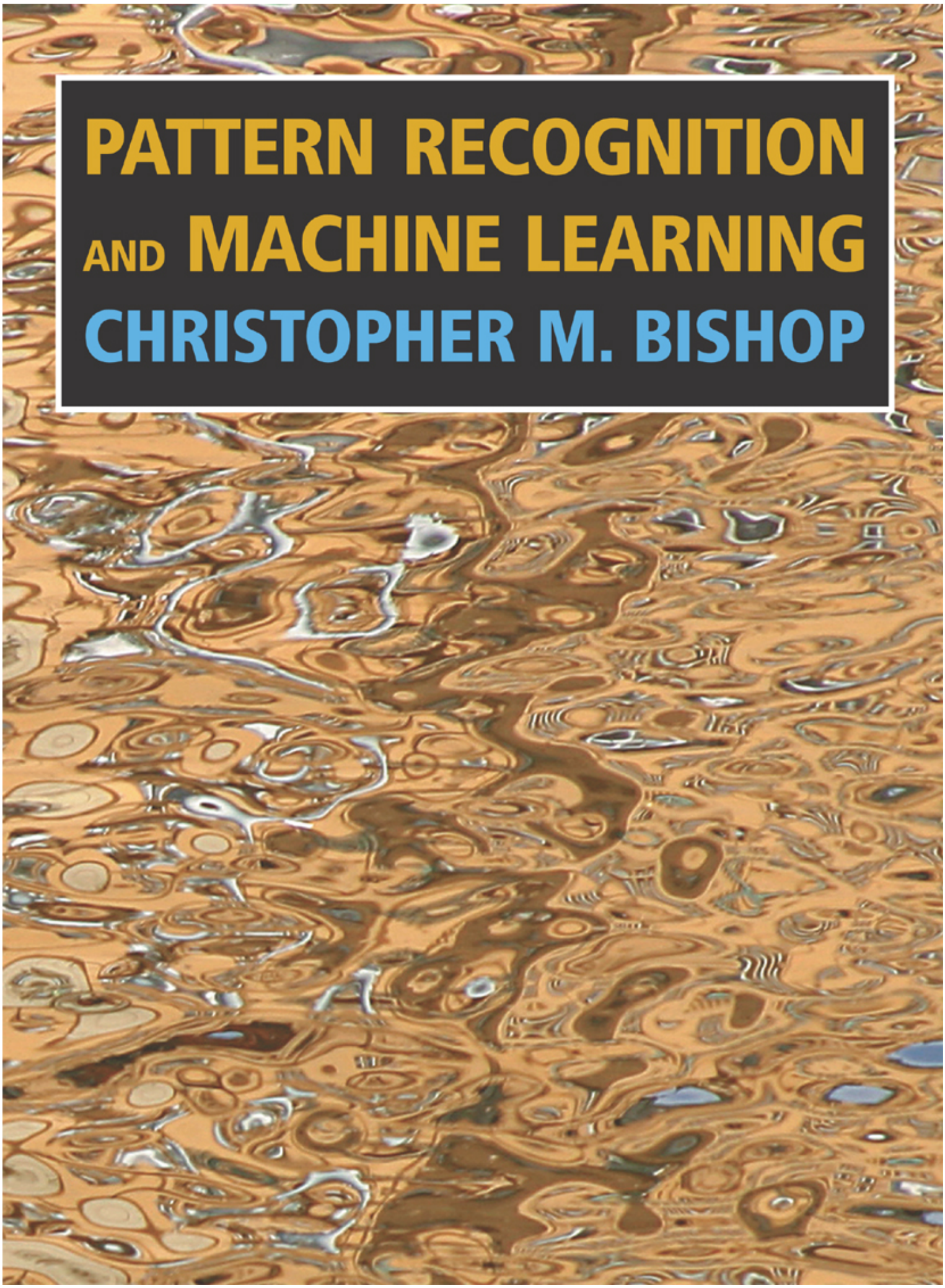

PATTERN RECOGNITION
AND MACHINE LEARNING
CHRISTOPHER M. BISHOP

Information Science and Statistics

Information Science and Statistics

Akaike and Kitagawa: The Practice of Time Series Analysis.
Bishop: Pattern Recognition and Machine Learning.
Cowell, Dawid, Lauritzen, and Spiegelhalter: Probabilistic Networks and Expert Systems.
Doucet, de Freitas, and Gordon: Sequential Monte Carlo Methods in Practice.
Fine: Feedforward Neural Network Methodology.
Hawkins and Olwell: Cumulative Sum Charts and Charting for Quality Improvement.
Jensen: Bayesian Networks and Decision Graphs.
Marchette: Computer Intrusion Detection and Network Monitoring: A Statistical Viewpoint.
Rubinstein and Kroese: The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte Carlo Simulation, and Machine Learning.
Studený: Probabilistic Conditional Independence Structures.
Vapnik: The Nature of Statistical Learning Theory, Second Edition.
Wallace: Statistical and Inductive Inference by Minimum Massage Length.

Christopher M. Bishop

Pattern Recognition and Machine Learning

Christopher M. Bishop F.R.Eng.
Assistant Director
Microsoft Research Ltd
Cambridge CB3 0FB, U.K.
cmbishop@microsoft.com
http://research.microsoft.com/~cmbishop

Library of Congress Control Number: 2006922522

ISBN-10: 0-387-31073-8
ISBN-13: 978-0387-31073-2

Printed on acid-free paper.

Printed in Singapore. (KYO)

9 8 7 6 5 4 3 2 1

springer.com

This book is dedicated to my family:

Jenna, Mark, and Hugh

Total eclipse of the sun, Antalya, Turkey, 29 March 2006.

Preface

Pattern recognition has its origins in engineering, whereas machine learning grew out of computer science. However, these activities can be viewed as two facets of the same field, and together they have undergone substantial development over the past ten years. In particular, Bayesian methods have grown from a specialist niche to become mainstream, while graphical models have emerged as a general framework for describing and applying probabilistic models. Also, the practical applicability of Bayesian methods has been greatly enhanced through the development of a range of approximate inference algorithms such as variational Bayes and expectation propagation. Similarly, new models based on kernels have had significant impact on both algorithms and applications.

This new textbook reflects these recent developments while providing a comprehensive introduction to the fields of pattern recognition and machine learning. It is aimed at advanced undergraduates or first year PhD students, as well as researchers and practitioners, and assumes no previous knowledge of pattern recognition or machine learning concepts. Knowledge of multivariate calculus and basic linear algebra is required, and some familiarity with probabilities would be helpful though not essential as the book includes a self-contained introduction to basic probability theory.

Because this book has broad scope, it is impossible to provide a complete list of references, and in particular no attempt has been made to provide accurate historical attribution of ideas. Instead, the aim has been to give references that offer greater detail than is possible here and that hopefully provide entry points into what, in some cases, is a very extensive literature. For this reason, the references are often to more recent textbooks and review articles rather than to original sources.

The book is supported by a great deal of additional material, including lecture slides as well as the complete set of figures used in the book, and the reader is encouraged to visit the book web site for the latest information:

http://research.microsoft.com/∼cmbishop/PRML

Exercises

The exercises that appear at the end of every chapter form an important component of the book. Each exercise has been carefully chosen to reinforce concepts explained in the text or to develop and generalize them in significant ways, and each is graded according to difficulty ranging from ($\star$), which denotes a simple exercise taking a few minutes to complete, through to ($\star\star\star$), which denotes a significantly more complex exercise.

It has been difficult to know to what extent these solutions should be made widely available. Those engaged in self study will find worked solutions very beneficial, whereas many course tutors request that solutions be available only via the publisher so that the exercises may be used in class. In order to try to meet these conflicting requirements, those exercises that help amplify key points in the text, or that fill in important details, have solutions that are available as a PDF file from the book web site. Such exercises are denoted by **www**. Solutions for the remaining exercises are available to course tutors by contacting the publisher (contact details are given on the book web site). Readers are strongly encouraged to work through the exercises unaided, and to turn to the solutions only as required.

Although this book focuses on concepts and principles, in a taught course the students should ideally have the opportunity to experiment with some of the key algorithms using appropriate data sets. A companion volume (Bishop and Nabney, 2008) will deal with practical aspects of pattern recognition and machine learning, and will be accompanied by Matlab software implementing most of the algorithms discussed in this book.

