

Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series

MACHINE LEARNING

An Algorithmic Perspective

SECOND EDITION

STEPHEN MARSLAND



CRC Press
Taylor & Francis Group

A CHAPMAN & HALL BOOK

WITH VITALSOURCE®
EBOOK



MACHINE LEARNING

An Algorithmic Perspective

SECOND EDITION

Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series

SERIES EDITORS

Ralf Herbrich
Amazon Development Center
Berlin, Germany

Thore Graepel
Microsoft Research Ltd.
Cambridge, UK

AIMS AND SCOPE

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

PUBLISHED TITLES

BAYESIAN PROGRAMMING

Pierre Bessière, Emmanuel Mazer, Juan-Manuel Ahuactzin, and Kamel Mekhnacha

UTILITY-BASED LEARNING FROM DATA

Craig Friedman and Sven Sandow

HANDBOOK OF NATURAL LANGUAGE PROCESSING, SECOND EDITION

Nitin Indurkha and Fred J. Damerau

COST-SENSITIVE MACHINE LEARNING

Balaji Krishnapuram, Shipeng Yu, and Bharat Rao

COMPUTATIONAL TRUST MODELS AND MACHINE LEARNING

Xin Liu, Anwitaman Datta, and Ee-Peng Lim

**MULTILINEAR SUBSPACE LEARNING: DIMENSIONALITY REDUCTION OF
MULTIDIMENSIONAL DATA**

Haiping Lu, Konstantinos N. Plataniotis, and Anastasios N. Venetsanopoulos

MACHINE LEARNING: An Algorithmic Perspective, Second Edition

Stephen Marsland

A FIRST COURSE IN MACHINE LEARNING

Simon Rogers and Mark Girolami

MULTI-LABEL DIMENSIONALITY REDUCTION

Liang Sun, Shuiwang Ji, and Jieping Ye

ENSEMBLE METHODS: FOUNDATIONS AND ALGORITHMS

Zhi-Hua Zhou

Chapman & Hall/CRC
Machine Learning & Pattern Recognition Series

MACHINE LEARNING

An Algorithmic Perspective

SECOND EDITION

STEPHEN MARSLAND



CRC Press

Taylor & Francis Group

Boca Raton London New York

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

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

© 2015 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works
Version Date: 20140826

International Standard Book Number-13: 978-1-4665-8333-7 (eBook - PDF)

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

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

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

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

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

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

Again, for Monika

Contents

Prologue to 2nd Edition	xvii
Prologue to 1st Edition	xix
CHAPTER 1 ■ Introduction	1
1.1 IF DATA HAD MASS, THE EARTH WOULD BE A BLACK HOLE	1
1.2 LEARNING	4
1.2.1 Machine Learning	4
1.3 TYPES OF MACHINE LEARNING	5
1.4 SUPERVISED LEARNING	6
1.4.1 Regression	6
1.4.2 Classification	8
1.5 THE MACHINE LEARNING PROCESS	10
1.6 A NOTE ON PROGRAMMING	11
1.7 A ROADMAP TO THE BOOK	12
FURTHER READING	13
CHAPTER 2 ■ Preliminaries	15
2.1 SOME TERMINOLOGY	15
2.1.1 Weight Space	16
2.1.2 The Curse of Dimensionality	17
2.2 KNOWING WHAT YOU KNOW: TESTING MACHINE LEARNING ALGORITHMS	19
2.2.1 Overfitting	19
2.2.2 Training, Testing, and Validation Sets	20
2.2.3 The Confusion Matrix	21
2.2.4 Accuracy Metrics	22
2.2.5 The Receiver Operator Characteristic (ROC) Curve	24
2.2.6 Unbalanced Datasets	25
2.2.7 Measurement Precision	25
2.3 TURNING DATA INTO PROBABILITIES	27
2.3.1 Minimising Risk	30

2.3.2	The Naïve Bayes' Classifier	30
2.4	SOME BASIC STATISTICS	32
2.4.1	Averages	32
2.4.2	Variance and Covariance	32
2.4.3	The Gaussian	34
2.5	THE BIAS-VARIANCE TRADEOFF	35
	FURTHER READING	36
	PRACTICE QUESTIONS	37
CHAPTER 3 ■ Neurons, Neural Networks, and Linear Discriminants		39
<hr/>		
3.1	THE BRAIN AND THE NEURON	39
3.1.1	Hebb's Rule	40
3.1.2	McCulloch and Pitts Neurons	40
3.1.3	Limitations of the McCulloch and Pitts Neuronal Model	42
3.2	NEURAL NETWORKS	43
3.3	THE PERCEPTRON	43
3.3.1	The Learning Rate η	46
3.3.2	The Bias Input	46
3.3.3	The Perceptron Learning Algorithm	47
3.3.4	An Example of Perceptron Learning: Logic Functions	48
3.3.5	Implementation	49
3.4	LINEAR SEPARABILITY	55
3.4.1	The Perceptron Convergence Theorem	57
3.4.2	The Exclusive Or (XOR) Function	58
3.4.3	A Useful Insight	59
3.4.4	Another Example: The Pima Indian Dataset	61
3.4.5	Preprocessing: Data Preparation	63
3.5	LINEAR REGRESSION	64
3.5.1	Linear Regression Examples	66
	FURTHER READING	67
	PRACTICE QUESTIONS	68
CHAPTER 4 ■ The Multi-layer Perceptron		71
<hr/>		
4.1	GOING FORWARDS	73
4.1.1	Biases	73
4.2	GOING BACKWARDS: BACK-PROPAGATION OF ERROR	74
4.2.1	The Multi-layer Perceptron Algorithm	77
4.2.2	Initialising the Weights	80
4.2.3	Different Output Activation Functions	81

