - \log \Gamma(a) \\ &\quad - \frac{NM}{2} \log 2\pi + \frac{NM}{2} \log \tau - \frac{\tau}{2} \sum_n \sum_m (y_{nm} - \mathbf{w}_m^\top \mathbf{x}_n)^2 \end{aligned}$$

Removing new terms that do not depend on τ leaves (and remembering that $\exp(A+B) = \exp(A)\exp(B)$):

$$\begin{aligned} Q_\tau(\tau) &\propto \exp \left(\mathbf{E}_{Q_X(\mathbf{x})Q_W(\mathbf{w})} \{ \log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau) \} \right) \\ &\propto \exp \left((a-1) \log \tau - b\tau + \frac{NM}{2} \log \tau \right) \\ &\quad \times \exp \left(-\frac{\tau}{2} \mathbf{E}_{Q_X(\mathbf{x})Q_W(\mathbf{w})} \left\{ \sum_n \sum_m (y_{nm} - \mathbf{w}_m^\top \mathbf{x}_n)^2 \right\} \right). \end{aligned}$$

Bearing in mind that:

$$\mathbf{E}_{p(a)} \{f(a) + g(a)\} = \mathbf{E}_{p(a)} \{f(a)\} + \mathbf{E}_{p(a)} \{g(a)\},$$

we can take the expectations of all of the terms in the summation separately. Also y_{nm}^2 is the observed data and therefore:

$$\mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)Q_{\mathbf{w}_m}(\mathbf{w}_m)} \{y_{nm}^2\} = y_{nm}^2.$$

We are left with:

$$\exp \left(-\frac{\tau}{2} \sum_{n,m} \left(y_{nm}^2 + \mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)Q_{\mathbf{w}_m}(\mathbf{w}_m)} \{-2\mathbf{w}_m^\top \mathbf{x}_n + \mathbf{x}_n^\top \mathbf{w}_m \mathbf{w}_m^\top \mathbf{x}_n\} \right) \right).$$

At first glance, this still looks difficult, but consider the following expectation:

$$\mathbf{E}_{p(a)p(b)} \{f(a)f(b)\}$$

Writing it out in full gives us:

$$\begin{aligned} \mathbf{E}_{p(a)p(b)} \{f(a)f(b)\} &= \iint p(a)p(b)f(a)f(b) da db \\ &= \int \int p(a)f(a) dap(b)f(b) db \\ &= \int \mathbf{E}_{p(a)} \{f(a)\} p(b)f(b) \\ &= \mathbf{E}_{p(a)} \{f(a)\} \mathbf{E}_{p(b)} \{f(b)\} \end{aligned} \quad (7.11)$$

Using this result, we can evaluate the first argument inside our expectation:

$$\mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)Q_{\mathbf{w}_m}(\mathbf{w}_m)} \{-2\mathbf{w}_m^\top \mathbf{x}_n\} = -2\mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)} \{\mathbf{x}_n\}^\top \mathbf{E}_{Q_{\mathbf{w}_m}(\mathbf{w}_m)} \{\mathbf{w}_m\},$$

which is the expected (mean) value of \mathbf{x}_n multiplied by the expected value of \mathbf{w}_m . Before we continue, it is worth introducing a more useful notation as we are going to see a lot of expressions like this. From now on, we will denote expectations such as this as:

$$\mathbf{E}_{Q_{\boldsymbol{\theta}_l}(\boldsymbol{\theta}_l)} \{f(\boldsymbol{\theta}_l)\} = \langle f(\boldsymbol{\theta}_l) \rangle.$$

The first term in the expectation becomes:

$$\mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)Q_{\mathbf{w}_m}(\mathbf{w}_m)} \{-2\mathbf{w}_m^\top \mathbf{x}_n\} = -2 \langle \mathbf{x}_n \rangle^\top \langle \mathbf{w}_m \rangle.$$

The second term is a bit trickier. We cannot write it as $f(\mathbf{x}_n)g(\mathbf{w}_m)$ so we have to do the expectations one at a time:

$$\begin{aligned} \mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)Q_{\mathbf{w}_m}(\mathbf{w}_m)} \left\{ \mathbf{x}_n^\top \mathbf{w}_m \mathbf{w}_m^\top \mathbf{x}_n \right\} &= \mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)} \left\{ \mathbf{x}_n^\top \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \mathbf{x}_n \right\} \\ &= \left\langle \mathbf{x}_n^\top \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \mathbf{x}_n \right\rangle. \end{aligned}$$

Putting everything back together, we have:

$$\begin{aligned} Q_\tau(\tau) \propto \exp \left((a-1) \log \tau - b\tau + \frac{NM}{2} \log \tau \right. \\ \left. - \frac{\tau}{2} \sum_{n,m} \left(y_{nm}^2 - 2 \langle \mathbf{w}_n \rangle^\top \langle \mathbf{x}_n \rangle + \left\langle \mathbf{x}_n^\top \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \mathbf{x}_n \right\rangle \right) \right). \end{aligned}$$

This can be written as:

$$Q_\tau(\tau) \propto \tau^{e-1} \exp\{-\tau f\}, \quad (7.12)$$

where

$$\begin{aligned} e &= a + \frac{NM}{2} \\ f &= b + \frac{1}{2} \sum_{n,m} \left(y_{nm}^2 - 2 \langle \mathbf{w}_n \rangle^\top \langle \mathbf{x}_n \rangle + \left\langle \mathbf{x}_n^\top \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \mathbf{x}_n \right\rangle \right) \end{aligned}$$

The form of Equation 7.12 tells us that $Q_\tau(\tau)$ is a gamma distribution, with parameters e, f . If you are unsure of why this is the case, take the log of a gamma density with parameters e and f and remove terms that do not depend on τ – you will get the right hand side of Equation 7.12.

To summarise:

$$Q_\tau(\tau) = \Gamma(e, f)$$

We will now derive $Q_{\mathbf{x}_n}(\mathbf{x}_n)$ and $Q_{\mathbf{w}_m}(\mathbf{w}_m)$ – once we know the forms of these we will be able to compute the expectations required to calculate e and f .

7.5.2 $Q_{\mathbf{x}_n}(\mathbf{x}_n)$

The steps required to obtain $Q_{\mathbf{x}_n}(\mathbf{x}_n)$ are much the same as those we needed to get $Q_\tau(\tau)$. To start with, we extract the terms we need from $\log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau)$, ignoring everything that does not involve \mathbf{x}_n :

$$Q_{\mathbf{x}_n}(\mathbf{x}_n) \propto \exp \left(\mathbf{E}_{Q_{\mathbf{W}}(\mathbf{W})Q_\tau(\tau)} \left\{ \log p(\mathbf{x}_n) + \sum_{m=1}^M p(y_{nm} | \mathbf{w}_m, \mathbf{x}_n, \tau) \right\} \right)$$

Note that the expectation should also be with respect to all $Q_{\mathbf{x}_l}(\mathbf{x}_l)$ for all $l \neq n$. However, there are no \mathbf{x}_l terms in our expression and subsequently the expectations vanish. Expanding the two terms in the expectation and removing non \mathbf{x}_n terms, we have:

$$\begin{aligned} Q_{\mathbf{x}_n}(\mathbf{x}_n) &\propto \exp \left(\mathbf{E}_{Q_{\mathbf{W}}(\mathbf{W})Q_\tau(\tau)} \left\{ -\frac{1}{2} \mathbf{x}_n^\top \mathbf{x}_n \right. \right. \\ &\quad \left. \left. - \frac{1}{2} \tau \sum_m \left(-2y_{nm} \mathbf{x}_n^\top \mathbf{w}_m + \mathbf{x}_n^\top \mathbf{w}_m \mathbf{w}_m^\top \mathbf{x}_n \right) \right\} \right) \\ &\propto \exp \left(-\frac{1}{2} \mathbf{x}_n^\top \mathbf{x}_n - \frac{1}{2} \langle \tau \rangle \sum_m \left(-2y_{nm} \mathbf{x}_n^\top \langle \mathbf{w}_m \rangle + \mathbf{x}_n^\top \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \mathbf{x}_n \right) \right) \\ &\propto \exp \left(-\frac{1}{2} \left(\mathbf{x}_n^\top \left[\mathbf{I}_D + \langle \tau \rangle \sum_m \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \right] \mathbf{x}_n - 2 \langle \tau \rangle \mathbf{x}_n^\top \sum_m y_{nm} \langle \mathbf{w}_m \rangle \right) \right) \end{aligned}$$

The presence of the linear and quadratic terms within the expectation tells us that this is a Gaussian:

$$Q_{\mathbf{x}_n}(\mathbf{x}_n) = \mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}_n}, \boldsymbol{\Sigma}_{\mathbf{x}_n}).$$

Equating coefficients lets us read off expressions for $\mu_{\mathbf{x}_n}$ and $\Sigma_{\mathbf{x}_n}$:

$$\begin{aligned}\mathbf{x}_n^\top \Sigma_{\mathbf{x}_n}^{-1} \mathbf{x}_n &\equiv \mathbf{x}_n^\top \left[\mathbf{I}_D + \langle \tau \rangle \sum_m \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \right] \mathbf{x}_n \\ \Sigma_{\mathbf{x}_n} &= \left[\mathbf{I}_D + \langle \tau \rangle \sum_m \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \right]^{-1}\end{aligned}\quad (7.13)$$

$$\begin{aligned}-2\mathbf{x}_n^\top \Sigma_{\mathbf{x}_n}^{-1} \boldsymbol{\mu}_{\mathbf{x}_n} &\equiv -2\langle \tau \rangle \mathbf{x}_n^\top \sum_m y_{nm} \langle \mathbf{w}_m \rangle \\ \boldsymbol{\mu}_{\mathbf{x}_n} &= \langle \tau \rangle \Sigma_{\mathbf{x}_n} \sum_m y_{nm} \langle \mathbf{w}_m \rangle.\end{aligned}\quad (7.14)$$

Note that the expression for the covariance matrix $\Sigma_{\mathbf{x}_n}$ has no dependence on n . It can be computed once and used for all \mathbf{x}_n .