Acknowledgements

First of all I would like to express my sincere thanks to Markus Svensén who has provided immense help with preparation of figures and with the typesetting of the book in LaTeX. His assistance has been invaluable.

I am very grateful to Microsoft Research for providing a highly stimulating research environment and for giving me the freedom to write this book (the views and opinions expressed in this book, however, are my own and are therefore not necessarily the same as those of Microsoft or its affiliates).

Springer has provided excellent support throughout the final stages of preparation of this book, and I would like to thank my commissioning editor John Kimmel for his support and professionalism, as well as Joseph Piliero for his help in designing the cover and the text format and MaryAnn Brickner for her numerous contributions during the production phase. The inspiration for the cover design came from a discussion with Antonio Criminisi.

I also wish to thank Oxford University Press for permission to reproduce excerpts from an earlier textbook, *Neural Networks for Pattern Recognition* (Bishop, 1995a). The images of the Mark 1 perceptron and of Frank Rosenblatt are reproduced with the permission of Arvin Calspan Advanced Technology Center. I would also like to thank Asela Gunawardana for plotting the spectrogram in Figure 13.1, and Bernhard Schölkopf for permission to use his kernel PCA code to plot Figure 12.17.

Many people have helped by proofreading draft material and providing comments and suggestions, including Shivani Agarwal, Cédric Archambeau, Arik Azran, Andrew Blake, Hakan Cevikalp, Michael Fourman, Brendan Frey, Zoubin Ghahramani, Thore Graepel, Katherine Heller, Ralf Herbrich, Geoffrey Hinton, Adam Johansen, Matthew Johnson, Michael Jordan, Eva Kalyvianaki, Anitha Kannan, Julia Lasserre, David Liu, Tom Minka, Ian Nabney, Tonatiuh Pena, Yuan Qi, Sam Roweis, Balaji Sanjiya, Toby Sharp, Ana Costa e Silva, David Spiegelhalter, Jay Stokes, Tara Symeonides, Martin Szummer, Marshall Tappen, Ilkay Ulusoy, Chris Williams, John Winn, and Andrew Zisserman.

Finally, I would like to thank my wife Jenna who has been hugely supportive throughout the several years it has taken to write this book.

Chris Bishop
Cambridge
February 2006

Mathematical notation

I have tried to keep the mathematical content of the book to the minimum necessary to achieve a proper understanding of the field. However, this minimum level is nonzero, and it should be emphasized that a good grasp of calculus, linear algebra, and probability theory is essential for a clear understanding of modern pattern recognition and machine learning techniques. Nevertheless, the emphasis in this book is on conveying the underlying concepts rather than on mathematical rigour.

I have tried to use a consistent notation throughout the book, although at times this means departing from some of the conventions used in the corresponding research literature. Vectors are denoted by lower case bold Roman letters such as $\mathbf{x}$, and all vectors are assumed to be column vectors. A superscript T denotes the transpose of a matrix or vector, so that $\mathbf{x}^{\mathrm{T}}$ will be a row vector. Uppercase bold roman letters, such as $\mathbf{M}$, denote matrices. The notation $(w_1, \ldots, w_M)$ denotes a row vector with M elements, while the corresponding column vector is written as $\mathbf{w} = (w_1, \ldots, w_M)^{\mathrm{T}}$.

The notation $[a, b]$ is used to denote the *closed* interval from a to b, that is the interval including the values a and b themselves, while (a, b) denotes the corresponding *open* interval, that is the interval excluding a and b. Similarly, $[a, b)$ denotes an interval that includes a but excludes b. For the most part, however, there will be little need to dwell on such refinements as whether the end points of an interval are included or not.

The $M \times M$ identity matrix (also known as the unit matrix) is denoted $\mathbf{I}_M$, which will be abbreviated to $\mathbf{I}$ where there is no ambiguity about it dimensionality. It has elements I_{ij} that equal 1 if $i = j$ and 0 if $i \neq j$.

A functional is denoted $f[y]$ where $y(x)$ is some function. The concept of a functional is discussed in Appendix D.

The notation $g(x) = O(f(x))$ denotes that $|f(x)/g(x)|$ is bounded as $x \to \infty$. For instance if $g(x) = 3x^2 + 2$, then $g(x) = O(x^2)$.

The expectation of a function $f(x, y)$ with respect to a random variable x is denoted by $\mathbb{E}_x[f(x, y)]$. In situations where there is no ambiguity as to which variable is being averaged over, this will be simplified by omitting the suffix, for instance

$\mathbb{E}[x]$. If the distribution of x is conditioned on another variable z, then the corresponding conditional expectation will be written $\mathbb{E}_x[f(x)|z]$. Similarly, the variance is denoted $\mathrm{var}[f(x)]$, and for vector variables the covariance is written $\mathrm{cov}[\mathbf{x}, \mathbf{y}]$. We shall also use $\mathrm{cov}[\mathbf{x}]$ as a shorthand notation for $\mathrm{cov}[\mathbf{x}, \mathbf{x}]$. The concepts of expectations and covariances are introduced in Section 1.2.2.

If we have N values $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of a D-dimensional vector $\mathbf{x} = (x_1, \ldots, x_D)^{\mathrm{T}}$, we can combine the observations into a data matrix $\mathbf{X}$ in which the n^{th} row of $\mathbf{X}$ corresponds to the row vector $\mathbf{x}_n^{\mathrm{T}}$. Thus the n, i element of $\mathbf{X}$ corresponds to the i^{th} element of the n^{th} observation $\mathbf{x}_n$. For the case of one-dimensional variables we shall denote such a matrix by $\mathbf{x}$, which is a column vector whose n^{th} element is x_n. Note that $\mathbf{x}$ (which has dimensionality N) uses a different typeface to distinguish it from $\mathbf{x}$ (which has dimensionality D).

Contents

Preface vii

Mathematical notation xi

Contents xiii

1 Introduction 1
- 1.1 Example: Polynomial Curve Fitting . . . 4
- 1.2 Probability Theory . . . 12
  - 1.2.1 Probability densities . . . 17
  - 1.2.2 Expectations and covariances . . . 19
  - 1.2.3 Bayesian probabilities . . . 21
  - 1.2.4 The Gaussian distribution . . . 24
  - 1.2.5 Curve fitting re-visited . . . 28
  - 1.2.6 Bayesian curve fitting . . . 30
- 1.3 Model Selection . . . 32
- 1.4 The Curse of Dimensionality . . . 33
- 1.5 Decision Theory . . . 38
  - 1.5.1 Minimizing the misclassification rate . . . 39
  - 1.5.2 Minimizing the expected loss . . . 41
  - 1.5.3 The reject option . . . 42
  - 1.5.4 Inference and decision . . . 42
  - 1.5.5 Loss functions for regression . . . 46
- 1.6 Information Theory . . . 48
  - 1.6.1 Relative entropy and mutual information . . . 55

Exercises . . . 58

2 Probability Distributions **67**
2.1 Binary Variables . . . 68
2.1.1 The beta distribution . . . 71
2.2 Multinomial Variables . . . 74
2.2.1 The Dirichlet distribution . . . 76
2.3 The Gaussian Distribution . . . 78
2.3.1 Conditional Gaussian distributions . . . 85
2.3.2 Marginal Gaussian distributions . . . 88
2.3.3 Bayes' theorem for Gaussian variables . . . 90
2.3.4 Maximum likelihood for the Gaussian . . . 93
2.3.5 Sequential estimation . . . 94
2.3.6 Bayesian inference for the Gaussian . . . 97
2.3.7 Student's t-distribution . . . 102
2.3.8 Periodic variables . . . 105
2.3.9 Mixtures of Gaussians . . . 110
2.4 The Exponential Family . . . 113
2.4.1 Maximum likelihood and sufficient statistics . . . 116
2.4.2 Conjugate priors . . . 117
2.4.3 Noninformative priors . . . 117
2.5 Nonparametric Methods . . . 120
2.5.1 Kernel density estimators . . . 122
2.5.2 Nearest-neighbour methods . . . 124
Exercises . . . 127