4.2.4	Sequential and Batch Training	82
4.2.5	Local Minima	82
4.2.6	Picking Up Momentum	84
4.2.7	Minibatches and Stochastic Gradient Descent	85
4.2.8	Other Improvements	85
4.3	THE MULTI-LAYER PERCEPTRON IN PRACTICE	85
4.3.1	Amount of Training Data	86
4.3.2	Number of Hidden Layers	86
4.3.3	When to Stop Learning	88
4.4	EXAMPLES OF USING THE MLP	89
4.4.1	A Regression Problem	89
4.4.2	Classification with the MLP	92
4.4.3	A Classification Example: The Iris Dataset	93
4.4.4	Time-Series Prediction	95
4.4.5	Data Compression: The Auto-Associative Network	97
4.5	A RECIPE FOR USING THE MLP	100
4.6	DERIVING BACK-PROPAGATION	101
4.6.1	The Network Output and the Error	101
4.6.2	The Error of the Network	102
4.6.3	Requirements of an Activation Function	103
4.6.4	Back-Propagation of Error	104
4.6.5	The Output Activation Functions	107
4.6.6	An Alternative Error Function	108
	FURTHER READING	108
	PRACTICE QUESTIONS	109
CHAPTER 5 ■ Radial Basis Functions and Splines		111
<hr/>		
5.1	RECEPTIVE FIELDS	111
5.2	THE RADIAL BASIS FUNCTION (RBF) NETWORK	114
5.2.1	Training the RBF Network	117
5.3	INTERPOLATION AND BASIS FUNCTIONS	119
5.3.1	Bases and Basis Expansion	122
5.3.2	The Cubic Spline	123
5.3.3	Fitting the Spline to the Data	123
5.3.4	Smoothing Splines	124
5.3.5	Higher Dimensions	125
5.3.6	Beyond the Bounds	127
	FURTHER READING	127
	PRACTICE QUESTIONS	128

CHAPTER 6 ■ Dimensionality Reduction	129
6.1 LINEAR DISCRIMINANT ANALYSIS (LDA)	130
6.2 PRINCIPAL COMPONENTS ANALYSIS (PCA)	133
6.2.1 Relation with the Multi-layer Perceptron	137
6.2.2 Kernel PCA	138
6.3 FACTOR ANALYSIS	141
6.4 INDEPENDENT COMPONENTS ANALYSIS (ICA)	142
6.5 LOCALLY LINEAR EMBEDDING	144
6.6 ISOMAP	147
6.6.1 Multi-Dimensional Scaling (MDS)	147
FURTHER READING	150
PRACTICE QUESTIONS	151
CHAPTER 7 ■ Probabilistic Learning	153
7.1 GAUSSIAN MIXTURE MODELS	153
7.1.1 The Expectation-Maximisation (EM) Algorithm	154
7.1.2 Information Criteria	158
7.2 NEAREST NEIGHBOUR METHODS	158
7.2.1 Nearest Neighbour Smoothing	160
7.2.2 Efficient Distance Computations: the KD-Tree	160
7.2.3 Distance Measures	165
FURTHER READING	167
PRACTICE QUESTIONS	168
CHAPTER 8 ■ Support Vector Machines	169
8.1 OPTIMAL SEPARATION	170
8.1.1 The Margin and Support Vectors	170
8.1.2 A Constrained Optimisation Problem	172
8.1.3 Slack Variables for Non-Linearly Separable Problems	175
8.2 KERNELS	176
8.2.1 Choosing Kernels	178
8.2.2 Example: XOR	179
8.3 THE SUPPORT VECTOR MACHINE ALGORITHM	179
8.3.1 Implementation	180
8.3.2 Examples	183
8.4 EXTENSIONS TO THE SVM	184
8.4.1 Multi-Class Classification	184
8.4.2 SVM Regression	186

8.4.3 Other Advances	187
FURTHER READING	187
PRACTICE QUESTIONS	188
CHAPTER 9 ■ Optimisation and Search	189
<hr/>	
9.1 GOING DOWNHILL	190
9.1.1 Taylor Expansion	193
9.2 LEAST-SQUARES OPTIMISATION	194
9.2.1 The Levenberg–Marquardt Algorithm	194
9.3 CONJUGATE GRADIENTS	198
9.3.1 Conjugate Gradients Example	201
9.3.2 Conjugate Gradients and the MLP	201
9.4 SEARCH: THREE BASIC APPROACHES	204
9.4.1 Exhaustive Search	204
9.4.2 Greedy Search	205
9.4.3 Hill Climbing	205
9.5 EXPLOITATION AND EXPLORATION	206
9.6 SIMULATED ANNEALING	207
9.6.1 Comparison	208
FURTHER READING	209
PRACTICE QUESTIONS	209
CHAPTER 10 ■ Evolutionary Learning	211
<hr/>	
10.1 THE GENETIC ALGORITHM (GA)	212
10.1.1 String Representation	213
10.1.2 Evaluating Fitness	213
10.1.3 Population	214
10.1.4 Generating Offspring: Parent Selection	214
10.2 GENERATING OFFSPRING: GENETIC OPERATORS	216
10.2.1 Crossover	216
10.2.2 Mutation	217
10.2.3 Elitism, Tournaments, and Niching	218
10.3 USING GENETIC ALGORITHMS	220
10.3.1 Map Colouring	220
10.3.2 Punctuated Equilibrium	221
10.3.3 Example: The Knapsack Problem	222
10.3.4 Example: The Four Peaks Problem	222
10.3.5 Limitations of the GA	224
10.3.6 Training Neural Networks with Genetic Algorithms	225