7.5.3 $Q_{\mathbf{w}_m}(\mathbf{w}_m)$

The method for computing $Q_{\mathbf{w}_m}(\mathbf{w}_m)$ is essentially identical to that for computing $Q_{\mathbf{x}_n}(\mathbf{x}_n)$. We start by removing everything that doesn't have a \mathbf{w}_m term:

$$Q_{\mathbf{w}_m}(\mathbf{w}_m) \propto \exp \left(\mathbf{E}_{Q_{\mathbf{X}}(\mathbf{X})Q_{\tau}(\tau)} \left\{ \log p(\mathbf{w}_m) + \sum_{n=1}^N p(y_{nm} | \mathbf{w}_m, \mathbf{x}_n, \tau) \right\} \right)$$

Once again, the expectations with respect to $Q_{\mathbf{w}_l}(\mathbf{w}_l)$ for all $l \neq m$ vanish. Expanding, noticing that $\mathbf{w}_m^\top \mathbf{x}_n \equiv \mathbf{x}_n^\top \mathbf{w}_m$:

$$\begin{aligned}Q_{\mathbf{w}_m}(\mathbf{w}_m) &\propto \exp \left(\mathbf{E}_{Q_{\mathbf{X}}(\mathbf{X})Q_{\tau}(\tau)} \left\{ -\frac{1}{2} \mathbf{w}_m^\top \mathbf{w}_m \right. \right. \\ &\quad \left. \left. - \frac{1}{2} \tau \sum_n (-2y_{nm} \mathbf{w}_m^\top \mathbf{x}_n + \mathbf{w}_m^\top \mathbf{x}_n \mathbf{x}_n^\top \mathbf{w}_m) \right\} \right) \\ &\propto \exp \left(-\frac{1}{2} \mathbf{w}_m^\top \mathbf{w}_m - \frac{1}{2} \langle \tau \rangle \sum_n (-2y_{nm} \mathbf{w}_m^\top \langle \mathbf{x}_n \rangle + \mathbf{w}_m^\top \langle \mathbf{x}_n \mathbf{x}_n^\top \rangle \mathbf{w}_m) \right) \\ &\propto \exp \left(-\frac{1}{2} \left(\mathbf{w}_m^\top \left[\mathbf{I}_D + \langle \tau \rangle \sum_n \langle \mathbf{x}_n \mathbf{x}_n^\top \rangle \right] \mathbf{w}_m - 2\langle \tau \rangle \mathbf{w}_m^\top \sum_n y_{nm} \langle \mathbf{x}_n \rangle \right) \right)\end{aligned}$$

This is clearly another Gaussian:

$$\begin{aligned}Q_{\mathbf{w}_m}(\mathbf{w}_m) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{w}_m}, \Sigma_{\mathbf{w}_m}) \\ \Sigma_{\mathbf{w}_m} &= \left[\mathbf{I}_D + \langle \tau \rangle \sum_n \langle \mathbf{x}_n \mathbf{x}_n^\top \rangle \right]^{-1} \\ \boldsymbol{\mu}_{\mathbf{w}_m} &= \langle \tau \rangle \Sigma_{\mathbf{w}_m} \sum_n y_{nm} \langle \mathbf{x}_n \rangle\end{aligned}$$

As with $\Sigma_{\mathbf{x}_n}$, the covariance matrix $\Sigma_{\mathbf{w}_m}$ has no dependence on m and so can be computed once for all \mathbf{w}_m .

7.5.4 The required expectations

Each of the components of our approximate posterior, $Q_{\mathbf{x}_n}(\mathbf{x}_n)$, $Q_{\mathbf{w}_m}(\mathbf{w}_m)$, $Q_\tau(\tau)$, depends on expectations (e.g. $\langle \mathbf{x}_n \rangle$ and $\langle \mathbf{w}_m \mathbf{w}_m^\top \rangle$) with respect to the other components. As all of the components are well known distributions, these expectations are all standard results. $Q_{\mathbf{x}_n}(\mathbf{x}_n)$ and $Q_{\mathbf{w}_m}(\mathbf{w}_m)$ are both Gaussian and therefore:

$$\begin{aligned}\langle \mathbf{x}_n \rangle &= \boldsymbol{\mu}_{\mathbf{x}_n} & \left\langle \mathbf{x}_n \mathbf{x}_n^\top \right\rangle &= \boldsymbol{\Sigma}_{\mathbf{x}_n} + \boldsymbol{\mu}_{\mathbf{x}_n} \boldsymbol{\mu}_{\mathbf{x}_n}^\top \\ \langle \mathbf{w}_m \rangle &= \boldsymbol{\mu}_{\mathbf{w}_m} & \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle &= \boldsymbol{\Sigma}_{\mathbf{w}_m} + \boldsymbol{\mu}_{\mathbf{w}_m} \boldsymbol{\mu}_{\mathbf{w}_m}^\top\end{aligned}$$

$Q_\tau(\tau)$ is a gamma distribution so:

$$\langle \tau \rangle = \frac{e}{f}$$

The final expectation we need is $\langle \mathbf{x}_n^\top \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \mathbf{x}_n \rangle$. This is of the form $\langle \mathbf{z}^\top \mathbf{A} \mathbf{z} \rangle$, which, if $p(\mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, is equal to:

$$\left\langle \mathbf{z}^\top \mathbf{A} \mathbf{z} \right\rangle = \text{Tr}(\mathbf{A} \boldsymbol{\Sigma}) + \boldsymbol{\mu}^\top \mathbf{A} \boldsymbol{\mu}$$

Therefore:

$$\left\langle \mathbf{x}_n^\top \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \mathbf{x}_n \right\rangle = \text{Tr} \left(\left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \boldsymbol{\Sigma}_{\mathbf{x}_n} \right) + \boldsymbol{\mu}_{\mathbf{x}_n}^\top \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \boldsymbol{\mu}_{\mathbf{x}_n}$$

7.5.5 The algorithm

We now have everything we need to obtain an approximate posterior $Q(\mathbf{W}, \mathbf{X}, \tau)$ using variational Bayes (VB). We must first initialise the various parameters. We will start by initialising $\langle \tau \rangle = a/b$ (its expected prior value) and then sample each $\langle \mathbf{w}_m \rangle$ from $\mathcal{N}(\mathbf{0}, \mathbf{I}_D)$ and compute $\langle \mathbf{w} \mathbf{w}^\top \rangle = \mathbf{I}_D + \langle \mathbf{w}_m \rangle \langle \mathbf{w}_m \rangle^\top$. We can now compute $\boldsymbol{\mu}_{\mathbf{x}_n}$ and $\boldsymbol{\Sigma}_{\mathbf{x}_n}$ and hence $\langle \mathbf{x}_n \rangle$ and $\langle \mathbf{x}_n \mathbf{x}_n^\top \rangle$. We proceed as follows:

1. For all n , compute $\boldsymbol{\Sigma}_{\mathbf{x}_n}$ and $\boldsymbol{\mu}_{\mathbf{x}_n}$ and update $\langle \mathbf{x}_n \rangle$ and $\langle \mathbf{x}_n \mathbf{x}_n^\top \rangle$.
2. Using the new values of $\langle \mathbf{x}_n \rangle$ and $\langle \mathbf{x}_n \mathbf{x}_n^\top \rangle$, compute $\boldsymbol{\mu}_{\mathbf{w}_m}$ and $\boldsymbol{\Sigma}_{\mathbf{w}_m}$ and update $\langle \mathbf{w}_m \rangle$ and $\langle \mathbf{w}_m \mathbf{w}_m^\top \rangle$ for all m .
3. Compute $\langle \mathbf{x}_n^\top \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \mathbf{x}_n \rangle$ for all n and m .
4. Compute e and f and update $\langle \tau \rangle$.
5. If not converged, return to 1.

To check convergence, we can either monitor how much the various parameters are changing or compute the bound, $\mathcal{L}(\boldsymbol{\theta})$ (Equation 7.6), which will increase until convergence and then remain unchanged. The bound is given by:

$$\begin{aligned}\mathcal{L}(\mathbf{X}, \mathbf{W}, \tau) &= \int Q(\mathbf{X}, \mathbf{W}, \tau) \log \frac{p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau)}{Q(\mathbf{X}, \mathbf{W}, \tau)} dQ(\mathbf{X}, \mathbf{W}, \tau) \\ &= \int Q(\cdot) \log p(\cdot) dQ(\cdot) - \int Q(\cdot) \log Q(\cdot) dQ(\cdot).\end{aligned}$$

Making use of the independence assumptions and noting that both expressions are expectations with respect to $Q(\cdot)$, we can further decompose these two terms as:

$$\begin{aligned} \int Q(\cdot) \log p(\cdot) dQ(\cdot) &= \mathbf{E}_{Q_{\tau}(\tau)} \{ \log p(\tau|a, b) \} \\ &\quad + \sum_{n=1}^N \mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)} \{ \log p(\mathbf{x}_n) \} \\ &\quad + \sum_{m=1}^M \mathbf{E}_{Q_{\mathbf{w}_m}(\mathbf{w}_m)} \{ \log p(\mathbf{w}_m) \} \\ &\quad + \sum_{n=1}^N \sum_{m=1}^M \mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)Q_{\mathbf{w}_m}(\mathbf{w}_m)Q_{\tau}(\tau)} \{ \log p(y_{nm}|\mathbf{x}_n, \mathbf{w}_m, \tau) \}, \end{aligned}$$

and:

$$\begin{aligned} \int Q(\cdot) \log Q(\cdot) dQ(\cdot) &= \mathbf{E}_{Q_{\tau}(\tau)} \{ \log Q_{\tau}(\tau) \} \\ &\quad + \sum_{n=1}^N \mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n)} \{ \log Q_{\mathbf{x}_n}(\mathbf{x}_n) \} \\ &\quad + \sum_{m=1}^M \mathbf{E}_{Q_{\mathbf{w}_m}(\mathbf{w}_m)} \{ \log Q_{\mathbf{w}_m}(\mathbf{w}_m) \}. \end{aligned}$$