3 Linear Models for Regression **137**
3.1 Linear Basis Function Models . . . 138
3.1.1 Maximum likelihood and least squares . . . 140
3.1.2 Geometry of least squares . . . 143
3.1.3 Sequential learning . . . 143
3.1.4 Regularized least squares . . . 144
3.1.5 Multiple outputs . . . 146
3.2 The Bias-Variance Decomposition . . . 147
3.3 Bayesian Linear Regression . . . 152
3.3.1 Parameter distribution . . . 152
3.3.2 Predictive distribution . . . 156
3.3.3 Equivalent kernel . . . 159
3.4 Bayesian Model Comparison . . . 161
3.5 The Evidence Approximation . . . 165
3.5.1 Evaluation of the evidence function . . . 166
3.5.2 Maximizing the evidence function . . . 168
3.5.3 Effective number of parameters . . . 170
3.6 Limitations of Fixed Basis Functions . . . 172
Exercises . . . 173

4 Linear Models for Classification **179**
4.1 Discriminant Functions . . . 181
4.1.1 Two classes . . . 181
4.1.2 Multiple classes . . . 182
4.1.3 Least squares for classification . . . 184
4.1.4 Fisher's linear discriminant . . . 186
4.1.5 Relation to least squares . . . 189
4.1.6 Fisher's discriminant for multiple classes . . . 191
4.1.7 The perceptron algorithm . . . 192
4.2 Probabilistic Generative Models . . . 196
4.2.1 Continuous inputs . . . 198
4.2.2 Maximum likelihood solution . . . 200
4.2.3 Discrete features . . . 202
4.2.4 Exponential family . . . 202
4.3 Probabilistic Discriminative Models . . . 203
4.3.1 Fixed basis functions . . . 204
4.3.2 Logistic regression . . . 205
4.3.3 Iterative reweighted least squares . . . 207
4.3.4 Multiclass logistic regression . . . 209
4.3.5 Probit regression . . . 210
4.3.6 Canonical link functions . . . 212
4.4 The Laplace Approximation . . . 213
4.4.1 Model comparison and BIC . . . 216
4.5 Bayesian Logistic Regression . . . 217
4.5.1 Laplace approximation . . . 217
4.5.2 Predictive distribution . . . 218
Exercises . . . 220

5 Neural Networks **225**
5.1 Feed-forward Network Functions . . . 227
5.1.1 Weight-space symmetries . . . 231
5.2 Network Training . . . 232
5.2.1 Parameter optimization . . . 236
5.2.2 Local quadratic approximation . . . 237
5.2.3 Use of gradient information . . . 239
5.2.4 Gradient descent optimization . . . 240
5.3 Error Backpropagation . . . 241
5.3.1 Evaluation of error-function derivatives . . . 242
5.3.2 A simple example . . . 245
5.3.3 Efficiency of backpropagation . . . 246
5.3.4 The Jacobian matrix . . . 247
5.4 The Hessian Matrix . . . 249
5.4.1 Diagonal approximation . . . 250
5.4.2 Outer product approximation . . . 251
5.4.3 Inverse Hessian . . . 252

5.4.4 Finite differences . . . 252
5.4.5 Exact evaluation of the Hessian . . . 253
5.4.6 Fast multiplication by the Hessian . . . 254
5.5 Regularization in Neural Networks . . . 256
5.5.1 Consistent Gaussian priors . . . 257
5.5.2 Early stopping . . . 259
5.5.3 Invariances . . . 261
5.5.4 Tangent propagation . . . 263
5.5.5 Training with transformed data . . . 265
5.5.6 Convolutional networks . . . 267
5.5.7 Soft weight sharing . . . 269
5.6 Mixture Density Networks . . . 272
5.7 Bayesian Neural Networks . . . 277
5.7.1 Posterior parameter distribution . . . 278
5.7.2 Hyperparameter optimization . . . 280
5.7.3 Bayesian neural networks for classification . . . 281
Exercises . . . 284

6 Kernel Methods 291
6.1 Dual Representations . . . 293
6.2 Constructing Kernels . . . 294
6.3 Radial Basis Function Networks . . . 299
6.3.1 Nadaraya-Watson model . . . 301
6.4 Gaussian Processes . . . 303
6.4.1 Linear regression revisited . . . 304
6.4.2 Gaussian processes for regression . . . 306
6.4.3 Learning the hyperparameters . . . 311
6.4.4 Automatic relevance determination . . . 312
6.4.5 Gaussian processes for classification . . . 313
6.4.6 Laplace approximation . . . 315
6.4.7 Connection to neural networks . . . 319
Exercises . . . 320

7 Sparse Kernel Machines 325
7.1 Maximum Margin Classifiers . . . 326
7.1.1 Overlapping class distributions . . . 331
7.1.2 Relation to logistic regression . . . 336
7.1.3 Multiclass SVMs . . . 338
7.1.4 SVMs for regression . . . 339
7.1.5 Computational learning theory . . . 344
7.2 Relevance Vector Machines . . . 345
7.2.1 RVM for regression . . . 345
7.2.2 Analysis of sparsity . . . 349
7.2.3 RVM for classification . . . 353
Exercises . . . 357

8 Graphical Models **359**
8.1 Bayesian Networks . . . 360
8.1.1 Example: Polynomial regression . . . 362
8.1.2 Generative models . . . 365
8.1.3 Discrete variables . . . 366
8.1.4 Linear-Gaussian models . . . 370
8.2 Conditional Independence . . . 372
8.2.1 Three example graphs . . . 373
8.2.2 D-separation . . . 378
8.3 Markov Random Fields . . . 383
8.3.1 Conditional independence properties . . . 383
8.3.2 Factorization properties . . . 384
8.3.3 Illustration: Image de-noising . . . 387
8.3.4 Relation to directed graphs . . . 390
8.4 Inference in Graphical Models . . . 393
8.4.1 Inference on a chain . . . 394
8.4.2 Trees . . . 398
8.4.3 Factor graphs . . . 399
8.4.4 The sum-product algorithm . . . 402
8.4.5 The max-sum algorithm . . . 411
8.4.6 Exact inference in general graphs . . . 416
8.4.7 Loopy belief propagation . . . 417
8.4.8 Learning the graph structure . . . 418
Exercises . . . 418

9 Mixture Models and EM **423**
9.1 K-means Clustering . . . 424
9.1.1 Image segmentation and compression . . . 428
9.2 Mixtures of Gaussians . . . 430
9.2.1 Maximum likelihood . . . 432
9.2.2 EM for Gaussian mixtures . . . 435
9.3 An Alternative View of EM . . . 439
9.3.1 Gaussian mixtures revisited . . . 441
9.3.2 Relation to K-means . . . 443
9.3.3 Mixtures of Bernoulli distributions . . . 444
9.3.4 EM for Bayesian linear regression . . . 448
9.4 The EM Algorithm in General . . . 450
Exercises . . . 455

10 Approximate Inference **461**
10.1 Variational Inference . . . 462
10.1.1 Factorized distributions . . . 464
10.1.2 Properties of factorized approximations . . . 466
10.1.3 Example: The univariate Gaussian . . . 470
10.1.4 Model comparison . . . 473
10.2 Illustration: Variational Mixture of Gaussians . . . 474

10.2.1 Variational distribution . . . 475
10.2.2 Variational lower bound . . . 481
10.2.3 Predictive density . . . 482
10.2.4 Determining the number of components . . . 483
10.2.5 Induced factorizations . . . 485
10.3 Variational Linear Regression . . . 486
10.3.1 Variational distribution . . . 486
10.3.2 Predictive distribution . . . 488
10.3.3 Lower bound . . . 489
10.4 Exponential Family Distributions . . . 490
10.4.1 Variational message passing . . . 491
10.5 Local Variational Methods . . . 493
10.6 Variational Logistic Regression . . . 498
10.6.1 Variational posterior distribution . . . 498
10.6.2 Optimizing the variational parameters . . . 500
10.6.3 Inference of hyperparameters . . . 502
10.7 Expectation Propagation . . . 505
10.7.1 Example: The clutter problem . . . 511
10.7.2 Expectation propagation on graphs . . . 513
Exercises . . . 517

11 Sampling Methods **523**
11.1 Basic Sampling Algorithms . . . 526
11.1.1 Standard distributions . . . 526
11.1.2 Rejection sampling . . . 528
11.1.3 Adaptive rejection sampling . . . 530
11.1.4 Importance sampling . . . 532
11.1.5 Sampling-importance-resampling . . . 534
11.1.6 Sampling and the EM algorithm . . . 536
11.2 Markov Chain Monte Carlo . . . 537
11.2.1 Markov chains . . . 539
11.2.2 The Metropolis-Hastings algorithm . . . 541
11.3 Gibbs Sampling . . . 542
11.4 Slice Sampling . . . 546
11.5 The Hybrid Monte Carlo Algorithm . . . 548
11.5.1 Dynamical systems . . . 548
11.5.2 Hybrid Monte Carlo . . . 552
11.6 Estimating the Partition Function . . . 554
Exercises . . . 556