10.4	GENETIC PROGRAMMING	225
10.5	COMBINING SAMPLING WITH EVOLUTIONARY LEARNING	227
	FURTHER READING	228
	PRACTICE QUESTIONS	229
CHAPTER 11 ■ Reinforcement Learning		231
<hr/>		
11.1	OVERVIEW	232
11.2	EXAMPLE: GETTING LOST	233
11.2.1	State and Action Spaces	235
11.2.2	Carrots and Sticks: The Reward Function	236
11.2.3	Discounting	237
11.2.4	Action Selection	237
11.2.5	Policy	238
11.3	MARKOV DECISION PROCESSES	238
11.3.1	The Markov Property	238
11.3.2	Probabilities in Markov Decision Processes	239
11.4	VALUES	240
11.5	BACK ON HOLIDAY: USING REINFORCEMENT LEARNING	244
11.6	THE DIFFERENCE BETWEEN SARSA AND Q-LEARNING	245
11.7	USES OF REINFORCEMENT LEARNING	246
	FURTHER READING	247
	PRACTICE QUESTIONS	247
CHAPTER 12 ■ Learning with Trees		249
<hr/>		
12.1	USING DECISION TREES	249
12.2	CONSTRUCTING DECISION TREES	250
12.2.1	Quick Aside: Entropy in Information Theory	251
12.2.2	ID3	251
12.2.3	Implementing Trees and Graphs in Python	255
12.2.4	Implementation of the Decision Tree	255
12.2.5	Dealing with Continuous Variables	257
12.2.6	Computational Complexity	258
12.3	CLASSIFICATION AND REGRESSION TREES (CART)	260
12.3.1	Gini Impurity	260
12.3.2	Regression in Trees	261
12.4	CLASSIFICATION EXAMPLE	261
	FURTHER READING	263
	PRACTICE QUESTIONS	264

CHAPTER 13 ■ Decision by Committee: Ensemble Learning	267
13.1 BOOSTING	268
13.1.1 AdaBoost	269
13.1.2 Stumping	273
13.2 BAGGING	273
13.2.1 Subagging	274
13.3 RANDOM FORESTS	275
13.3.1 Comparison with Boosting	277
13.4 DIFFERENT WAYS TO COMBINE CLASSIFIERS	277
FURTHER READING	279
PRACTICE QUESTIONS	280
CHAPTER 14 ■ Unsupervised Learning	281
14.1 THE K -MEANS ALGORITHM	282
14.1.1 Dealing with Noise	285
14.1.2 The k -Means Neural Network	285
14.1.3 Normalisation	287
14.1.4 A Better Weight Update Rule	288
14.1.5 Example: The Iris Dataset Again	289
14.1.6 Using Competitive Learning for Clustering	290
14.2 VECTOR QUANTISATION	291
14.3 THE SELF-ORGANISING FEATURE MAP	291
14.3.1 The SOM Algorithm	294
14.3.2 Neighbourhood Connections	295
14.3.3 Self-Organisation	297
14.3.4 Network Dimensionality and Boundary Conditions	298
14.3.5 Examples of Using the SOM	300
FURTHER READING	300
PRACTICE QUESTIONS	303
CHAPTER 15 ■ Markov Chain Monte Carlo (MCMC) Methods	305
15.1 SAMPLING	305
15.1.1 Random Numbers	305
15.1.2 Gaussian Random Numbers	306
15.2 MONTE CARLO OR BUST	308
15.3 THE PROPOSAL DISTRIBUTION	310
15.4 MARKOV CHAIN MONTE CARLO	313
15.4.1 Markov Chains	313

15.4.2	The Metropolis–Hastings Algorithm	315
15.4.3	Simulated Annealing (Again)	316
15.4.4	Gibbs Sampling	318
	FURTHER READING	319
	PRACTICE QUESTIONS	320
CHAPTER 16 ■ Graphical Models		321
<hr/>		
16.1	BAYESIAN NETWORKS	322
16.1.1	Example: Exam Fear	323
16.1.2	Approximate Inference	327
16.1.3	Making Bayesian Networks	329
16.2	MARKOV RANDOM FIELDS	330
16.3	HIDDEN MARKOV MODELS (HMMS)	333
16.3.1	The Forward Algorithm	335
16.3.2	The Viterbi Algorithm	337
16.3.3	The Baum–Welch or Forward–Backward Algorithm	339
16.4	TRACKING METHODS	343
16.4.1	The Kalman Filter	343
16.4.2	The Particle Filter	350
	FURTHER READING	355
	PRACTICE QUESTIONS	356
CHAPTER 17 ■ Symmetric Weights and Deep Belief Networks		359
<hr/>		
17.1	ENERGETIC LEARNING: THE HOPFIELD NETWORK	360
17.1.1	Associative Memory	360
17.1.2	Making an Associative Memory	361
17.1.3	An Energy Function	365
17.1.4	Capacity of the Hopfield Network	367
17.1.5	The Continuous Hopfield Network	368
17.2	STOCHASTIC NEURONS — THE BOLTZMANN MACHINE	369
17.2.1	The Restricted Boltzmann Machine	371
17.2.2	Deriving the CD Algorithm	375
17.2.3	Supervised Learning	380
17.2.4	The RBM as a Directed Belief Network	381
17.3	DEEP LEARNING	385
17.3.1	Deep Belief Networks (DBN)	388
	FURTHER READING	393
	PRACTICE QUESTIONS	393