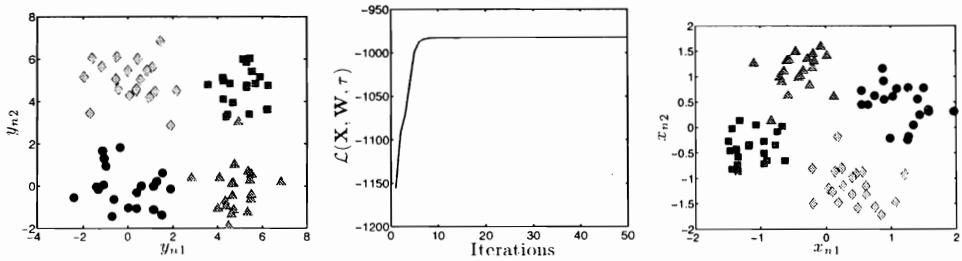
It is left to the reader (see Exercise EX 7.2) to show that these individual terms give the following bound (each line corresponds to one of the expectations, in the same order as above):

$$\begin{aligned} \mathcal{L}(\mathbf{X}, \mathbf{W}, \tau) &= a \log b + (a - 1) \langle \log \tau \rangle - b \langle \tau \rangle - \log \Gamma(a) \\ &\quad - \frac{ND}{2} \log 2\pi - \frac{1}{2} \sum_n \left(\text{Tr}(\Sigma_{\mathbf{x}_n}) + \mu_{\mathbf{x}_n}^\top \mu_{\mathbf{x}_n} \right) \\ &\quad - \frac{MD}{2} \log 2\pi - \frac{1}{2} \sum_m \left(\text{Tr}(\Sigma_{\mathbf{w}_m}) + \mu_{\mathbf{w}_m}^\top \mu_{\mathbf{w}_m} \right) \\ &\quad - \frac{NM}{2} \log 2\pi + \frac{NM}{2} \langle \log \tau \rangle - \frac{1}{2} \langle \tau \rangle \sum_{n,m} \langle (y_{nm} - \mathbf{w}_m^\top \mathbf{x}_n)^2 \rangle \\ &\quad - (e \log f + (e - 1) \langle \log \tau \rangle - f \langle \tau \rangle - \log \Gamma(e)) \\ &\quad - \left(-\frac{ND}{2} \log 2\pi - \frac{ND}{2} - \frac{1}{2} \sum_n \log |\Sigma_{\mathbf{x}_n}| \right) \\ &\quad - \left(-\frac{MD}{2} \log 2\pi - \frac{MD}{2} - \frac{1}{2} \sum_m \log |\Sigma_{\mathbf{w}_m}| \right) \end{aligned}$$

where

$$\langle (y_{nm} - \mathbf{w}_m^\top \mathbf{x}_n)^2 \rangle = y_{nm}^2 - 2y_{nm} \langle \mathbf{x}_n \rangle^\top \langle \mathbf{w}_m \rangle + \langle \mathbf{x}_n^\top \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \mathbf{x}_n \rangle,$$

all of which we have already computed. The only term in the bound that we have not seen before is $\langle \log \tau \rangle$. We are forced to approximate this term and can do so by



(a) First two dimensions of the data objects \mathbf{y}_n (b) Evolution of the lower bound $\mathcal{L}(\mathbf{X}, \mathbf{W}, \tau)$ (c) The posterior mean of the latent variables

FIGURE 7.6: Synthetic probabilistic PCA example.

sampling. If we take S samples, τ^1, \dots, τ^S , the approximation is given by:

$$\langle \log \tau \rangle \approx \frac{1}{S} \sum_{s=1}^S \log \tau^s.$$

7.5.6 An example

Figure 7.6(a) shows a dataset generated in the same manner as that used in the example depicted in Figure 7.4 (MATLAB script: `ppcaexample.m`). The first two dimensions are shown, in which there is clear cluster structure. To this, an additional five dimensions are added ($y_{nm} \sim \mathcal{N}(0, 1)$). The evolution of the bound, $\mathcal{L}(\mathbf{X}, \mathbf{W}, \tau)$, as the algorithm progresses (with $D = 2$) can be seen in Figure 7.6(b). The bound increases monotonically until it converges, which is after only a small number of iterations. In Figure 7.6(c) we can see the posterior means of the latent variables. These are analogous to the standard PCA projections (see, e.g. Figure 7.4(c)). It is clear that the cluster structure is captured in the latent space.

7.6 Missing values

One of our motivations for moving to a probabilistic representation was its ability to handle missing values. In the model we defined in the previous section, the easiest way to achieve this is to only define the model for the data that we observe. To this end, we introduce a new set of binary variables z_{nm} which are equal to 1 if we observe feature m for object n and 0 otherwise. Collecting all of the z_{nm} together

into a matrix \mathbf{Z} , we have:

$$\begin{aligned} p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau | \mathbf{Z}) &= p(\tau | a, b) \left[\prod_{m=1}^M p(\mathbf{w}_m) \right] \left[\prod_{n=1}^N p(\mathbf{x}_n) \prod_{m=1}^M p(y_{nm} | \mathbf{w}_m, \mathbf{x}_n, \tau)^{z_{nm}} \right] \\ \log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau | \mathbf{Z}) &= \log p(\tau | a, b) + \sum_{m=1}^M \log p(\mathbf{w}_m) + \sum_{n=1}^N \log p(\mathbf{x}_n) \\ &\quad + \sum_{n=1}^N \sum_{m=1}^M z_{nm} \log p(y_{nm} | \mathbf{w}_m, \mathbf{x}_n, \tau). \end{aligned} \quad (7.15)$$

The binary variables act as switches, only switching on the terms for which we observe data. Note how everything is conditioned on \mathbf{Z} . It is left as an exercise (see Exercise EX 7.3) for the reader to follow the steps detailed in the previous section to derive the necessary variational distributions. These are:

$$\begin{aligned} Q_{\mathbf{x}_n}(\mathbf{x}_n) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}_n}, \boldsymbol{\Sigma}_{\mathbf{x}_n}) \\ \boldsymbol{\Sigma}_{\mathbf{x}_n} &= \left[\mathbf{I}_D + \langle \tau \rangle \sum_m z_{nm} \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \right]^{-1} \\ \boldsymbol{\mu}_{\mathbf{x}_n} &= \langle \tau \rangle \boldsymbol{\Sigma}_{\mathbf{x}_n} \sum_m z_{nm} y_{nm} \langle \mathbf{w}_m \rangle \\ Q_{\mathbf{w}_m}(\mathbf{w}_m) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{w}_m}, \boldsymbol{\Sigma}_{\mathbf{w}_m}) \\ \boldsymbol{\Sigma}_{\mathbf{w}_m} &= \left[\mathbf{I}_D + \langle \tau \rangle \sum_n z_{nm} \langle \mathbf{x}_n \mathbf{x}_n^\top \rangle \right]^{-1} \\ \boldsymbol{\mu}_{\mathbf{w}_m} &= \langle \tau \rangle \boldsymbol{\Sigma}_{\mathbf{w}_m} \sum_n z_{nm} y_{nm} \langle \mathbf{x}_n \rangle \\ Q_\tau(\tau) &= \Gamma(e, f) \\ e &= a + \frac{1}{2} \sum_{n,m} z_{nm} \\ f &= b + \frac{1}{2} \sum_{n,m} z_{nm} \left(y_{nm}^2 - 2 \langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle + \langle \mathbf{x}_n^\top \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \mathbf{x}_n \rangle \right) \end{aligned}$$

In the previous section, we noted that the equations for $\boldsymbol{\Sigma}_{\mathbf{x}_n}$ and $\boldsymbol{\Sigma}_{\mathbf{w}_m}$ had no dependence on n and m , respectively, and therefore did not have to be computed for each n and m . Due to the presence of z_{nm} in both expressions, this is no longer the case and a different $\boldsymbol{\Sigma}_{\mathbf{x}_n}$ must be computed for each \mathbf{x}_n (and a $\boldsymbol{\Sigma}_{\mathbf{w}_m}$ for each \mathbf{w}_m). Both \mathbf{x}_n and \mathbf{w}_m are D -dimensional and so for large N , M or D , this is a considerable additional computational overhead. In Section 7.4.1 we mentioned the possibility of making additional independence assumptions when defining the components of the approximate posterior. In particular:

$$Q_{\mathbf{x}_n}(\mathbf{x}_n) = \prod_{d=1}^D Q_{x_{nd}}(x_{nd})$$

would result in us having to work with scalar variances rather than the $D \times D$ covariance matrices, $\boldsymbol{\Sigma}_{\mathbf{x}_n}$. This would represent a considerable computational saving but would come at the cost of a poorer approximation to the posterior.

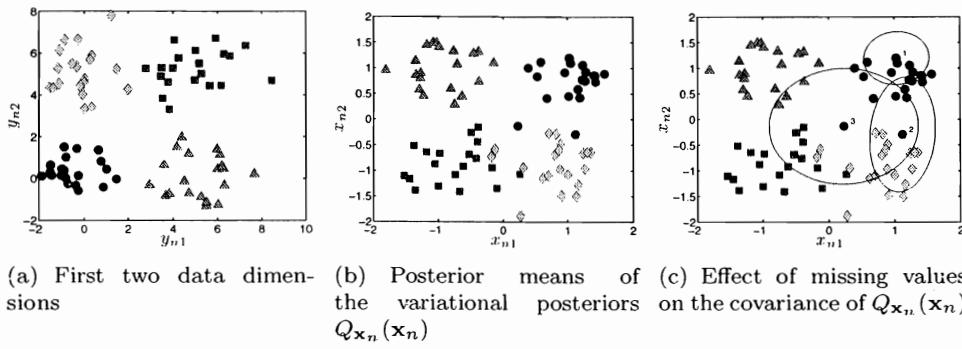


FIGURE 7.7: Variational Bayesian PPCA model with missing values. The data is the same as that shown in Figure 7.6 with each value of y_{nm} removed with a probability of 0.05.

Figure 7.7 gives an example of the Probabilistic PCA model with missing values (MATLAB script: `ppcamvexample.m`). Figure 7.7(b) shows the posterior means of the variational posteriors $Q_{\mathbf{x}_n}(\mathbf{x}_n)$. The data is the same as in the previous examples (cluster structure in the first two dimensions, shown in Figure 7.7(a), noise in the remaining 5), with each observation y_{nm} removed with probability 0.05. As we might expect, the effect of removing data is that the cluster structure becomes less distinct. The individual covariance matrices make this effect easy to visualise. In Figure 7.7(c), we have visualised the covariance matrices as ellipses for three objects belonging to the ‘circle’ class. The ellipses show us the level of uncertainty that the model ascribes to the values of the latent variables. Object 1 has no values missing in the two dimensions that encode the cluster structure and sits comfortably with the other objects of the same class. Object 2 is missing its value for y_{n2} and hence the information that determines whether it is in the circle or diamond class (see Figure 7.7(a)). This is reflected in both its mean and covariance in Figure 7.7(c) – the model places it halfway between the two groups, but this allows the possibility that it could be in either. Finally, object 3 is missing values for both y_{n1} and y_{n2} – all of the non-noise features. The model places it close to the origin (remember that the prior $p(\mathbf{x}_n) = \mathcal{N}(\mathbf{0}, \mathbf{I}_D)$) but with very high uncertainty – it could belong to any group.

It is clear that the covariance information shown in Figure 7.7(c) is a useful output of the model. It is important to know if there is a high level of uncertainty about the position, \mathbf{x}_n , of an object in the latent space. In other words, if we just saw Figure 7.7(c), we could potentially deduce that we should not draw any serious conclusions about the position of object 3 – the covariance is high because many values are missing. In Section 7.7.2 we shall see an interesting example where covariance information is useful.