12 Continuous Latent Variables **559**
12.1 Principal Component Analysis . . . 561
12.1.1 Maximum variance formulation . . . 561
12.1.2 Minimum-error formulation . . . 563
12.1.3 Applications of PCA . . . 565
12.1.4 PCA for high-dimensional data . . . 569

12.2 Probabilistic PCA 570
12.2.1 Maximum likelihood PCA 574
12.2.2 EM algorithm for PCA 577
12.2.3 Bayesian PCA 580
12.2.4 Factor analysis 583
12.3 Kernel PCA 586
12.4 Nonlinear Latent Variable Models 591
12.4.1 Independent component analysis 591
12.4.2 Autoassociative neural networks 592
12.4.3 Modelling nonlinear manifolds 595
Exercises 599

13 Sequential Data **605**
13.1 Markov Models 607
13.2 Hidden Markov Models 610
13.2.1 Maximum likelihood for the HMM 615
13.2.2 The forward-backward algorithm 618
13.2.3 The sum-product algorithm for the HMM 625
13.2.4 Scaling factors 627
13.2.5 The Viterbi algorithm 629
13.2.6 Extensions of the hidden Markov model 631
13.3 Linear Dynamical Systems 635
13.3.1 Inference in LDS 638
13.3.2 Learning in LDS 642
13.3.3 Extensions of LDS 644
13.3.4 Particle filters 645
Exercises 646

14 Combining Models **653**
14.1 Bayesian Model Averaging 654
14.2 Committees 655
14.3 Boosting 657
14.3.1 Minimizing exponential error 659
14.3.2 Error functions for boosting 661
14.4 Tree-based Models 663
14.5 Conditional Mixture Models 666
14.5.1 Mixtures of linear regression models 667
14.5.2 Mixtures of logistic models 670
14.5.3 Mixtures of experts 672
Exercises 674

Appendix A Data Sets **677**

Appendix B Probability Distributions **685**

Appendix C Properties of Matrices **695**

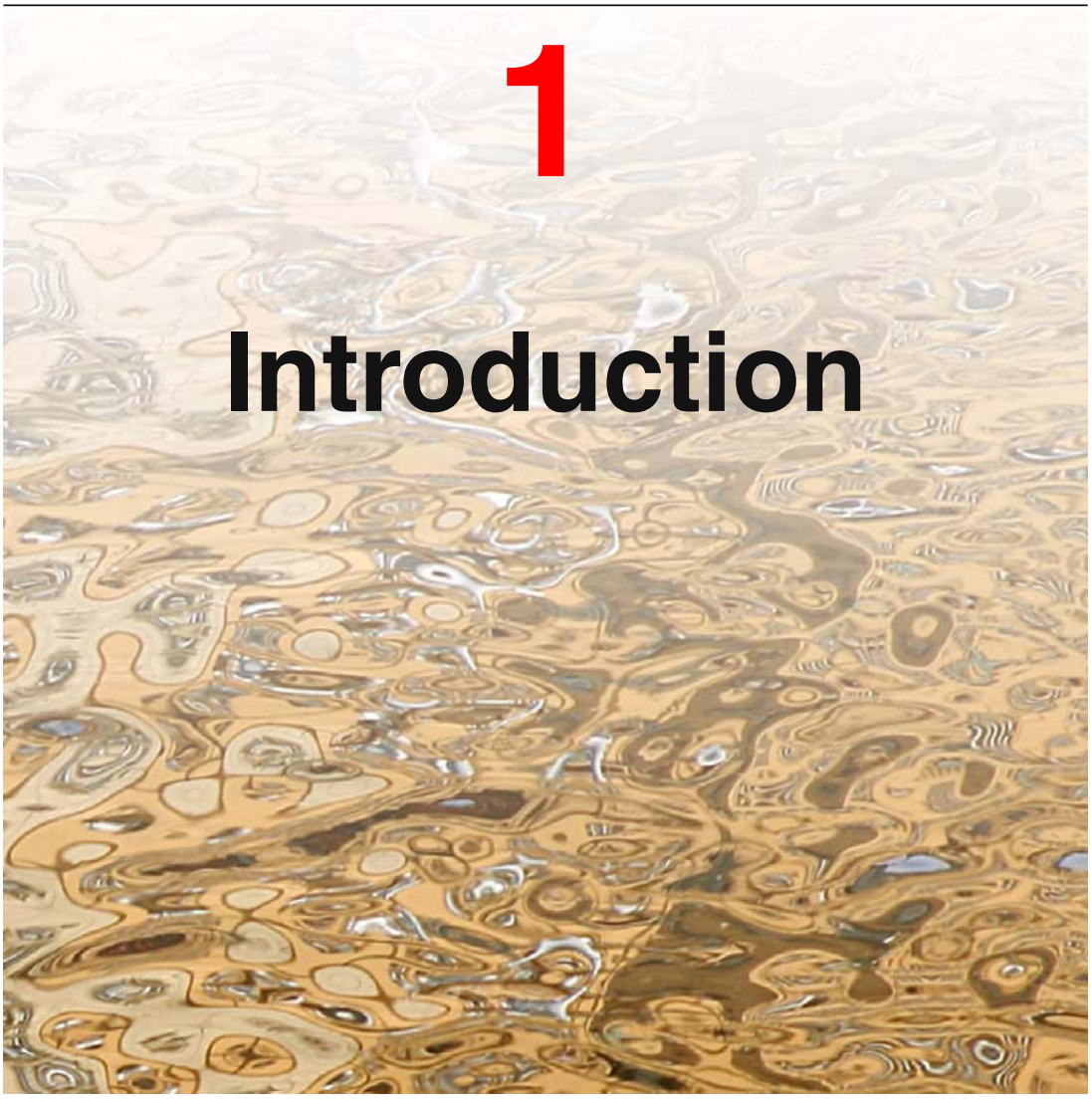
Appendix D Calculus of Variations **703**

Appendix E Lagrange Multipliers **707**

References **711**

Index **729**

1 Introduction



The problem of searching for patterns in data is a fundamental one and has a long and successful history. For instance, the extensive astronomical observations of Tycho Brahe in the 16^{th} century allowed Johannes Kepler to discover the empirical laws of planetary motion, which in turn provided a springboard for the development of classical mechanics. Similarly, the discovery of regularities in atomic spectra played a key role in the development and verification of quantum physics in the early twentieth century. The field of pattern recognition is concerned with the automatic discovery of regularities in data through the use of computer algorithms and with the use of these regularities to take actions such as classifying the data into different categories.

Consider the example of recognizing handwritten digits, illustrated in Figure 1.1. Each digit corresponds to a 28×28 pixel image and so can be represented by a vector $\mathbf{x}$ comprising 784 real numbers. The goal is to build a machine that will take such a vector $\mathbf{x}$ as input and that will produce the identity of the digit $0, \ldots, 9$ as the output. This is a nontrivial problem due to the wide variability of handwriting. It could be

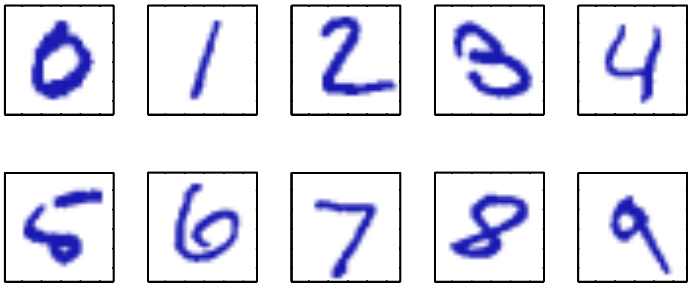


Figure 1.1 Examples of hand-written digits taken from US zip codes.

tackled using handcrafted rules or heuristics for distinguishing the digits based on the shapes of the strokes, but in practice such an approach leads to a proliferation of rules and of exceptions to the rules and so on, and invariably gives poor results.

Far better results can be obtained by adopting a machine learning approach in which a large set of N digits $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ called a *training set* is used to tune the parameters of an adaptive model. The categories of the digits in the training set are known in advance, typically by inspecting them individually and hand-labelling them. We can express the category of a digit using *target vector* $\mathbf{t}$, which represents the identity of the corresponding digit. Suitable techniques for representing categories in terms of vectors will be discussed later. Note that there is one such target vector $\mathbf{t}$ for each digit image $\mathbf{x}$.