CHAPTER 18 ■ Gaussian Processes	395
18.1 GAUSSIAN PROCESS REGRESSION	397
18.1.1 Adding Noise	398
18.1.2 Implementation	402
18.1.3 Learning the Parameters	403
18.1.4 Implementation	404
18.1.5 Choosing a (set of) Covariance Functions	406
18.2 GAUSSIAN PROCESS CLASSIFICATION	407
18.2.1 The Laplace Approximation	408
18.2.2 Computing the Posterior	408
18.2.3 Implementation	410
FURTHER READING	412
PRACTICE QUESTIONS	413
APPENDIX A ■ Python	415
A.1 INSTALLING PYTHON AND OTHER PACKAGES	415
A.2 GETTING STARTED	415
A.2.1 Python for MATLAB [®] and R users	418
A.3 CODE BASICS	419
A.3.1 Writing and Importing Code	419
A.3.2 Control Flow	420
A.3.3 Functions	420
A.3.4 The <code>doc</code> String	421
A.3.5 <code>map</code> and <code>lambda</code>	421
A.3.6 Exceptions	422
A.3.7 Classes	422
A.4 USING NUMPY AND MATPLOTLIB	423
A.4.1 Arrays	423
A.4.2 Random Numbers	427
A.4.3 Linear Algebra	427
A.4.4 Plotting	427
A.4.5 One Thing to Be Aware of	429
FURTHER READING	430
PRACTICE QUESTIONS	430
Index	431

Prologue to 2nd Edition

There have been some interesting developments in machine learning over the past four years, since the 1st edition of this book came out. One is the rise of Deep Belief Networks as an area of real research interest (and business interest, as large internet-based companies look to snap up every small company working in the area), while another is the continuing work on statistical interpretations of machine learning algorithms. This second one is very good for the field as an area of research, but it does mean that computer science students, whose statistical background can be rather lacking, find it hard to get started in an area that they are sure should be of interest to them. The hope is that this book, focussing on the *algorithms* of machine learning as it does, will help such students get a handle on the ideas, and that it will start them on a journey towards mastery of the relevant mathematics and statistics as well as the necessary programming and experimentation.

In addition, the libraries available for the Python language have continued to develop, so that there are now many more facilities available for the programmer. This has enabled me to provide a simple implementation of the Support Vector Machine that can be used for experiments, and to simplify the code in a few other places. All of the code that was used to create the examples in the book is available at <http://stephenmonika.net/> (in the ‘Book’ tab), and use and experimentation with any of this code, as part of any study on machine learning, is strongly encouraged.

Some of the changes to the book include:

- the addition of two new chapters on two of those new areas: Deep Belief Networks (Chapter 17) and Gaussian Processes (Chapter 18).
- a reordering of the chapters, and some of the material within the chapters, to make a more natural flow.
- the reworking of the Support Vector Machine material so that there is running code and the suggestions of experiments to be performed.
- the addition of Random Forests (as Section 13.3), the Perceptron convergence theorem (Section 3.4.1), a proper consideration of accuracy methods (Section 2.2.4), conjugate gradient optimisation for the MLP (Section 9.3.2), and more on the Kalman filter and particle filter in Chapter 16.
- improved code including better use of naming conventions in Python.
- various improvements in the clarity of explanation and detail throughout the book.

I would like to thank the people who have written to me about various parts of the book, and made suggestions about things that could be included or explained better. I would also like to thank the students at Massey University who have studied the material with me, either as part of their coursework, or as first steps in research, whether in the theory or the application of machine learning. Those that have contributed particularly to the content of the second edition include Nirosha Priyadarshani, James Curtis, Andy Gilman, Örjan

Ekeberg, and the Osnabrück Knowledge-Based Systems Research group, especially Joachim Hertzberg, Sven Albrecht, and Thomas Wieman.

Stephen Marsland
Ashhurst, New Zealand

Prologue to 1st Edition

One of the most interesting features of machine learning is that it lies on the boundary of several different academic disciplines, principally computer science, statistics, mathematics, and engineering. This has been a problem as well as an asset, since these groups have traditionally not talked to each other very much. To make it even worse, the areas where machine learning methods can be applied vary even more widely, from finance to biology and medicine to physics and chemistry and beyond. Over the past ten years this inherent multi-disciplinarity has been embraced and understood, with many benefits for researchers in the field. This makes writing a textbook on machine learning rather tricky, since it is potentially of interest to people from a variety of different academic backgrounds.

In universities, machine learning is usually studied as part of artificial intelligence, which puts it firmly into computer science and—given the focus on algorithms—it certainly fits there. However, understanding why these algorithms work requires a certain amount of statistical and mathematical sophistication that is often missing from computer science undergraduates. When I started to look for a textbook that was suitable for classes of undergraduate computer science and engineering students, I discovered that the level of mathematical knowledge required was (unfortunately) rather in excess of that of the majority of the students. It seemed that there was a rather crucial gap, and it resulted in me writing the first draft of the student notes that have become this book. The emphasis is on the algorithms that make up the machine learning methods, and on understanding how and why these algorithms work. It is intended to be a practical book, with lots of programming examples and is supported by a website that makes available all of the code that was used to make the figures and examples in the book. The website for the book is: <http://stephenmonika.net/MLbook.html>.

For this kind of practical approach, examples in a real programming language are preferred over some kind of pseudocode, since it enables the reader to run the programs and experiment with data without having to work out irrelevant implementation details that are specific to their chosen language. Any computer language can be used for writing machine learning code, and there are very good resources available in many different languages, but the code examples in this book are written in Python. I have chosen Python for several reasons, primarily that it is freely available, multi-platform, relatively nice to use and is becoming a default for scientific computing. If you already know how to write code in any other programming language, then you should not have many problems learning Python. If you don't know how to code at all, then it is an ideal first language as well. Chapter A provides a basic primer on using Python for numerical computing.

Machine learning is a rich area. There are lots of very good books on machine learning for those with the mathematical sophistication to follow them, and it is hoped that this book could provide an entry point to students looking to study the subject further as well as those studying it as part of a degree. In addition to books, there are many resources for machine learning available via the Internet, with more being created all the time. The Machine Learning Open Source Software website at <http://mloss.org/software/> provides links to a host of software in different languages.