7.6.1 Missing values as latent variables

In the previous section we saw how the VB framework allowed us to solve the problem of missing values – we only included the observed values in our model.

The increased uncertainty that should be present when many values are missing is naturally handled through the individual covariance matrices, $\Sigma_{\mathbf{x}_n}$. Individual covariance matrices come with an additional computational load which may, in some cases, be prohibitive. Alternatively, it is possible to consider the missing values as additional latent variables. Introducing the superscripts h and o to denote hidden and observed, respectively, this corresponds to the following joint log likelihood:

$$\begin{aligned} \log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \tau | \mathbf{Z}) &= \log p(\tau | a, b) + \sum_{m=1}^M \log p(\mathbf{w}_m) + \sum_{n=1}^N \log p(\mathbf{x}_n) \\ &\quad + \sum_{n=1}^N \sum_{m=1}^M z_{nm} \log p(y_{nm}^o | \mathbf{w}_m, \mathbf{x}_n, \tau) \\ &\quad + \sum_{n=1}^N \sum_{m=1}^M (1 - z_{nm}) \log p(y_{nm}^h | \mathbf{w}_m, \mathbf{x}_n, \tau). \end{aligned}$$

In addition, we require another set of variational posteriors: $Q_{y_{nm}^h}(y_{nm}^h)$. We will omit the derivation of the VB algorithm here but will just state the important results. First, the additional variational posteriors:

$$Q_{y_{nm}^h}(y_{nm}^h) = \mathcal{N}\left(\langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle, \langle \tau \rangle^{-1}\right).$$

Therefore, $\langle y_{nm}^h \rangle = \langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle$. $Q_{\mathbf{x}_n}(\mathbf{x}_n)$ is given by a Gaussian with parameters:

$$\begin{aligned} \Sigma_{\mathbf{x}_n} &= \left[\mathbf{I}_D + \langle \tau \rangle \sum_m z_{nm} \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle + \langle \tau \rangle \sum_m (1 - z_{nm}) \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \right]^{-1} \\ &= \left[\mathbf{I}_D + \langle \tau \rangle \sum_m \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \right]^{-1} \\ \mu_{\mathbf{x}_n} &= \langle \tau \rangle \Sigma_{\mathbf{x}_n} \sum_m \left(z_{nm} y_{nm}^o + (1 - z_{nm}) \langle y_{nm}^h \rangle \right) \langle \mathbf{w}_m \rangle \\ &= \langle \tau \rangle \Sigma_{\mathbf{x}_n} \sum_m y_{nm}^* \langle \mathbf{w}_m \rangle \end{aligned} \tag{7.16}$$

where \mathbf{y}_n^* is a vector with elements y_{nm}^o and $\langle y_{nm}^h \rangle$ depending on whether the particular parameter is observed or not. $Q_{\mathbf{w}_m}(\mathbf{w}_m)$ and $Q_\tau(\tau)$ follow similarly.

It is clear from Equation 7.16 that the covariance of $Q_{\mathbf{x}_n}(\mathbf{x}_n)$ no longer depends on n and so we no longer require specific covariance matrices for each object. In fact, one can view the resulting VB algorithm as identical to our original VB PCA algorithm where we insert the value that the model expects for y_{nm} (i.e. $\langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle$) into each missing value. This gives us a large computational saving but we lose the object specific covariance matrices, $\Sigma_{\mathbf{x}_n}$. All objects have the same covariance in the latent space, regardless of how many values are observed/missing because the expected values of the unobserved data, $\langle y_{nm}^h \rangle$, are given the same influence as the actual observations. If there are very few missing values, this is unlikely to be a problem. If there are many, it should be avoided.

7.6.2 Predicting missing values

One of the benefits of considering the missing values as latent variables is that it automatically imputes the missing values. However, we can still achieve this with the original missing value model. In particular, the expected value of y_{nm}^h with respect to the variational posteriors is:

$$\begin{aligned}\mathbf{E}_{Q(\cdot)} \left\{ y_{nm}^h \right\} &= \mathbf{E}_{Q(\cdot)} \left\{ \mathbf{w}_m^\top \mathbf{x}_n + \epsilon \right\} \\ &= \langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle\end{aligned}$$

where $\epsilon \sim \mathcal{N}(0, \tau^{-1})$. The variance of the predicted value is given by:

$$\begin{aligned}\text{var}\{y_{nm}^h\} &= \mathbf{E}_{Q(\cdot)} \left\{ (y_{nm}^h)^2 \right\} - \mathbf{E}_{Q(\cdot)} \left\{ y_{nm}^h \right\}^2 \\ &= \left\langle \mathbf{x}_n^\top \left\langle \mathbf{w}_m \mathbf{w}_m^\top \right\rangle \mathbf{x}_n \right\rangle + \langle \tau \rangle^{-1} - \langle \mathbf{x}_n \rangle^\top \langle \mathbf{w}_m \rangle \langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle\end{aligned}$$

As an example, consider object 2 in Figure 7.7(c) – using these expressions, the missing value y_{n2}^h has a mean of 0.5839 and a variance of 5.4070.

7.7 Non-real-valued data

Wanting to handle non-real-valued data was the second motivation we discussed in Section 7.2.2 for moving to a probabilistic representation. In our derivation for VB PPCA, we used a Gaussian likelihood. Using the same steps, we can use VB inference for PCA-like models with alternative likelihoods. An interesting dataset that consists of both non-real data and missing values is the voting history of British members of parliament (MPs) available from the Public Whip (<http://www.publicwhip.org.uk>). British MPs are appointed for a parliament at a general election. A parliament typically lasts four or five years, in which time there will be > 1000 divisions (votes) in which they may take part. Each division consists of a binary choice (the MP is either for or against whatever is being proposed). MPs do not have to vote – they can abstain or they might simply not be present in parliament on a particular day. Hence, the data is both non-real valued (binary) and contains many missing values.

As we have already seen in Chapter 4, likelihoods for binary data normally come with analytical problems. Rather than going over the material in that chapter again, we will now show an alternative approach based on introducing an auxiliary (or hidden) variable. It is not our intention to suggest this is the only way of solving this problem for PPCA-like models but it is a good example of a more general technique for handling binary likelihoods. An alternative probabilistic binary PCA algorithm is given in the suggested reading at the end of this chapter.

7.7.1 Probit PPCA

We potentially observe M votes for N MPs. For each vote, assuming the value is not missing (i.e. $z_{nm} = 1$), we observe $y_{nm} = \pm 1$. As before, we will assume

that there exist some D -dimensional unobserved latent variables \mathbf{x}_n which are then projected by a set of vectors \mathbf{w}_m . In our previous examples, we used a Gaussian likelihood for $p(y_{nm}|\mathbf{w}_m, \mathbf{x}_n)$. To model the binary MP data, we will use the *probit* likelihood instead. The probit function (also known as the normal cdf function) is defined as:

$$\phi(z) = \int_{-\infty}^z \exp\left\{-\frac{1}{2}x^2\right\} dx$$

and transforms a real-valued argument, z , into the range 0 to 1 (like the sigmoid function we used for logistic regression in Chapter 4). In particular, we will define:

$$P(y_{nm} = 1|\mathbf{w}_m, \mathbf{x}_n) = \phi(\mathbf{w}_m^\top \mathbf{x}_n) \quad (7.17)$$

and

$$P(y_{nm} = -1|\mathbf{w}_m, \mathbf{x}_n) = 1 - P(y_{nm} = 1|\mathbf{w}_m, \mathbf{x}_n).$$

Unfortunately, if we tried to derive variational posteriors $Q_{\mathbf{x}_n}(\mathbf{x}_n)$, etc., we would discover that they were not of any recognisable form. At this point, we make use of a slightly odd trick. We start by introducing a new set of (real-valued) variables, q_{nm} :

$$p(q_{nm}|\mathbf{w}_m, \mathbf{x}_n) = \mathcal{N}(\mathbf{w}_m^\top \mathbf{x}_n, 1),$$

which we link to the observed data y_{nm} through the following likelihood:

$$P(y_{nm} = 1|q_{nm}) = \delta(q_{nm} > 0)$$

and

$$P(y_{nm} = -1|q_{nm}) = \delta(q_{nm} < 0).$$

To justify this choice, consider the joint distribution over q_{nm} and y_{nm} :

$$p(y_{nm} = 1, q_{nm}|\mathbf{w}_m, \mathbf{x}_n) = P(y_{nm} = 1|q_{nm})p(q_{nm}|\mathbf{w}_m, \mathbf{x}_n).$$

Choosing $P(y_{nm} = 1|q_{nm}) = \delta(q_{nm} > 0)$ means that if we marginalise over q_{nm} , we get back to our original probit likelihood (Equation 7.17):

$$\begin{aligned} P(y_{nm} = 1|\mathbf{w}_m, \mathbf{x}_n) &= \int p(y_{nm} = 1, q_{nm}|\mathbf{w}_m, \mathbf{x}_n) dq_{nm} \\ &= \int P(y_{nm} = 1|q_{nm})p(q_{nm}|\mathbf{w}_m, \mathbf{x}_n) dq_{nm} \\ &= \int_{-\infty}^{\infty} \delta(q_{nm} > 0)\mathcal{N}(\mathbf{w}_m^\top \mathbf{x}_n, 1) dq_{nm} \\ &= \int_0^{\infty} \mathcal{N}(\mathbf{w}_m^\top \mathbf{x}_n, 1) dq_{nm} \\ &= \int_{-\mathbf{w}_m^\top \mathbf{x}_n}^{\infty} \mathcal{N}(0, 1) dq_{nm} \\ &= \int_{-\infty}^{\mathbf{w}_m^\top \mathbf{x}_n} \mathcal{N}(0, 1) dq_{nm} = \phi(\mathbf{w}_m^\top \mathbf{x}_n). \end{aligned}$$

This suggests that we can think of the probit likelihood as the result of a model with an additional parameter, q_{nm} , that has been integrated out. It turns out that if the parameter is left in (we treat q_{nm} as a latent variable and infer its value) the

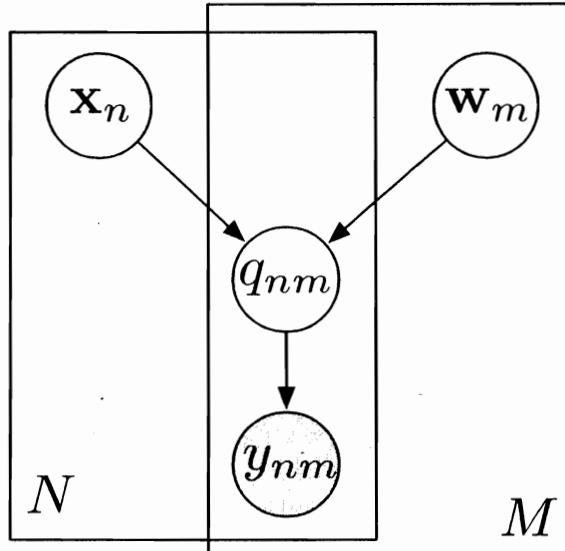


FIGURE 7.8: Graphical representation of probit PCA model.