The result of running the machine learning algorithm can be expressed as a function $\mathbf{y}(\mathbf{x})$ which takes a new digit image $\mathbf{x}$ as input and that generates an output vector $\mathbf{y}$, encoded in the same way as the target vectors. The precise form of the function $\mathbf{y}(\mathbf{x})$ is determined during the *training* phase, also known as the *learning* phase, on the basis of the training data. Once the model is trained it can then determine the identity of new digit images, which are said to comprise a *test set*. The ability to categorize correctly new examples that differ from those used for training is known as *generalization*. In practical applications, the variability of the input vectors will be such that the training data can comprise only a tiny fraction of all possible input vectors, and so generalization is a central goal in pattern recognition.

For most practical applications, the original input variables are typically *preprocessed* to transform them into some new space of variables where, it is hoped, the pattern recognition problem will be easier to solve. For instance, in the digit recognition problem, the images of the digits are typically translated and scaled so that each digit is contained within a box of a fixed size. This greatly reduces the variability within each digit class, because the location and scale of all the digits are now the same, which makes it much easier for a subsequent pattern recognition algorithm to distinguish between the different classes. This pre-processing stage is sometimes also called *feature extraction*. Note that new test data must be pre-processed using the same steps as the training data.

Pre-processing might also be performed in order to speed up computation. For example, if the goal is real-time face detection in a high-resolution video stream, the computer must handle huge numbers of pixels per second, and presenting these directly to a complex pattern recognition algorithm may be computationally infeasible. Instead, the aim is to find useful features that are fast to compute, and yet that

also preserve useful discriminatory information enabling faces to be distinguished from non-faces. These features are then used as the inputs to the pattern recognition algorithm. For instance, the average value of the image intensity over a rectangular subregion can be evaluated extremely efficiently (Viola and Jones, 2004), and a set of such features can prove very effective in fast face detection. Because the number of such features is smaller than the number of pixels, this kind of pre-processing represents a form of dimensionality reduction. Care must be taken during pre-processing because often information is discarded, and if this information is important to the solution of the problem then the overall accuracy of the system can suffer.

Applications in which the training data comprises examples of the input vectors along with their corresponding target vectors are known as *supervised learning* problems. Cases such as the digit recognition example, in which the aim is to assign each input vector to one of a finite number of discrete categories, are called *classification* problems. If the desired output consists of one or more continuous variables, then the task is called *regression*. An example of a regression problem would be the prediction of the yield in a chemical manufacturing process in which the inputs consist of the concentrations of reactants, the temperature, and the pressure.

In other pattern recognition problems, the training data consists of a set of input vectors $\mathbf{x}$ without any corresponding target values. The goal in such *unsupervised learning* problems may be to discover groups of similar examples within the data, where it is called *clustering*, or to determine the distribution of data within the input space, known as *density estimation*, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of *visualization*.

Finally, the technique of *reinforcement learning* (Sutton and Barto, 1998) is concerned with the problem of finding suitable actions to take in a given situation in order to maximize a reward. Here the learning algorithm is not given examples of optimal outputs, in contrast to supervised learning, but must instead discover them by a process of trial and error. Typically there is a sequence of states and actions in which the learning algorithm is interacting with its environment. In many cases, the current action not only affects the immediate reward but also has an impact on the reward at all subsequent time steps. For example, by using appropriate reinforcement learning techniques a neural network can learn to play the game of backgammon to a high standard (Tesauro, 1994). Here the network must learn to take a board position as input, along with the result of a dice throw, and produce a strong move as the output. This is done by having the network play against a copy of itself for perhaps a million games. A major challenge is that a game of backgammon can involve dozens of moves, and yet it is only at the end of the game that the reward, in the form of victory, is achieved. The reward must then be attributed appropriately to all of the moves that led to it, even though some moves will have been good ones and others less so. This is an example of a *credit assignment* problem. A general feature of reinforcement learning is the trade-off between *exploration*, in which the system tries out new kinds of actions to see how effective they are, and *exploitation*, in which the system makes use of actions that are known to yield a high reward. Too strong a focus on either exploration or exploitation will yield poor results. Reinforcement learning continues to be an active area of machine learning research. However, a

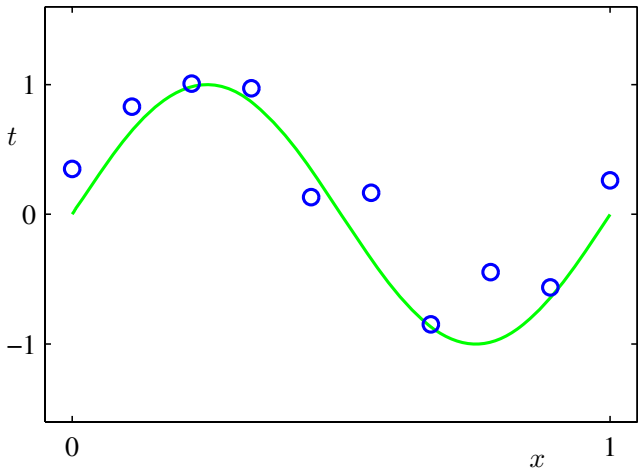


Figure 1.2 Plot of a training data set of $N = 10$ points, shown as blue circles, each comprising an observation of the input variable x along with the corresponding target variable t. The green curve shows the function $\sin(2\pi x)$ used to generate the data. Our goal is to predict the value of t for some new value of x, without knowledge of the green curve.

detailed treatment lies beyond the scope of this book.

Although each of these tasks needs its own tools and techniques, many of the key ideas that underpin them are common to all such problems. One of the main goals of this chapter is to introduce, in a relatively informal way, several of the most important of these concepts and to illustrate them using simple examples. Later in the book we shall see these same ideas re-emerge in the context of more sophisticated models that are applicable to real-world pattern recognition applications. This chapter also provides a self-contained introduction to three important tools that will be used throughout the book, namely probability theory, decision theory, and information theory. Although these might sound like daunting topics, they are in fact straightforward, and a clear understanding of them is essential if machine learning techniques are to be used to best effect in practical applications.

1.1. Example: Polynomial Curve Fitting

We begin by introducing a simple regression problem, which we shall use as a running example throughout this chapter to motivate a number of key concepts. Suppose we observe a real-valued input variable x and we wish to use this observation to predict the value of a real-valued target variable t. For the present purposes, it is instructive to consider an artificial example using synthetically generated data because we then know the precise process that generated the data for comparison against any learned model. The data for this example is generated from the function $\sin(2\pi x)$ with random noise included in the target values, as described in detail in Appendix A.

Now suppose that we are given a training set comprising N observations of x, written $\mathbf{x} \equiv (x_1, \ldots, x_N)^{\mathrm{T}}$, together with corresponding observations of the values of t, denoted $\mathbf{t} \equiv (t_1, \ldots, t_N)^{\mathrm{T}}$. Figure 1.2 shows a plot of a training set comprising $N = 10$ data points. The input data set $\mathbf{x}$ in Figure 1.2 was generated by choosing values of x_n, for $n = 1, \ldots, N$, spaced uniformly in range $[0, 1]$, and the target data set $\mathbf{t}$ was obtained by first computing the corresponding values of the function

$\sin(2\pi x)$ and then adding a small level of random noise having a Gaussian distribution (the Gaussian distribution is discussed in Section 1.2.4) to each such point in order to obtain the corresponding value t_n. By generating data in this way, we are capturing a property of many real data sets, namely that they possess an underlying regularity, which we wish to learn, but that individual observations are corrupted by random noise. This noise might arise from intrinsically stochastic (i.e. random) processes such as radioactive decay but more typically is due to there being sources of variability that are themselves unobserved.

Our goal is to exploit this training set in order to make predictions of the value $\widehat{t}$ of the target variable for some new value $\widehat{x}$ of the input variable. As we shall see later, this involves implicitly trying to discover the underlying function $\sin(2\pi x)$. This is intrinsically a difficult problem as we have to generalize from a finite data set. Furthermore the observed data are corrupted with noise, and so for a given $\widehat{x}$ there is uncertainty as to the appropriate value for $\widehat{t}$. Probability theory, discussed in Section 1.2, provides a framework for expressing such uncertainty in a precise and quantitative manner, and decision theory, discussed in Section 1.5, allows us to exploit this probabilistic representation in order to make predictions that are optimal according to appropriate criteria.