There is a very useful resource for machine learning in the UCI Machine Learning Repos-

itory (<http://archive.ics.uci.edu/ml/>). This website holds lots of datasets that can be downloaded and used for experimenting with different machine learning algorithms and seeing how well they work. The repository is going to be the principal source of data for this book. By using these test datasets for experimenting with the algorithms, we do not have to worry about getting hold of suitable data and **preprocessing** it into a suitable form for learning. This is typically a large part of any real problem, but it gets in the way of learning about the algorithms.

I am very grateful to a lot of people who have read sections of the book and provided suggestions, spotted errors, and given encouragement when required. In particular for the first edition, thanks to Zbigniew Nowicki, Joseph Marsland, Bob Hodgson, Patrick Rynhart, Gary Allen, Linda Chua, Mark Bebbington, JP Lewis, Tom Duckett, and Monika Nowicki. Thanks especially to Jonathan Shapiro, who helped me discover machine learning and who may recognise some of his own examples.

Stephen Marsland
Ashhurst, New Zealand

Introduction

Suppose that you have a website selling software that you've written. You want to make the website more personalised to the user, so you start to collect data about visitors, such as their computer type/operating system, web browser, the country that they live in, and the time of day they visited the website. You can get this data for any visitor, and for people who actually buy something, you know what they bought, and how they paid for it (say PayPal or a credit card). So, for each person who buys something from your website, you have a list of data that looks like (computer type, web browser, country, time, software bought, how paid). For instance, the first three pieces of data you collect could be:

- Macintosh OS X, Safari, UK, morning, SuperGame1, credit card
- Windows XP, Internet Explorer, USA, afternoon, SuperGame1, PayPal
- Windows Vista, Firefox, NZ, evening, SuperGame2, PayPal

Based on this data, you would like to be able to populate a 'Things You Might Be Interested In' box within the webpage, so that it shows software that might be relevant to each visitor, based on the data that you can access while the webpage loads, i.e., computer and OS, country, and the time of day. Your hope is that as more people visit your website and you store more data, you will be able to identify trends, such as that Macintosh users from New Zealand (NZ) love your first game, while Firefox users, who are often more knowledgeable about computers, want your automatic download application and virus/internet worm detector, etc.

Once you have collected a large set of such data, you start to examine it and work out what you can do with it. The problem you have is one of **prediction**: given the data you have, predict what the next person will buy, and the reason that you think that it might work is that people who seem to be similar often act similarly. So how can you actually go about solving the problem? This is one of the fundamental problems that this book tries to solve. It is an example of what is called **supervised learning**, because we know what the right answers are for some examples (the software that was actually bought) so we can give the learner some examples where we know the right answer. We will talk about supervised learning more in Section 1.3.

1.1 IF DATA HAD MASS, THE EARTH WOULD BE A BLACK HOLE

Around the world, computers capture and store terabytes of data every day. Even leaving aside your collection of MP3s and holiday photographs, there are computers belonging to shops, banks, hospitals, scientific laboratories, and many more that are storing data incessantly. For example, banks are building up pictures of how people spend their money,

hospitals are recording what treatments patients are on for which ailments (and how they respond to them), and engine monitoring systems in cars are recording information about the engine in order to detect when it might fail. The challenge is to do something useful with this data: if the bank's computers can learn about spending patterns, can they detect credit card fraud quickly? If hospitals share data, then can treatments that don't work as well as expected be identified quickly? Can an intelligent car give you early warning of problems so that you don't end up stranded in the worst part of town? These are some of the questions that machine learning methods can be used to answer.

Science has also taken advantage of the ability of computers to store massive amounts of data. Biology has led the way, with the ability to measure gene expression in DNA microarrays producing immense datasets, along with protein transcription data and phylogenetic trees relating species to each other. However, other sciences have not been slow to follow. Astronomy now uses digital telescopes, so that each night the world's observatories are storing incredibly high-resolution images of the night sky; around a terabyte per night. Equally, medical science stores the outcomes of medical tests from measurements as diverse as magnetic resonance imaging (MRI) scans and simple blood tests. The explosion in stored data is well known; the challenge is to do something useful with that data. The Large Hadron Collider at CERN apparently produces about 25 petabytes of data per year.

The size and complexity of these datasets mean that humans are unable to extract useful information from them. Even the way that the data is stored works against us. Given a file full of numbers, our minds generally turn away from looking at them for long. Take some of the same data and plot it in a graph and we can do something. Compare the table and graph shown in Figure 1.1: the graph is rather easier to look at and deal with. Unfortunately, our three-dimensional world doesn't let us do much with data in higher dimensions, and even the simple webpage data that we collected above has four different features, so if we plotted it with one dimension for each feature we'd need four dimensions! There are two things that we can do with this: reduce the number of dimensions (until our simple brains can deal with the problem) or use computers, which don't know that high-dimensional problems are difficult, and don't get bored with looking at massive data files of numbers. The two pictures in Figure 1.2 demonstrate one problem with reducing the number of dimensions (more technically, **projecting it into fewer dimensions**), which is that it can hide useful information and make things look rather strange. This is one reason why **machine learning** is becoming so popular — the problems of our human limitations go away if we can make computers do the dirty work for us. There is one other thing that can help if the number of dimensions is not too much larger than three, which is to use **glyphs** that use other representations, such as size or colour of the datapoints to represent information about some other dimension, but this does not help if the dataset has 100 dimensions in it.

In fact, you have probably interacted with machine learning algorithms at some time. They are used in many of the software programs that we use, such as Microsoft's infamous paperclip in Office (maybe not the most positive example), spam filters, voice recognition software, and lots of computer games. They are also part of automatic number-plate recognition systems for petrol station security cameras and toll roads, are used in some anti-skid braking and vehicle stability systems, and they are even part of the set of algorithms that decide whether a bank will give you a loan.