VB algorithm becomes quite simple, even though there are an additional $N \times M$ parameters. The graphical representation of this model can be seen in Figure 7.8.

Collating all of the q_{nm} into an $N \times M$ matrix \mathbf{Q} gives us the following starting point for VB:

$$\begin{aligned} \log p(\mathbf{Y}, \mathbf{X}, \mathbf{W}, \mathbf{Q}) &= \log \left[\prod_{m=1}^M p(\mathbf{w}_m) \right] \left[\prod_{n=1}^N p(\mathbf{x}_n) \right] \\ &\quad \times \left[\prod_{n=1}^N \prod_{m=1}^M p(y_{nm} | q_{nm})^{z_{nm}} p(q_{nm} | \mathbf{w}_m, \mathbf{x}_n)^{z_{nm}} \right] \\ &= \sum_{m=1}^M \log p(\mathbf{w}_m) + \sum_{n=1}^N \log p(\mathbf{x}_n) \\ &\quad + \sum_{n=1}^N \sum_{m=1}^M z_{nm} [\log p(y_{nm} | q_{nm}) + \log p(q_{nm} | \mathbf{w}_m, \mathbf{x}_n)]. \end{aligned}$$

For our variational approximation, we need $Q_{\mathbf{x}_n}(\mathbf{x}_n)$ and $Q_{\mathbf{w}_m}(\mathbf{w}_m)$ as before and, additionally, $Q_{q_{nm}}(q_{nm})$. Gathering together the terms involving \mathbf{x}_n and \mathbf{w}_m , we notice that they are identical to the terms in our real-valued model but with y_{nm} replaced by q_{nm} , which has a variance of 1 rather than τ^{-1} . Therefore, we already know what the respective variational distributions will be:

$$\begin{aligned} Q_{\mathbf{x}_n}(\mathbf{x}_n) &= \mathcal{N}(\boldsymbol{\mu}_{\mathbf{x}_n}, \boldsymbol{\Sigma}_{\mathbf{x}_n}) \\ \boldsymbol{\Sigma}_{\mathbf{x}_n} &= \left[\mathbf{I}_D + \sum_m z_{nm} \langle \mathbf{w}_m \mathbf{w}_m^\top \rangle \right]^{-1} \\ \boldsymbol{\mu}_{\mathbf{x}_n} &= \boldsymbol{\Sigma}_{\mathbf{x}_n} \sum_m z_{nm} \langle q_{nm} \rangle \langle \mathbf{w}_m \rangle \end{aligned}$$

$$Q_{\mathbf{w}_m}(\mathbf{w}_m) = \mathcal{N}(\mu_{\mathbf{w}_m}, \Sigma_{\mathbf{w}_m})$$

$$\Sigma_{\mathbf{w}_m} = \left[\mathbf{I}_D + \sum_n z_{nm} \langle \mathbf{x}_n \mathbf{x}_n^\top \rangle \right]^{-1}$$

$$\mu_{\mathbf{w}_m} = \Sigma_{\mathbf{w}_m} \sum_n z_{nm} \langle q_{nm} \rangle \langle \mathbf{x}_n \rangle.$$

For $Q_{q_{nm}}(q_{nm})$ we need to do a bit of work. Recalling Equation 7.9, we know that $Q_{q_{nm}}(q_{nm})$ will be given by:

$$Q_{q_{nm}}(q_{nm}) \propto \exp(\mathbf{E}_{Q_{\mathbf{x}_n}(\mathbf{x}_n) Q_{\mathbf{w}_m}(\mathbf{w}_m)} \{\log p(y_{nm}|q_{nm}) + \log p(q_{nm}|\mathbf{w}_m, \mathbf{x}_n)\})$$

Isolating terms only involving q_{nm} , we have:

$$Q_{q_{nm}}(q_{nm}) \propto p(y_{nm}|q_{nm}) \exp \left\{ -\frac{1}{2}(q_{nm}^2 - 2q_{nm} \langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle) \right\}$$

which is $p(y_{nm}|q_{nm})$ multiplied by a Gaussian:

$$Q_{q_{nm}}(q_{nm}) \propto p(y_{nm}|q_{nm}) \mathcal{N}(\langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle, 1).$$

For the time being, we shall assume that $y_{nm} = 1$. Therefore, we have:

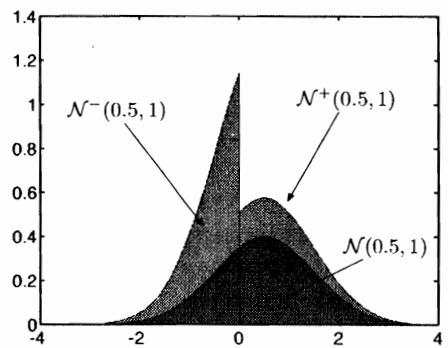
$$Q_{q_{nm}}(q_{nm}) \propto \delta(q_{nm} > 0) \mathcal{N}(\langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle, 1)$$

$$= \mathcal{N}^+(\langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle, 1)$$

where $\mathcal{N}^+(\cdot)$ is used to denote a Gaussian *truncated* (see Comment 7.3) such that q_{nm} must be positive. If $y_{nm} = -1$ we end up with $\mathcal{N}^-(\langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle, 1)$ – a Gaussian with the same mean truncated such that q_{nm} must be negative.

Comment 7.3 – Truncated Gaussian densities: A truncated Gaussian density is a Gaussian density with an additional restriction placed on the random variable. We will only be interested in Gaussian densities truncated above or below the origin.

The figure on the right shows a standard Gaussian density (with mean 0.5 and variance 1) as well as the positively and negatively truncated densities. The truncated densities have the same shape as the standard density but are both higher. This is because they must still integrate to 1 over their reduced range.



Comment 7.3 - Truncated Gaussian densities (continued): Sampling from a truncated Gaussian is fairly straightforward – one can just sample from the un-truncated density and throw away samples that do not fulfil the necessary constraint. The expected values of positively and negatively truncated Gaussians are given by:

$$p(x) = \mathcal{N}^+(\mu, \sigma^2), \quad \langle x \rangle = \mu + \frac{\mathcal{N}_{\mu/\sigma}(0, 1)}{1 - \phi(-\mu/\sigma)}$$

$$p(x) = \mathcal{N}^-(\mu, \sigma^2), \quad \langle x \rangle = \mu - \frac{\mathcal{N}_{\mu/\sigma}(0, 1)}{\phi(-\mu/\sigma)}$$

where $\mathcal{N}_a(0, 1)$ is the standard Gaussian pdf evaluated at a and $\phi(a)$ is the standard normal cdf function evaluated at a .

To compute $Q_{\mathbf{x}_n}(\mathbf{x}_n)$ and $Q_{\mathbf{w}_m}(\mathbf{w}_m)$, we need $\langle q_{nm} \rangle$. This is the expected value of a Gaussian truncated to be positive or negative (depending on the value of y_{nm}). General expressions to compute this are given in Comment 7.3. Defining $\mu_{nm} = \langle \mathbf{w}_m \rangle^\top \langle \mathbf{x}_n \rangle$ and $\sigma = 1$, these are:

$$y_{nm} = 1 : \quad \langle q_{nm} \rangle = \mu_{nm} + \frac{\mathcal{N}_{\mu_{nm}}(0, 1)}{1 - \phi(-\mu_{nm})}$$

$$y_{nm} = -1 : \quad \langle q_{nm} \rangle = \mu_{nm} - \frac{\mathcal{N}_{\mu_{nm}}(0, 1)}{\phi(-\mu_{nm})}.$$

This completes the expressions required for the VB algorithm. In the next section, we shall show an example of the algorithm in operation.

7.7.2 Visualising parliamentary data

The motivation for developing this model was the vote data for members of parliament. We shall look at data for the UK parliament that sat between 2005 and 2010. To show the benefit of sensibly handling the missing values *and* using a suitable likelihood, we will compare the model with the simplest approach we could use for visualising this data – standard, nonprobabilistic principal components where we will use a value of zero for the missing values (i.e. a value halfway between the possible votes, ± 1) and not make any concessions to the fact that the data are not real-valued.

To get a feel for the complexity of the problem, the dataset consists of voting records for some 657 MPs and 1288 divisions (votes). The average number of votes attended per MP is 853 (66%). The most active MP voted 1237 times (96%) and the least active 20 times (1.6%).

The result of running standard PCA on the data can be seen in Figure 7.9(a). Clear cluster structure is present in the latent space. In Figure 7.9(b) we label the MPs on the plot by their party affiliation and it becomes clear that the cluster structure corresponds to the three main political parties (Labour, Conservative and Liberal Democrats). The cluster structure present is not surprising as MPs often vote according to party affiliations. However, it is reassuring that it appears clearly in the first two principal components.

Some MPs seem to also be being pulled towards the origin. This could be interpreted as a measure of rebelliousness – these MPs do not vote as often along party

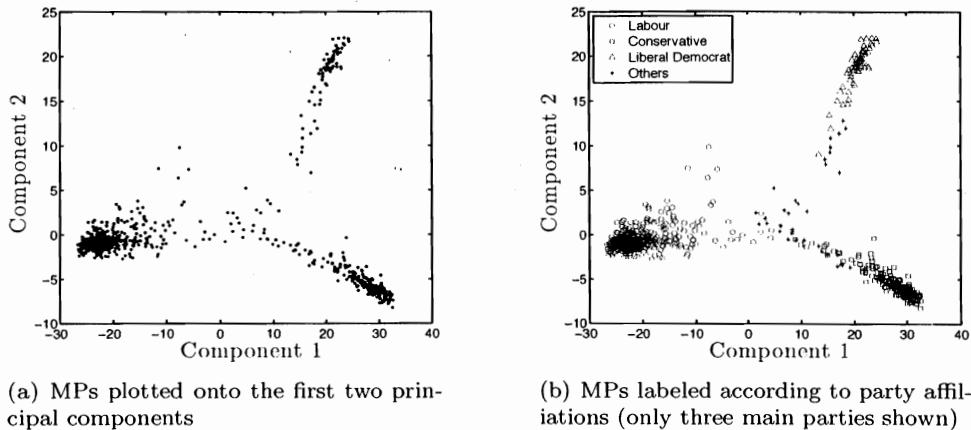


FIGURE 7.9: Standard principal components visualisation of the 2005 MP voting data. Each point corresponds to an MP.

lines. However, unfortunately, what it is showing is that these MPs simply do not vote very often. To illustrate this, we can plot the number of votes made against distance from the origin in this PCA plot, as shown in Figure 7.10. It is clear that the large number of missing values is having an unhelpful effect on the analysis – position in the latent space is a function of both political preference *and* attendance.