For the moment, however, we shall proceed rather informally and consider a simple approach based on curve fitting. In particular, we shall fit the data using a polynomial function of the form

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j \tag{1.1}$$

where M is the *order* of the polynomial, and x^j denotes x raised to the power of j. The polynomial coefficients $w_0, \ldots, w_M$ are collectively denoted by the vector $\mathbf{w}$. Note that, although the polynomial function $y(x, \mathbf{w})$ is a nonlinear function of x, it is a linear function of the coefficients $\mathbf{w}$. Functions, such as the polynomial, which are linear in the unknown parameters have important properties and are called *linear models* and will be discussed extensively in Chapters 3 and 4.

The values of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimizing an *error function* that measures the misfit between the function $y(x, \mathbf{w})$, for any given value of $\mathbf{w}$, and the training set data points. One simple choice of error function, which is widely used, is given by the sum of the squares of the errors between the predictions $y(x_n, \mathbf{w})$ for each data point x_n and the corresponding target values t_n, so that we minimize

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2 \tag{1.2}$$

where the factor of $1/2$ is included for later convenience. We shall discuss the motivation for this choice of error function later in this chapter. For the moment we simply note that it is a nonnegative quantity that would be zero if, and only if, the

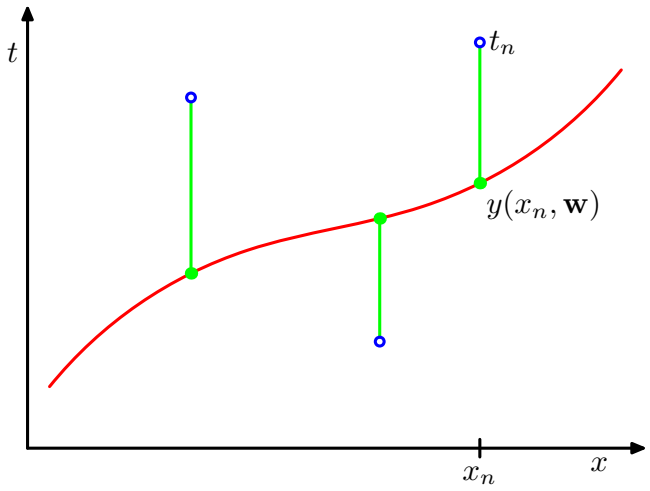


Figure 1.3 The error function (1.2) corresponds to (one half of) the sum of the squares of the displacements (shown by the vertical green bars) of each data point from the function $y(x, \mathbf{w})$.

function $y(x, \mathbf{w})$ were to pass exactly through each training data point. The geometrical interpretation of the sum-of-squares error function is illustrated in Figure 1.3.

We can solve the curve fitting problem by choosing the value of $\mathbf{w}$ for which $E(\mathbf{w})$ is as small as possible. Because the error function is a quadratic function of the coefficients $\mathbf{w}$, its derivatives with respect to the coefficients will be linear in the elements of $\mathbf{w}$, and so the minimization of the error function has a unique solution, denoted by $\mathbf{w}^\star$, which can be found in closed form. The resulting polynomial is given by the function $y(x, \mathbf{w}^\star)$.

Exercise 1.1

There remains the problem of choosing the order M of the polynomial, and as we shall see this will turn out to be an example of an important concept called *model comparison* or *model selection*. In Figure 1.4, we show four examples of the results of fitting polynomials having orders $M = 0, 1, 3$, and 9 to the data set shown in Figure 1.2.

We notice that the constant ($M = 0$) and first order ($M = 1$) polynomials give rather poor fits to the data and consequently rather poor representations of the function $\sin(2\pi x)$. The third order ($M = 3$) polynomial seems to give the best fit to the function $\sin(2\pi x)$ of the examples shown in Figure 1.4. When we go to a much higher order polynomial ($M = 9$), we obtain an excellent fit to the training data. In fact, the polynomial passes exactly through each data point and $E(\mathbf{w}^\star) = 0$. However, the fitted curve oscillates wildly and gives a very poor representation of the function $\sin(2\pi x)$. This latter behaviour is known as *over-fitting*.

As we have noted earlier, the goal is to achieve good generalization by making accurate predictions for new data. We can obtain some quantitative insight into the dependence of the generalization performance on M by considering a separate test set comprising 100 data points generated using exactly the same procedure used to generate the training set points but with new choices for the random noise values included in the target values. For each choice of M, we can then evaluate the residual value of $E(\mathbf{w}^\star)$ given by (1.2) for the training data, and we can also evaluate $E(\mathbf{w}^\star)$ for the test data set. It is sometimes more convenient to use the root-mean-square

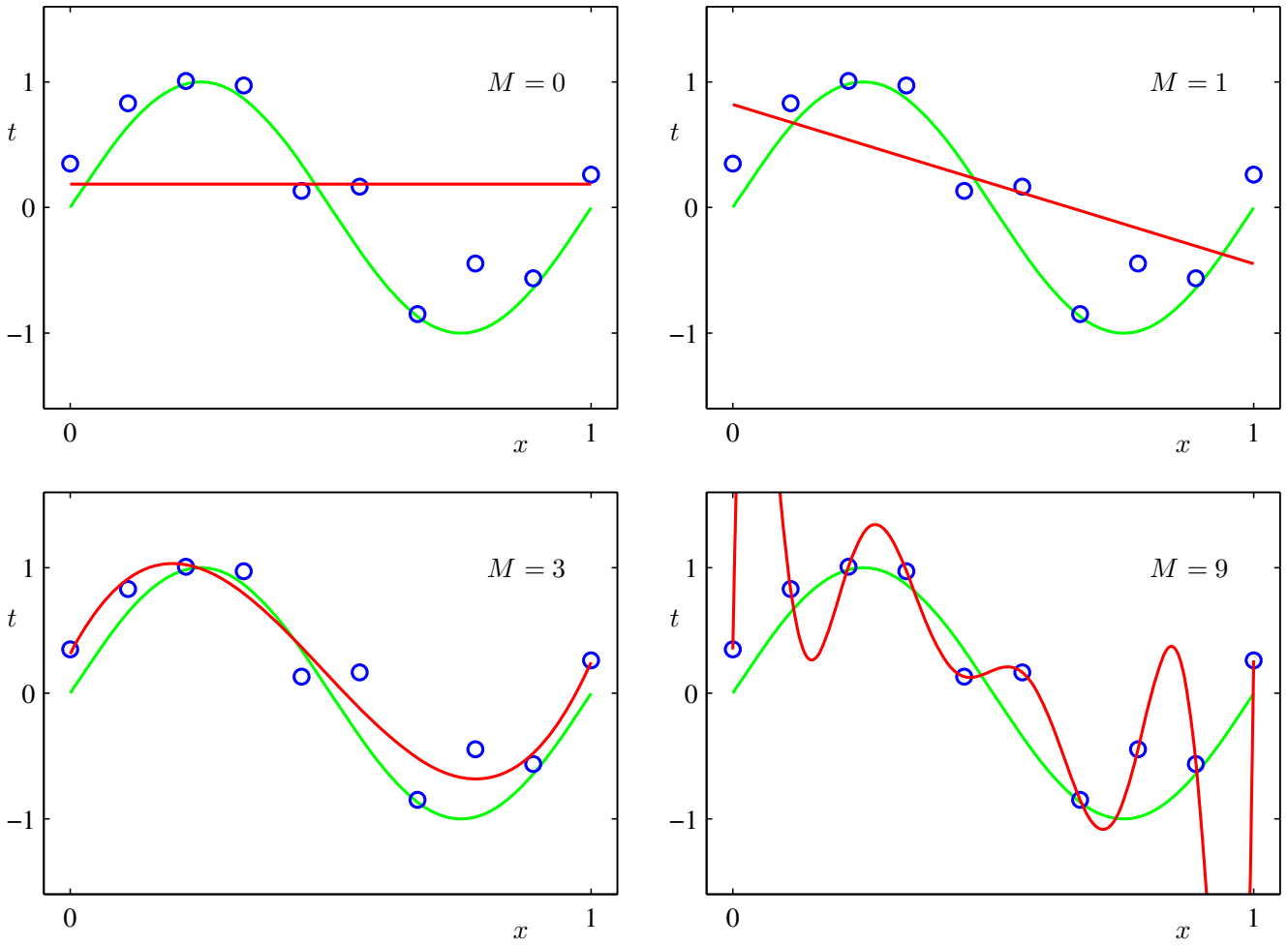


Figure 1.4 Plots of polynomials having various orders M, shown as red curves, fitted to the data set shown in Figure 1.2.