The attention-grabbing title to this section would only be true if data was very heavy. It is very hard to work out how much data there actually is in all of the world's computers, but it was estimated in 2012 that was about 2.8 zettabytes (2.8×10^{21} bytes), up from about 160 exabytes (160×10^{18} bytes) of data that were created and stored in 2006, and projected to reach 40 zettabytes by 2020. However, to make a black hole the size of the earth would

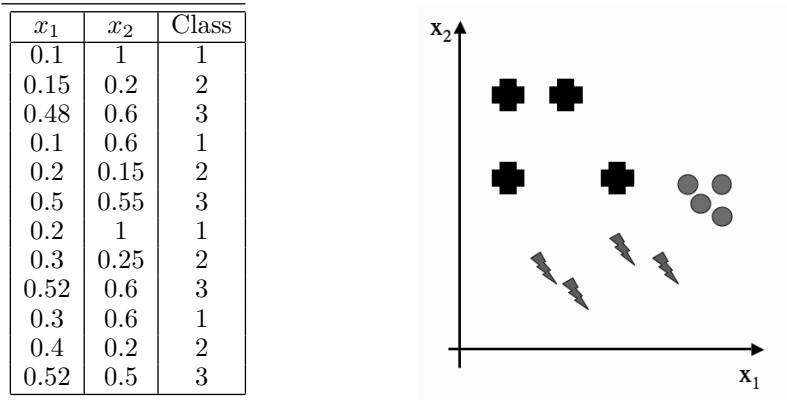


FIGURE 1.1 A set of datapoints as numerical values and as points plotted on a graph. It is easier for us to visualise data than to see it in a table, but if the data has more than three dimensions, we can't view it all at once.



FIGURE 1.2 Two views of the same two wind turbines (Te Apiti wind farm, Ashhurst, New Zealand) taken at an angle of about 30° to each other. The two-dimensional projections of three-dimensional objects hides information.

take a mass of about 40×10^{35} grams. So data would have to be so heavy that you couldn't possibly lift a data pen, let alone a computer before the section title were true! However, and more interestingly for machine learning, the same report that estimated the figure of 2.8 zettabytes (*'Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East'* by John Gantz and David Reinsel and sponsored by EMC Corporation) also reported that while a quarter of this data could produce useful information, only around 3% of it was tagged, and less than 0.5% of it was actually used for analysis!

1.2 LEARNING

Before we delve too much further into the topic, let's step back and think about what learning actually is. The key concept that we will need to think about for our machines is **learning from data**, since data is what we have; terabytes of it, in some cases. However, it isn't too large a step to put that into human behavioural terms, and talk about **learning from experience**. Hopefully, we all agree that humans and other animals can display behaviours that we label as intelligent by learning from experience. Learning is what gives us flexibility in our life; the fact that we can adjust and adapt to new circumstances, and learn new tricks, no matter how old a dog we are! The important parts of animal learning for this book are **remembering**, **adapting**, and **generalising**: recognising that last time we were in this situation (saw this data) we tried out some particular action (gave this output) and it worked (was correct), so we'll try it again, or it didn't work, so we'll try something different. The last word, generalising, is about recognising similarity between different situations, so that things that applied in one place can be used in another. This is what makes learning useful, because we can use our knowledge in lots of different places.

Of course, there are plenty of other bits to intelligence, such as **reasoning**, and logical deduction, but we won't worry too much about those. We are interested in the most fundamental parts of intelligence—learning and adapting—and how we can model them in a computer. There has also been a lot of interest in making computers reason and deduce facts. This was the basis of most early **Artificial Intelligence**, and is sometimes known as **symbolic processing** because the computer manipulates symbols that reflect the environment. In contrast, machine learning methods are sometimes called **subsymbolic** because no symbols or symbolic manipulation are involved.

1.2.1 Machine Learning

Machine learning, then, is about making computers **modify** or **adapt** their actions (whether these actions are making predictions, or controlling a robot) so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones. Imagine that you are playing Scrabble (or some other game) against a computer. You might beat it every time in the beginning, but after lots of games it starts beating you, until finally you never win. Either you are getting worse, or the computer is learning how to win at Scrabble. Having learnt to beat you, it can go on and use the same strategies against other players, so that it doesn't start from scratch with each new player; this is a form of generalisation.

It is only over the past decade or so that the inherent multi-disciplinarity of machine learning has been recognised. It merges ideas from neuroscience and biology, statistics, mathematics, and physics, to make computers learn. There is a fantastic existence proof that learning is possible, which is the bag of water and electricity (together with a few trace chemicals) sitting between your ears. In Section 3.1 we will have a brief peek inside and see

if there is anything we can borrow/steal in order to make machine learning algorithms. It turns out that there is, and **neural networks** have grown from exactly this, although even their own father wouldn't recognise them now, after the developments that have seen them reinterpreted as statistical learners. Another thing that has driven the change in direction of machine learning research is **data mining**, which looks at the extraction of useful information from massive datasets (by men with computers and pocket protectors rather than pickaxes and hard hats), and which requires efficient algorithms, putting more of the emphasis back onto computer science.

The **computational complexity** of the machine learning methods will also be of interest to us since what we are producing is **algorithms**. It is particularly important because we might want to use some of the methods on very large datasets, so algorithms that have high-degree polynomial complexity in the size of the dataset (or worse) will be a problem. The complexity is often broken into two parts: the complexity of training, and the complexity of applying the trained algorithm. Training does not happen very often, and is not usually time critical, so it can take longer. However, we often want a decision about a test point quickly, and there are potentially lots of test points when an algorithm is in use, so this needs to have low computational cost.