The results of using the VB binary PCA algorithm can be seen in Figure 7.11(a) (MATLAB script: `mpvis.m`). This plot shows $\langle \mathbf{x}_n \rangle$ and cluster structure is again clearly present. In Figure 7.11(b) we can see the MPs labeled according to party

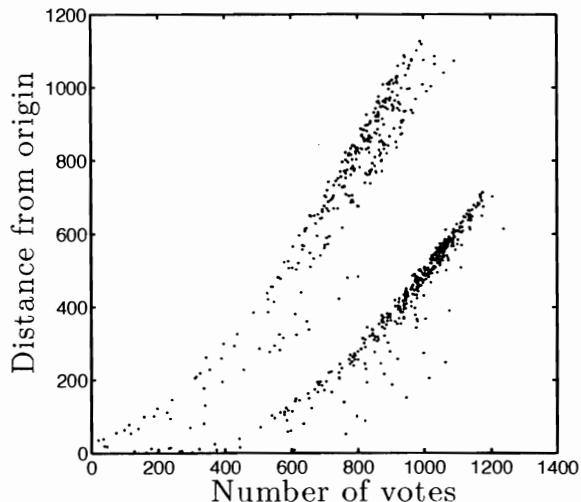


FIGURE 7.10: Number of votes cast versus distance from the origin (of the PCA plot).

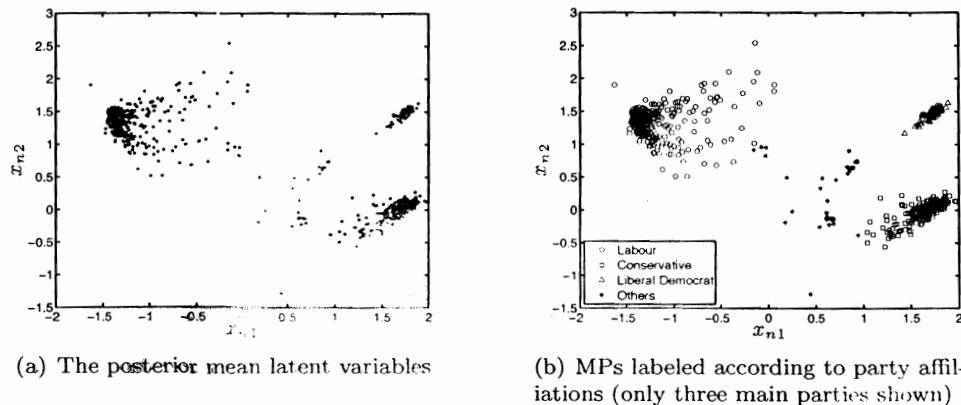


FIGURE 7.11: Probabilistic binary principal components visualisation of the 2005 MP voting data. Each point corresponds to an MP.

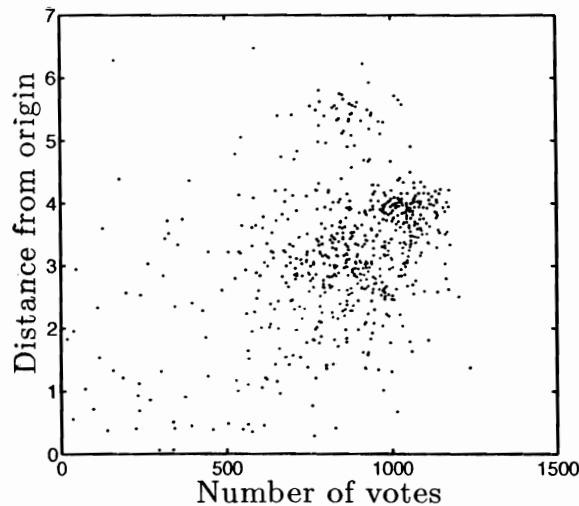


FIGURE 7.12: Number of votes cast versus distance from the origin for the probabilistic binary PCA.

affiliations. Once again, the cluster structure corresponds to the different political parties. Because we are modelling the missing values correctly, we no longer get the pull towards the origin. To demonstrate this, Figure 7.12 shows the number of votes cast versus distance from the origin – the very clear relationships present in Figure 7.10 are no longer so pronounced. The variations we can see in Figure 7.11(a) show political tendencies and not attendance tendencies.

This point is well illustrated by considering some of the smaller parties within the parliament. Figure 7.13 highlights the position of four small parties – the Democratic Unionist Party (DUP), Plaid Cymru (PC), Scottish National Party (SNP) and the Social Democratic and Labour Party (SDLP). In the traditional PCA analysis, it

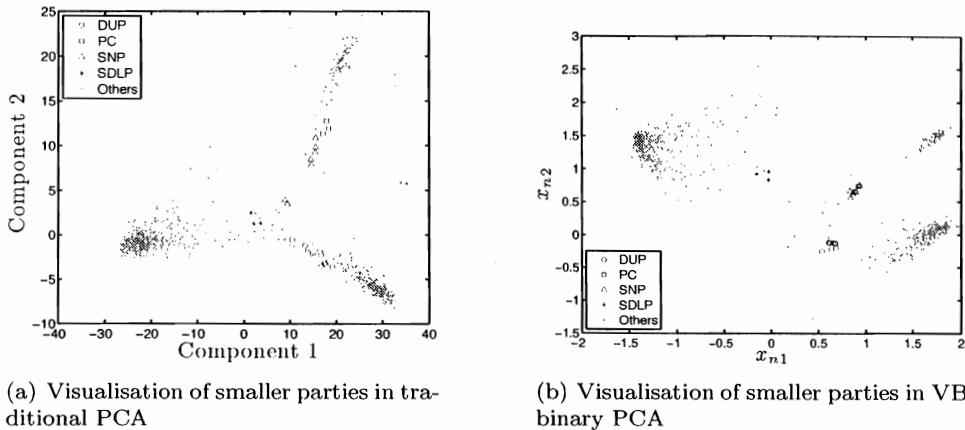


FIGURE 7.13: Visualisation of the small parties using the two PCA methods.

looks like the DUP members' votes sit within the cluster of Conservative MPs (see Figure 7.9(b)) and, to a lesser extent, PC and SNP members sit with the Liberal Democrats. However, comparing with the output of the binary PCA algorithm, we can see that the DUP form their own coherent cluster, away from the Conservatives, whilst the SNP and PC members form a very tight cluster of their own. It looks like the position of these groups in the original PCA was heavily influenced by the poor modelling of the missing data.

Finally, because of the way we have chosen to model the missing values, we have an individual covariance matrix Σ_{x_n} for each MP. In Figure 7.14 we visualise, with ellipses, the covariance matrices of the 20 MPs for which the model is least certain. These MPs tend to be those who cast the least votes. It is clear that there is no real

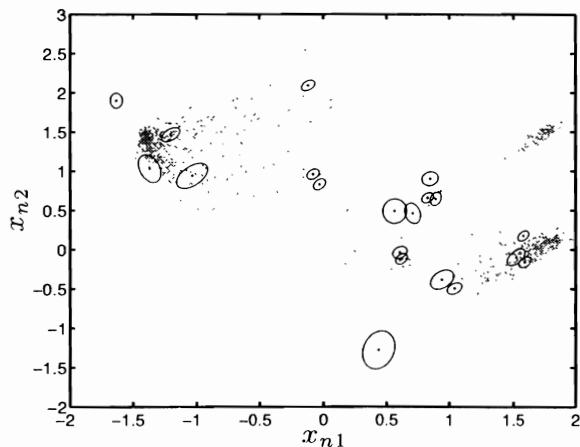


FIGURE 7.14: Covariance matrix visualisation for the 20 MPs corresponding to the highest uncertainty.

pattern to where they are located – the model does not pull MPs who do not vote often towards the origin.

Much more interesting analysis could be done with this data but is beyond the scope of this book. The important point is that a model based on sensible assumptions that can correctly handle missing values (the binary probabilistic PCA) is likely to be able to give us more insight into interesting variability amongst the MPs than using basic PCA.

7.7.2.1 Aside – relationship to classification

Before we finish, it is worth trying to get an intuitive feel for how this model works. On the face of it it appears rather complex but perhaps the easiest way to look at it is as a classification model. The training data consists of labels for M classification tasks (one for each vote) but no input features. The model infers a set of latent observations (the \mathbf{x}_n) and M classification functions (defined by the \mathbf{w}_m) such that we can satisfy as many of the classification labels as possible. Figure 7.15 shows four example votes (input features) and the corresponding decision boundary

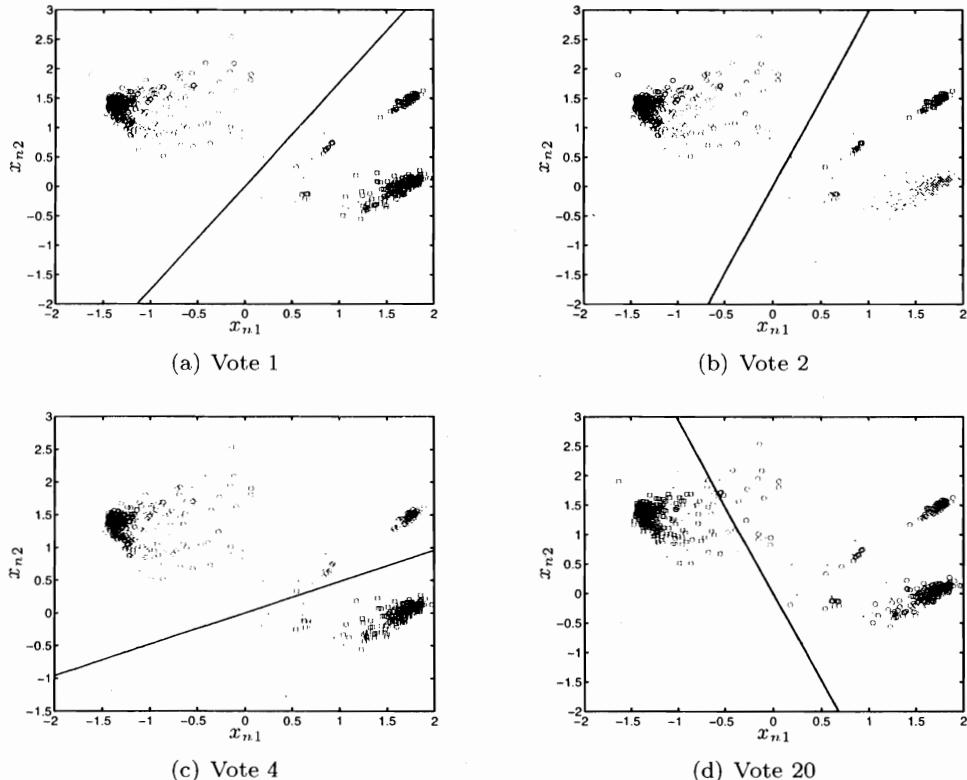


FIGURE 7.15: Example of four of the votes – each MP is displayed as a circle or square depending on how they voted (light grey vote if they didn't vote).

in the latent space. MPs are plotted at their posterior mean position and labeled according to how they voted (circle/square for ± 1 , light grey dot for missing value). The model has positioned the MPs in the latent space and constructed decision boundaries in such a way as to enable as many of the classification labels to be satisfied as possible. It is not always possible to satisfy all of the labels – see, for example, the circles to the right of the boundary in vote 1.