(RMS) error defined by

$$E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N} \tag{1.3}$$

in which the division by N allows us to compare different sizes of data sets on an equal footing, and the square root ensures that E_{RMS} is measured on the same scale (and in the same units) as the target variable t. Graphs of the training and test set RMS errors are shown, for various values of M, in Figure 1.5. The test set error is a measure of how well we are doing in predicting the values of t for new data observations of x. We note from Figure 1.5 that small values of M give relatively large values of the test set error, and this can be attributed to the fact that the corresponding polynomials are rather inflexible and are incapable of capturing the oscillations in the function $\sin(2\pi x)$. Values of M in the range $3 \leqslant M \leqslant 8$ give small values for the test set error, and these also give reasonable representations of the generating function $\sin(2\pi x)$, as can be seen, for the case of $M = 3$, from Figure 1.4.

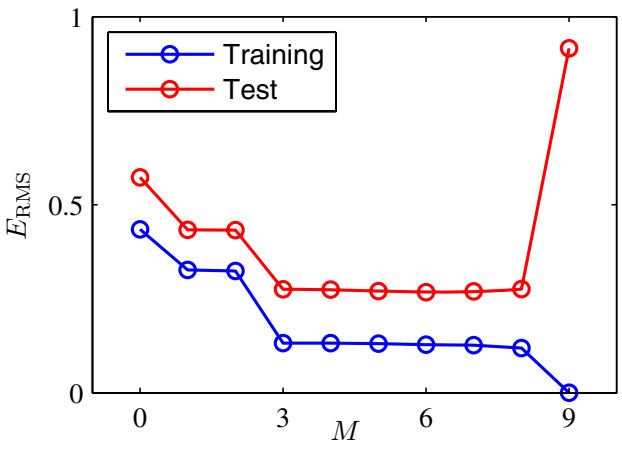


Figure 1.5 Graphs of the root-mean-square error, defined by (1.3), evaluated on the training set and on an independent test set for various values of M.

For $M = 9$, the training set error goes to zero, as we might expect because this polynomial contains 10 degrees of freedom corresponding to the 10 coefficients $w_0, \ldots, w_9$, and so can be tuned exactly to the 10 data points in the training set. However, the test set error has become very large and, as we saw in Figure 1.4, the corresponding function $y(x, \mathbf{w}^\star)$ exhibits wild oscillations.

This may seem paradoxical because a polynomial of given order contains all lower order polynomials as special cases. The $M = 9$ polynomial is therefore capable of generating results at least as good as the $M = 3$ polynomial. Furthermore, we might suppose that the best predictor of new data would be the function $\sin(2\pi x)$ from which the data was generated (and we shall see later that this is indeed the case). We know that a power series expansion of the function $\sin(2\pi x)$ contains terms of all orders, so we might expect that results should improve monotonically as we increase M.

We can gain some insight into the problem by examining the values of the coefficients $\mathbf{w}^\star$ obtained from polynomials of various order, as shown in Table 1.1. We see that, as M increases, the magnitude of the coefficients typically gets larger. In particular for the $M = 9$ polynomial, the coefficients have become finely tuned to the data by developing large positive and negative values so that the correspond-

Table 1.1 Table of the coefficients $\mathbf{w}^\star$ for polynomials of various order. Observe how the typical magnitude of the coefficients increases dramatically as the order of the polynomial increases.

	$M = 0$	$M = 1$	$M = 6$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

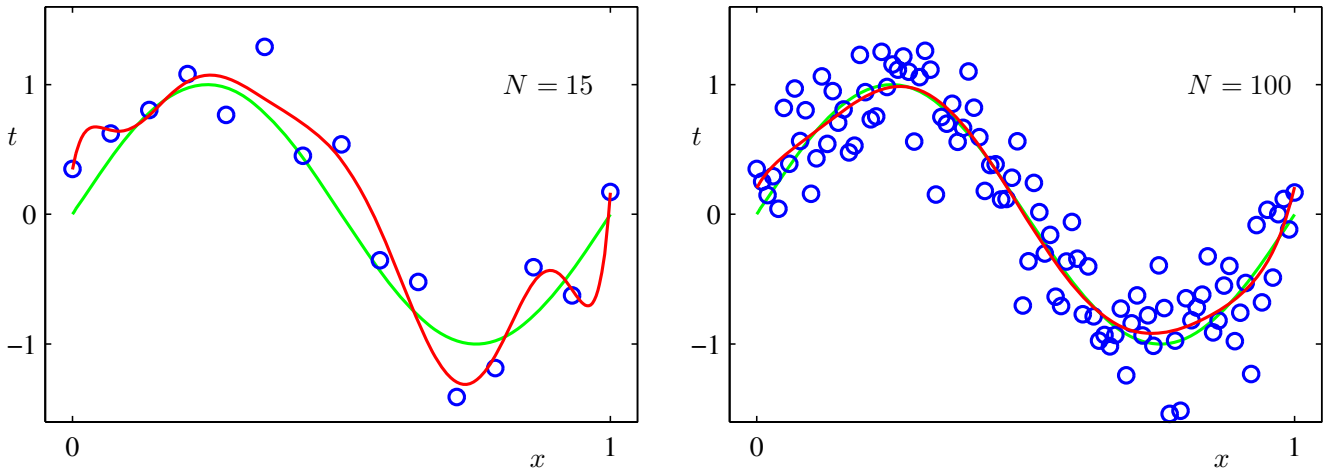


Figure 1.6 Plots of the solutions obtained by minimizing the sum-of-squares error function using the $M = 9$ polynomial for $N = 15$ data points (left plot) and $N = 100$ data points (right plot). We see that increasing the size of the data set reduces the over-fitting problem.

ing polynomial function matches each of the data points exactly, but between data points (particularly near the ends of the range) the function exhibits the large oscillations observed in Figure 1.4. Intuitively, what is happening is that the more flexible polynomials with larger values of M are becoming increasingly tuned to the random noise on the target values.

It is also interesting to examine the behaviour of a given model as the size of the data set is varied, as shown in Figure 1.6. We see that, for a given model complexity, the over-fitting problem become less severe as the size of the data set increases. Another way to say this is that the larger the data set, the more complex (in other words more flexible) the model that we can afford to fit to the data. One rough heuristic that is sometimes advocated is that the number of data points should be no less than some multiple (say 5 or 10) of the number of adaptive parameters in the model. However, as we shall see in Chapter 3, the number of parameters is not necessarily the most appropriate measure of model complexity.

Also, there is something rather unsatisfying about having to limit the number of parameters in a model according to the size of the available training set. It would seem more reasonable to choose the complexity of the model according to the complexity of the problem being solved. We shall see that the least squares approach to finding the model parameters represents a specific case of *maximum likelihood* (discussed in Section 1.2.5), and that the over-fitting problem can be understood as a general property of maximum likelihood. By adopting a *Bayesian* approach, the over-fitting problem can be avoided. We shall see that there is no difficulty from a Bayesian perspective in employing models for which the number of parameters greatly exceeds the number of data points. Indeed, in a Bayesian model the *effective* number of parameters adapts automatically to the size of the data set.