1.3 TYPES OF MACHINE LEARNING

In the example that started the chapter, your webpage, the aim was to predict what software a visitor to the website might buy based on information that you can collect. There are a couple of interesting things in there. The first is the data. It might be useful to know what software visitors have bought before, and how old they are. However, it is not possible to get that information from their web browser (even cookies can't tell you how old somebody is), so you can't use that information. Picking the variables that you want to use (which are called **features** in the jargon) is a very important part of finding good solutions to problems, and something that we will talk about in several places in the book. Equally, choosing how to process the data can be important. This can be seen in the example in the time of access. Your computer can store this down to the nearest millisecond, but that isn't very useful, since you would like to spot similar patterns between users. For this reason, in the example above I chose to **quantise** it down to one of the set **morning, afternoon, evening, night**; obviously I need to ensure that these times are correct for their time zones, too.

We are going to loosely define learning as meaning **getting better at some task through practice**. This leads to a couple of vital questions: how does the computer know whether it is getting better or not, and how does it know how to improve? There are several different possible answers to these questions, and they produce different types of machine learning. For now we will consider the question of knowing whether or not the machine is learning. We can tell the algorithm the correct answer for a problem so that it gets it right next time (which is what would happen in the webpage example, since we know what software the person bought). We hope that we only have to tell it a few right answers and then it can 'work out' how to get the correct answers for other problems (**generalise**). Alternatively, we can tell it whether or not the answer was correct, but not how to find the correct answer, so that it has to **search** for the right answer. A variant of this is that we give a score for the answer, according to how correct it is, rather than just a 'right or wrong' response. Finally, we might not have any correct answers; we just want the algorithm to find inputs that have something in common.

These different answers to the question provide a useful way to classify the different algorithms that we will be talking about:

Supervised learning A training set of examples with the correct responses (**targets**) is provided and, based on this training set, the algorithm **generalises** to respond correctly to all possible inputs. This is also called learning from **exemplars**.

Unsupervised learning Correct responses are not provided, but instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are **categorised** together. The statistical approach to unsupervised learning is known as **density estimation**.

Reinforcement learning This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right. Reinforcement learning is sometime called learning with a **critic** because of this monitor that scores the answer, but does not suggest improvements.

Evolutionary learning Biological evolution can be seen as a learning process: biological organisms adapt to improve their survival rates and chance of having offspring in their environment. We'll look at how we can model this in a computer, using an idea of **fitness**, which corresponds to a score for how good the current solution is.

The most common type of learning is supervised learning, and it is going to be the focus of the next few chapters. So, before we get started, we'll have a look at what it is, and the kinds of problems that can be solved using it.

1.4 SUPERVISED LEARNING

As has already been suggested, the webpage example is a typical problem for supervised learning. There is a set of data (the **training data**) that consists of a set of **input** data that has **target** data, which is the answer that the algorithm should produce, attached. This is usually written as a set of data $(\mathbf{x}_i, \mathbf{t}_i)$, where the inputs are \mathbf{x}_i , the targets are \mathbf{t}_i , and the i index suggests that we have lots of pieces of data, indexed by i running from 1 to some upper limit N . Note that the inputs and targets are written in boldface font to signify vectors, since each piece of data has values for several different features; the notation used in the book is described in more detail in Section 2.1. If we had examples of every possible piece of input data, then we could put them together into a big look-up table, and there would be no need for machine learning at all. The thing that makes machine learning better than that is **generalisation**: the algorithm should produce sensible outputs for inputs that weren't encountered during learning. This also has the result that the algorithm can deal with **noise**, which is small inaccuracies in the data that are inherent in measuring any real world process. It is hard to specify rigorously what generalisation means, but let's see if an example helps.

1.4.1 Regression

Suppose that I gave you the following datapoints and asked you to tell me the value of the output (which we will call y since it is not a target datapoint) when $x = 0.44$ (here, x , t , and y are not written in boldface font since they are **scalars**, as opposed to vectors).

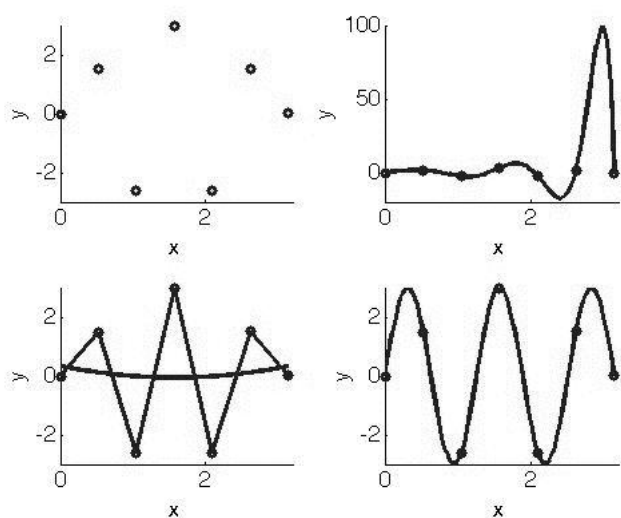


FIGURE 1.3 *Top left:* A few datapoints from a sample problem. *Bottom left:* Two possible ways to predict the values between the known datapoints: connecting the points with straight lines, or using a cubic approximation (which in this case misses all of the points). *Top and bottom right:* Two more complex approximators (see the text for details) that pass through the points, although the lower one is rather better than the top.

x	t
0	0
0.5236	1.5
1.0472	-2.5981
1.5708	3.0
2.0944	-2.5981
2.6180	1.5
3.1416	0

Since the value $x = 0.44$ isn't in the examples given, you need to find some way to **predict** what value it has. You assume that the values come from some sort of function, and try to find out what the function is. Then you'll be able to give the output value y for any given value of x . This is known as a **regression** problem in statistics: fit a mathematical function describing a curve, so that the curve passes as close as possible to all of the datapoints. It is generally a problem of **function approximation** or **interpolation**, working out the value between values that we know.