7.8 Summary

In this chapter, we have used principal components analysis (PCA) and some probabilistic variants to introduce the concepts of latent variable models and the inference technique variational Bayes. There are many other latent variable models being used for diverse applications – the field of Information Retrieval is particularly full of them – and our hope is that the techniques introduced here will enable the reader to understand some of these more application-specific models.

Variational Bayes (VB) is an increasingly popular inference technique within the field of Machine Learning. With any posterior approximation technique we are making a tradeoff between accuracy of approximation and computational difficulty. The empirical evidence suggests that VB finds a good balance between tractability and accuracy. It is important to remember that other methods can be used for performing inference in latent variable models – we saw the EM algorithm in Chapter 6 being used for a mixture model. Also, the use of the auxiliary variable trick with the probit likelihood is certainly not the only way we can overcome tricky binary likelihoods – we saw alternatives in Chapter 4.

7.9 Exercises

- EX 7.1. Show that the bound given in Equation 7.6 is maximised (i.e. equal to the true log marginal likelihood) when $Q(\boldsymbol{\theta})$ is identical to the true posterior $p(\boldsymbol{\theta}|\mathbf{X})$.
- EX 7.2. Compute each term in the lower bound, $\mathcal{L}(\boldsymbol{\theta})$, for the probabilistic PCA model given in Section 7.5.
- EX 7.3. Compute the components of the variational posterior for the probabilistic PCA model with missing values described by Equation 7.15.

Further reading

- [1] Christopher M. Bishop. Variational principal components. In *Proceedings Ninth International Conference on Artificial Neural Networks, ICANN'99*, pages 509–514, 1999.

An example of a probabilistic principal components model where the inference is performed using variational Bayes. A prior is used to encourage sparsity in the latent dimensions which goes some way towards avoiding having to choose the size of the latent space.
- [2] I.T. Jolliffe. *Principal Component Analysis*. Springer, second edition, 2002.

A comprehensive textbook on principal components analysis.
- [3] Michael Jordan, Z. Ghahramani, T.S. Jaakkola, and L.K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999.
- [4] Arto Klami and Samuel Kaski. Probabilistic approaches to detecting dependencies between data sets. *Neurocomputing*, 72:39–46, 2008.

An interesting latent variable model that extends the ideas of probabilistic PCA into the scenario where two data sets are being analysed together. Also provides an example of the EM algorithm and variational Bayes.
- [5] S. Roweis and Z. Ghahramani. A unifying review of linear Gaussian models. *Neural Computation*, 11(2):305–345, 1999.

An excellent, accessible review of linear Gaussian models giving many examples.
- [6] Michael Tipping. Probabilistic visualisation of high-dimensional binary data. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pages 592–598, Cambridge, MA, USA, 1999. MIT Press.

A probabilistic binary PCA algorithm using the logistic likelihood we saw in Chapter 4 and an EM-like inference algorithm.
- [7] Michael Tipping and Christopher Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 61(3):611–622, 1999.

An interesting application of maximum likelihood. Here it is applied to one of the first probabilistic approaches to the the classical statistical problem of principal components analysis.

Glossary

Analytical solution An analytical solution to a particular mathematical problem (e.g. optimising a quantity or evaluating an integral) is one in which the solution can be obtained exactly. Many of the problems that we will deal with will not have analytical solutions, necessitating the use of iterative algorithms or sampling techniques.

Biased An estimator (e.g. $\widehat{\sigma^2}$ in Chapter 2) is said to be biased if, on average, it does not equal the true value.

Binomial distribution A popular probability distribution that describes the number of successes in a set of binary trials.

Burn-in When generating samples using MCMC, it is common to throw away the first N as the algorithm may not have converged and hence these are not representative. Determining N is not straightforward.

Conditional independence Two (or more) random variables A and B are said to be conditionally independent if their joint distribution, conditioned on C , can be factorised as $P(A, B \mid C) = P(A \mid C)P(B \mid C)$. Conditional independence does not imply unconditional independence.

Conditional probabilities Conditional probabilities are used to describe the probability of events that depend on the outcome of other events. For example, if the value of the random variable A depends upon the value of the random variable B , we can write the probability of A given the value of C as $P(A \mid C)$.

Conjugate A prior and likelihood are said to be conjugate if they result in a posterior of the same form as the prior.

Continuous random variables Random variables defined on a sample space that cannot be systematically enumerated. For example, random variables defined over all real numbers.

Convergence (sampler) A sampler is said to have converged when the samples it is generating are all coming from the same distribution. Before the sampler has converged, the samples should not be used.

Covariance Covariance is the generalisation of variance to the distributions over several variables. The covariance matrix describes how the different variables co-vary – how they are related.

Cross-validation A technique used for validation and model selection. The data is randomly partitioned into K groups. The model is then trained K times, each time with one of the groups left out.

Decision boundary A line separating two classes in a classification problem.

Deterministic Something that is not random. For example, our model in Chapter 1, $t = \mathbf{w}^T \mathbf{x}$, is deterministic. The same value of \mathbf{x} will always give the same value of t .

Discrete random variables Random variables defined over a sample space that can be systematically enumerated.

Discriminative classifier A classifier that explicitly defines (and optimises) decision boundaries between the classes.

Expectation For a (discrete) random variable, X , the expected value of some function of X , $f(X)$, is defined as:

$$\mathbf{E}_{p(X)} \{f(X)\} = \sum_x P(x)f(x).$$

This can be thought of as an average weighted by how likely the different values of X are. For continuous random variables, the summation is exchanged for an integral.

Feature selection In some classification problems it is useful to reduce the number of attributes. This process is known as feature selection. Common techniques for feature selection are scoring functions (pick the attributes/features) with the highest scores, clustering (cluster the attributes and use the cluster means as the new attributes) and projection techniques such as principal components analysis.

Fisher information The Fisher information is a measure of how much information a random variable provides about a particular model parameter.

Function A way of defining a relationship between two or more variables. For example,

$$t = f(x)$$

tells us that t depends on x – if we know x we can compute t .

Generalisation Generalisation is the ability to take something that has been learnt from one set of objects and apply it to previously unseen objects. For example, our Olympic model in Chapter 1 is generalising well if it makes good predictions for future Olympic sprints. In other words, an algorithm that exhibits good generalisation performance is one that is able to make good predictions on previously unseen data.

Generative model A generative model defines a process that could have generated the observed data. Thinking in terms of potential generative processes is often a useful abstraction when building models.

Global optimum For a function that can have many maxima (or minima), the global optimum is described as the highest (or lowest).

Graphical model A graphical representation of a probability distribution in which nodes correspond to random variables and directed edges to dependency relationships.

Hessian matrix The matrix of second derivatives of a function with respect to each pair of variables. Developed and named after Ludwig Otto Hesse, a 19th century German mathematician.

Hyper-parameter A parameter controlling the prior over another parameter in an hierarchical Bayesian model.

Information theory The quantitative study of information. In particular, the information content of a random variable is linked to its probability distribution. A distribution that is very uncertain has a high information content.

Joint probability The joint probability of two random variables A and B is the probability that they each take a specific value. For example, the probability that A takes value a and B takes value b . This is written as $P(A = a, B = b)$.

Likelihood The value of the density function (or distribution if the data are discrete) of the data, conditioned on any model parameters, evaluated at the data. This is a single numerical value, which is optimised with respect to the model parameters to produce the maximum likelihood solution.

Linear A function $t = f(x)$ is said to be linear if it satisfies the following conditions:

$$\begin{aligned}f(x_1 + x_2) &= f(x_1) + f(x_2) \\f(ax) &= af(x)\end{aligned}$$

A common example is $f(x) = wx$.

Mahanalobis distance The Mahanalobis distance between two objects \mathbf{x}_n and \mathbf{x}_m is defined as:

$$(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{A} (\mathbf{x}_i - \mathbf{x}_j).$$

Substituting $\mathbf{A} = \mathbf{I}$, we recover the standard squared Euclidean distance. The matrix \mathbf{A} creates a warping of the space such that distances are not the same in all directions. The set of points that have a particular squared Euclidean distance away from, say, \mathbf{x}_n , form a circle. The set of points a certain Mahanalobis distance away from \mathbf{x}_n form an ellipse, the shape of which is defined by \mathbf{A} .

Marginal likelihood The denominator of Bayes' rule. A useful quantity for model comparison and choice.

Marginalisation The act of removing a random variable from a joint distribution by summing (or integrating if it is continuous) the joint distribution over all possible values that the random variable can take. For example:

$$P(A = a) = \sum_b P(A = a, B = b).$$

Maximum likelihood A popular parameter estimation scheme, where parameters are chosen that maximise the likelihood of the observed data.

Maximum a posteriori A popular way of choosing point estimates for parameter values that extends maximum likelihood by adding a regularising prior term.

Metropolis–Hastings A popular algorithm for generating samples from a density that does not require evaluation of the normalising constant.

Model complexity A term used to describe how complex a model is. For example, $t = w_0 + w_1x$ is less complex than $t = w_0 + w_1x + w_2x^2$ and as such, is not able to find as complex patterns in data.

Model selection Model selection is the task of selecting which model to use for a particular task. The model choices could all come from the same family, although they don't have to. For example, if we wish to use a polynomial function $t = \sum_{k=0}^K w_k x^k$, choosing a suitable value for K is a model selection problem.