Section 3.4

For the moment, however, it is instructive to continue with the current approach and to consider how in practice we can apply it to data sets of limited size where we

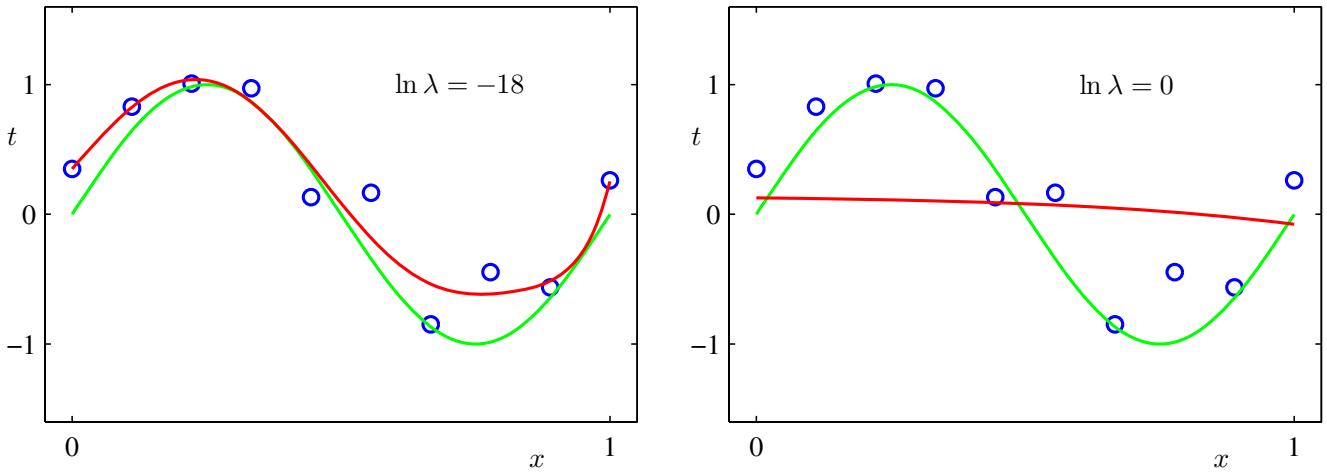


Figure 1.7 Plots of $M = 9$ polynomials fitted to the data set shown in Figure 1.2 using the regularized error function (1.4) for two values of the regularization parameter λ corresponding to $\ln \lambda = -18$ and $\ln \lambda = 0$. The case of no regularizer, i.e., $\lambda = 0$, corresponding to $\ln \lambda = -\infty$, is shown at the bottom right of Figure 1.4.

may wish to use relatively complex and flexible models. One technique that is often used to control the over-fitting phenomenon in such cases is that of *regularization*, which involves adding a penalty term to the error function (1.2) in order to discourage the coefficients from reaching large values. The simplest such penalty term takes the form of a sum of squares of all of the coefficients, leading to a modified error function of the form

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2 \tag{1.4}$$

where $\|\mathbf{w}\|^2 \equiv \mathbf{w}^{\mathrm{T}}\mathbf{w} = w_0^2 + w_1^2 + \ldots + w_M^2$, and the coefficient λ governs the relative importance of the regularization term compared with the sum-of-squares error term. Note that often the coefficient w_0 is omitted from the regularizer because its inclusion causes the results to depend on the choice of origin for the target variable (Hastie *et al.*, 2001), or it may be included but with its own regularization coefficient (we shall discuss this topic in more detail in Section 5.5.1). Again, the error function in (1.4) can be minimized exactly in closed form. Techniques such as this are known in the statistics literature as *shrinkage* methods because they reduce the value of the coefficients. The particular case of a quadratic regularizer is called *ridge regression* (Hoerl and Kennard, 1970). In the context of neural networks, this approach is known as *weight decay*.

Exercise 1.2

Figure 1.7 shows the results of fitting the polynomial of order $M = 9$ to the same data set as before but now using the regularized error function given by (1.4). We see that, for a value of $\ln \lambda = -18$, the over-fitting has been suppressed and we now obtain a much closer representation of the underlying function $\sin(2\pi x)$. If, however, we use too large a value for λ then we again obtain a poor fit, as shown in Figure 1.7 for $\ln \lambda = 0$. The corresponding coefficients from the fitted polynomials are given in Table 1.2, showing that regularization has the desired effect of reducing

Table 1.2 Table of the coefficients $\mathbf{w}^\star$ for $M = 9$ polynomials with various values for the regularization parameter λ. Note that $\ln \lambda = -\infty$ corresponds to a model with no regularization, i.e., to the graph at the bottom right in Figure 1.4. We see that, as the value of λ increases, the typical magnitude of the coefficients gets smaller.

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

the magnitude of the coefficients.

The impact of the regularization term on the generalization error can be seen by plotting the value of the RMS error (1.3) for both training and test sets against $\ln \lambda$, as shown in Figure 1.8. We see that in effect λ now controls the effective complexity of the model and hence determines the degree of over-fitting.

The issue of model complexity is an important one and will be discussed at length in Section 1.3. Here we simply note that, if we were trying to solve a practical application using this approach of minimizing an error function, we would have to find a way to determine a suitable value for the model complexity. The results above suggest a simple way of achieving this, namely by taking the available data and partitioning it into a training set, used to determine the coefficients $\mathbf{w}$, and a separate *validation* set, also called a *hold-out* set, used to optimize the model complexity (either M or λ). In many cases, however, this will prove to be too wasteful of valuable training data, and we have to seek more sophisticated approaches.

Section 1.3

So far our discussion of polynomial curve fitting has appealed largely to intuition. We now seek a more principled approach to solving problems in pattern recognition by turning to a discussion of probability theory. As well as providing the foundation for nearly all of the subsequent developments in this book, it will also

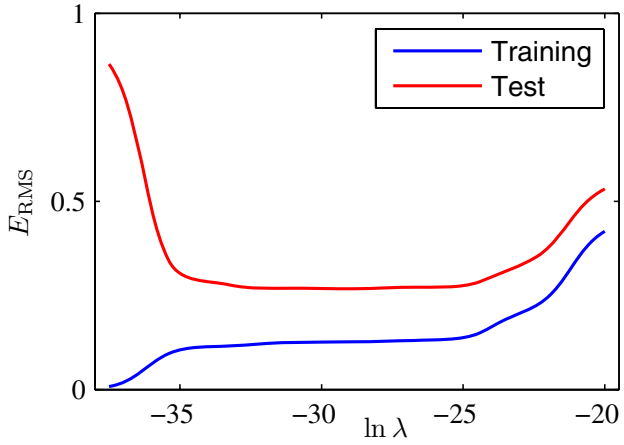


Figure 1.8 Graph of the root-mean-square error (1.3) versus $\ln \lambda$ for the $M = 9$ polynomial.

give us some important insights into the concepts we have introduced in the context of polynomial curve fitting and will allow us to extend these to more complex situations.

1.2. Probability Theory

A key concept in the field of pattern recognition is that of uncertainty. It arises both through noise on measurements, as well as through the finite size of data sets. Probability theory provides a consistent framework for the quantification and manipulation of uncertainty and forms one of the central foundations for pattern recognition. When combined with decision theory, discussed in Section 1.5, it allows us to make optimal predictions given all the information available to us, even though that information may be incomplete or ambiguous.

We will introduce the basic concepts of probability theory by considering a simple example. Imagine we have two boxes, one red and one blue, and in the red box we have 2 apples and 6 oranges, and in the blue box we have 3 apples and 1 orange. This is illustrated in Figure 1.9. Now suppose we randomly pick one of the boxes and from that box we randomly select an item of fruit, and having observed which sort of fruit it is we replace it in the box from which it came. We could imagine repeating this process many times. Let us suppose that in so doing we pick the red box 40% of the time and we pick the blue box 60% of the time, and that when we remove an item of fruit from a box we are equally likely to select any of the pieces of fruit in the box.

In this example, the identity of the box that will be chosen is a random variable, which we shall denote by B. This random variable can take one of two possible values, namely r (corresponding to the red box) or b (corresponding to the blue box). Similarly, the identity of the fruit is also a random variable and will be denoted by F. It can take either of the values a (for apple) or o (for orange).

To begin with, we shall define the probability of an event to be the fraction of times that event occurs out of the total number of trials, in the limit that the total number of trials goes to infinity. Thus the probability of selecting the red box is $4/10$

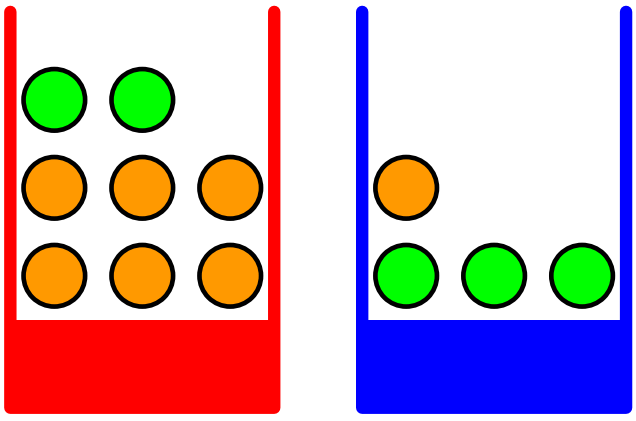


Figure 1.9 We use a simple example of two coloured boxes each containing fruit (apples shown in green and oranges shown in orange) to introduce the basic ideas of probability.