The problem is how to work out what function to choose. Have a look at Figure 1.3. The top-left plot shows a plot of the 7 values of x and y in the table, while the other plots show different attempts to fit a curve through the datapoints. The bottom-left plot shows two possible answers found by using straight lines to connect up the points, and also what happens if we try to use a cubic function (something that can be written as $ax^3 + bx^2 + cx + d = 0$). The top-right plot shows what happens when we try to match the function using a different polynomial, this time of the form $ax^{10} + bx^9 + \dots + jx + k = 0$,

and finally the bottom-right plot shows the function $y = 3 \sin(5x)$. Which of these functions would you choose?

The straight-line approximation probably isn't what we want, since it doesn't tell us much about the data. However, the cubic plot on the same set of axes is terrible: it doesn't get anywhere near the datapoints. What about the plot on the top-right? It looks like it goes through all of the datapoints exactly, but it is very wiggly (look at the value on the y -axis, which goes up to 100 instead of around three, as in the other figures). In fact, the data were made with the sine function plotted on the bottom-right, so that is the correct answer in this case, but the algorithm doesn't know that, and to it the two solutions on the right both look equally good. The only way we can tell which solution is better is to test how well they generalise. We pick a value that is between our datapoints, use our curves to predict its value, and see which is better. This will tell us that the bottom-right curve is better in the example.

So one thing that our machine learning algorithms can do is interpolate between datapoints. This might not seem to be intelligent behaviour, or even very difficult in two dimensions, but it is rather harder in higher dimensional spaces. The same thing is true of the other thing that our algorithms will do, which is **classification**—grouping examples into different classes—which is discussed next. However, the algorithms are learning by our definition if they adapt so that their performance improves, and it is surprising how often real problems that we want to solve can be reduced to classification or regression problems.

1.4.2 Classification

The classification problem consists of taking input vectors and deciding which of N classes they belong to, based on training from **exemplars** of each class. The most important point about the classification problem is that it is discrete — each example belongs to precisely one class, and the set of classes covers the whole possible output space. These two constraints are not necessarily realistic; sometimes examples might belong partially to two different classes. There are **fuzzy classifiers** that try to solve this problem, but we won't be talking about them in this book. In addition, there are many places where we might not be able to categorise every possible input. For example, consider a vending machine, where we use a neural network to learn to recognise all the different coins. We train the classifier to recognise all New Zealand coins, but what if a British coin is put into the machine? In that case, the classifier will identify it as the New Zealand coin that is closest to it in appearance, but this is not really what is wanted: rather, the classifier should identify that it is not one of the coins it was trained on. This is called **novelty detection**. For now we'll assume that we will not receive inputs that we cannot classify accurately.

Let's consider how to set up a coin classifier. When the coin is pushed into the slot, the machine takes a few measurements of it. These could include the diameter, the weight, and possibly the shape, and are the **features** that will generate our input vector. In this case, our input vector will have three elements, each of which will be a number showing the measurement of that feature (choosing a number to represent the shape would involve an **encoding**, for example that 1=circle, 2=hexagon, etc.). Of course, there are many other features that we could measure. If our vending machine included an atomic absorption spectroscope, then we could estimate the density of the material and its composition, or if it had a camera, we could take a photograph of the coin and feed that image into the classifier. The question of which features to choose is not always an easy one. We don't want to use too many inputs, because that will make the training of the classifier take longer (and also, as the number of input dimensions grows, the number of datapoints required increases



FIGURE 1.4 The New Zealand coins.

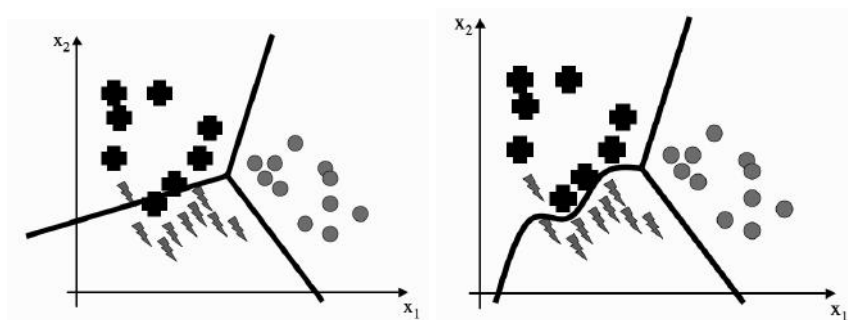


FIGURE 1.5 *Left:* A set of straight line decision boundaries for a classification problem. *Right:* An alternative set of decision boundaries that separate the pluses from the lightning strikes better, but requires a line that isn't straight.

faster; this is known as the curse of dimensionality and will be discussed in Section 2.1.2), but we need to make sure that we can reliably separate the classes based on those features. For example, if we tried to separate coins based only on colour, we wouldn't get very far, because the 20c and 50c coins are both silver and the \$1 and \$2 coins both bronze. However, if we use colour and diameter, we can do a pretty good job of the coin classification problem for NZ coins. There are some features that are entirely useless. For example, knowing that the coin is circular doesn't tell us anything about NZ coins, which are all circular (see Figure 1.4). In other countries, though, it could be very useful.

The methods of performing classification that we will see during this book are very different in the ways that they learn about the solution; in essence they aim to do the same thing: find decision boundaries that can be used to separate out the different classes. Given the features that are used as inputs to the classifier, we need to identify some values of those features that will enable us to decide which class the current input is in. Figure 1.5 shows a set of 2D inputs with three different classes shown, and two different decision boundaries; on the left they are straight lines, and are therefore simple, but don't categorise as well as the non-linear curve on the right.

Now that we have seen these two types of problem, let's take a look at the whole process of machine learning from the practitioner's viewpoint.