Model A mathematical description of a process. For example, in Chapter 1 we proposed the model $t = w_0 + w_1x$ to represent the winning time in a 100 m sprint in Olympic year x .

Mode The mode of a distribution over some random variable is the most likely value.

Monotonic function A monotonic function is one that increases or decreases indefinitely. A common example is $\log(x)$ that always increases as x increases. This has the useful property that the value of x that minimises $f(x)$ will also minimise $\log(f(x))$.

Monte Carlo approximation An approximation to an expectation performed by drawing samples from the appropriate distribution. An expectation of the form:

$$\mathbf{E}_{p(x)} \{f(x)\} = \int f(x)p(x) dx$$

is approximated by:

$$\mathbf{E}_{p(x)} \{f(x)\} \approx \frac{1}{S} \sum_{s=1}^S f(x^s),$$

where x^1, \dots, x^S are S samples from $p(x)$.

Multinomial distribution A popular distribution over vectors of integers. For example, if I role a die N times and record the number of times I obtain each face value in a six-dimensional vector, this vector could be modelled as a random variable with a multinomial distribution.

Natural logarithm The logarithm to the base e , referred to here as \log but often referred to as \ln .

Noise Variability in data that is assumed to be not of interest for the problem at hand. For example, random fluctuations brought about by measurement error.

Over-fitting A model is said to be over-fitting if it is too complex and is using its surplus complexity to fit to noise. Over-fitted models usually generalise badly.

Parameters Variables used to define a model. For example, the model

$$t = w_0 + w_1x$$

has two parameters – w_0 and w_1 .

Partial derivatives Taking partial derivatives of a function of several variables involves differentiation with respect to each variable whilst treating other variables as constant. For example, if the function $t = f(x, y)$ is defined as

$$t = 2x^2 + 3y^3 + xy,$$

the partial derivatives with respect to x and y are:

$$\begin{aligned}\frac{\partial f(x, y)}{\partial x} &= 4x + y \\ \frac{\partial f(x, y)}{\partial y} &= 9y^2 + x\end{aligned}$$

Plate In graphical models, a shorthand used to show that there are several instances of a particular type of random variables.

Polynomial A polynomial function $t = f(x)$ has the form $t = \sum_{k=0}^K w_k x^k$. Common examples are the first order (or linear) polynomial $t = w_0 + w_1 x = \sum_{k=0}^1 w_k x^k$ (called first order because the highest power to which x is raised is 1) and quadratic (second order) polynomial $t = w_0 + w_1 x + w_2 x^2 = \sum_{k=0}^2 w_k x^k$. Note that $x^0 = 1$.

Posterior distribution The posterior distribution is the distribution over our parameter values after we have observed some data.

Precision In hierarchical Bayesian models it is often convenient to work with the precision rather than the variance. The precision is defined as:

$$\tau = \frac{1}{\sigma^2}.$$

Hence a Gaussian with mean μ and variance σ^2 can also be represented using precision τ as:

$$\mathcal{N}(\mu, \tau^{-1}).$$

Prior distributions Distributions describing our knowledge parameter values before any data has been observed.

Probability density function A probability density function (pdf) describes how the probability mass of a continuous random variable is distributed across its sample space. Probability density functions must always be positive and the integral of the probability density function over the sample space must be equal to 1.

Probability distribution A function or set of values that describes the characteristics of a random variable.

Probability The probability of an event taking place is a number between 0 and 1 that describes how likely the event is to take place.

Projection algorithms A family of Machine Learning algorithms that project data from M dimensions into D dimensions ($D \ll M$). Projection techniques are useful for visualisation (with $D = 2$) and can also be used for data pre-processing for, for example, classification.

Quadratic A quadratic function $t = f(x)$ is a polynomial function where the highest power to which x is raised is 2. For example, $t = x^2$ and $t = w_0 + w_1x + w_2x^2$ are both quadratic functions.

Random events Events for which we cannot (or do not want or need to) define a deterministic model. For example, rolling a die or tossing a coin. Although we do not know the outcome of such events, it is likely that we will know the relative likelihoods of different outcomes.

Random variable A variable that stores the result of a random event. For example, if we toss a coin and assign the variable X the value 1 if the coin lands with the heads face up and 0 if it lands with the tails face up is a random variable.

Random walk A sequence of samples where each depends on the previous one.

Regularisation Regularisation is the act of placing restrictions on parameter values to limit the maximum complexity of a model.

Sample space The space of possible values that can be taken by a random variable.
In other words, the set of the possible outcomes of a particular random event.

Statistics Statistics describes the collection of techniques and principles concerning the collection and interpretation of data.

Supervised learning Machine learning tasks where one is provided with a set of data objects and some associated labels.

Symmetric matrix A square matrix \mathbf{X} is symmetric if $x_{ij} = x_{ji}$ for all j, i . If this is the case, then it follows that $\mathbf{X}^\top = \mathbf{X}$.

Unbiased An estimator (for example, $\hat{\mathbf{w}}$) is said to be unbiased if, on average, its value is equal to the true value.

Unsupervised learning Learning algorithms that do not require targets or labels.
Examples include clustering and projection.

Validation data Data that is used to help choose model type and parameters that is not directly used to train the model.

Variance Variance is the expected squared difference between the random variable and its mean.

Index

- absolute loss, 5
- attributes, 1, 84, 208
- auxiliary variables, 265
- bag-of-words, 176
- Bayes' rule, 49, 98, 120, 140, 170
- Bayesian classifier, 170
- Bayesian inference, 139
- Bayesian Machine Learning, 98
- Bernoulli distribution, 53, 230
- beta distribution, 60, 100
- bias-variance trade-off, 75
- Binomial distribution, 53, 95
- causality, 2
- chain rule (differentiation), 146
- classification, 140, 169
 - discriminative versus generative, 203
 - non-probabilistic, 183
 - probabilistic, 170
 - text, 175
- classification accuracy, 198
- clustering, 207
 - similarity measures, 209
- combinations, 55
- confusion matrix, 201
- conjugate prior, 102
 - non-conjugate models, 139
- covariance, 52, 78
 - Gaussian, 62
- cross-validation, 29, 131, 185, 196, 228
 - computational scaling, 32
 - leave-one-out, 31
- decision boundary, 147
- definite integrals, 57
- dependence, 46
- Dirichlet distribution, 178
- Eigenvectors and eigenvalues, 244
- evidence, 105
- expectation
 - with respect to posterior, 109
- expectations, 50
 - continuous, 58
 - for predictions, 98, 152
 - with respect to posterior, 129
- Fisher information, 80
- function, 2
 - linear, 1
 - polynomial, 25
 - quadratic, 25
- Gaussian, 61
 - likelihood, 124
 - noise, 66, 69
 - process, 182
 - truncated, 267
- generalisation, 28, 34, 74, 75, 196
- generative model, 40, 216
- graphical models, 120, 253
 - plates, 121
- hyper-parameters, 119
- independence, 46
 - in Variational Bayes, 251
 - multivariate Gaussian, 64
- information theory, 80
- Jensen's inequality, 219, 250
- K-means, 208
- K-nearest neighbours, 183
- kernel density estimation, 163
- kernel K-means, 212
- kernel KNN, 196
- kernel methods, 186, 193, 212
- Kullback-Leibler divergence, 251

- Lagrange multipliers, 188, 223
- Laplace approximation, 149
 - for logistic regression, 151
- latent variables, 248
- likelihood, 67
 - binary, 142
 - classification, 171
 - in Bayes' rule, 99
 - log, 69
- linear
 - nonlinear responses, 25
- linear model, 85
- linear modelling, 1, 25
- logistic regression, 179
- margin, 186
 - maximisation, 187
 - soft, 192
- marginal distribution, 101
- marginal likelihood, 101, 117, 141, 171, 249
- matrix, 16
 - determinant, 64
 - Fisher information, 80
 - Hessian, 72, 80, 144
 - identity, 21
 - inversion, 22
 - multiplication, 18
 - notation, 15
 - symmetric, 71
 - trace, 88
 - transpose, 18
- maximum likelihood, 69
 - bias of estimator, 86, 88
 - bias of variance estimate, 82
- maximum-a-posteriori, 126, 143, 178, 232
- Metropolis-Hastings, 154
- minimum loss, 6
 - equivalence to Gaussian ML, 70
- missing data, 260
- mixture model
 - likelihood, 217
- mixture models, 207, 215
 - Bayesian treatment, 233
- model assumptions, 3
- model complexity, 33, 196
- model selection, 25
 - difficulty, 31
- K-means, 210
- via marginal likelihood, 117
- with likelihood, 74
- with loss, 28
- Monte-Carlo, 58
- mRNA data, 208
- multinomial distribution, 54, 177
- multivariate Gaussian, 62
 - covariance, 62, 78
 - independence, 62
- Naive Bayes, 175
- Naive Bayes classifier, 175
- Newton-Raphson, 144
- noise, 39, 76, 82, 85
 - additive, 66
 - Gaussian, 85
- nonlinear responses, 27
- normal, *see* Gaussian
- over-fitting, 28, 33, 34, 74, 75, 196, 228
- parameter, 2
- point predictions, 12, 110
- posterior approximation
 - Laplace, 149
 - sampling, 156, 163
- posterior distribution, 101
 - exact computation, 103, 120
 - expectation with respect to, 109
 - sampling from, 127
- predictions, 1
 - uncertainty, 84, 85
- Principal Components Analysis, 242
- prior distribution, 75, 99
 - choice, 111
 - conjugate, 139, 173
 - strength, 113, 116
- probability, 39
 - conditional, 44
 - joint, 45
- probit, 265
- projection, 239
- random variable, 41
 - continuous, 42, 55
 - density, 55
 - discrete, 41
 - distributions, 42
 - marginalisation, 47

marginalisation, continuous, 58
vectors, 52

regression
 logistic, 179

regularisation, 33, 75

ROC analysis, 199
 AUC, 200

sampling, 59, 153, 154
 burn-in, 161
 convergence, 161
 from posterior, 153
 visualising output, 163

sensitivity and specificity, 198

sigmoid, 142

smoothing, 177

squared loss, 4
 matrix form, 19
 minimising, 6

Support Vector Machines, 186

Taylor expansion, 150

turning points, 6

uncertainty, 48
 in parameters, 39, 76, 78, 80, 82,
 148
 in predictions, 39, 83, 85, 152

uniform distribution, 58

validation, 29

variance, 51
 reduction in posterior, 105

Variational Bayes, 249

vector, 16
 differentiation with respect to, 20
 indexing, 17
 inner product, 18
 transpose, 16

