



C o m m u n i t y E x p e r i e n c e D i s t i l l e d

Scala for Machine Learning

Leverage Scala and Machine Learning to construct and study systems that can learn from data

Patrick R. Nicolas

[PACKT] open source*
PUBLISHING community experience distilled

Scala for Machine Learning

Leverage Scala and Machine Learning to construct and study systems that can learn from data

Patrick R. Nicolas



BIRMINGHAM - MUMBAI

Scala for Machine Learning

Copyright © 2014 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: December 2014

Production reference: 1121214

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78355-874-2

www.packtpub.com

Credits

Author

Patrick R. Nicolas

Project Coordinator

Danuta Jones

Reviewers

Subhajit Datta
Rui Gonçalves
Patricia Hoffman, PhD
Md Zahidul Islam

Proofreaders

Simran Bhogal
Maria Gould
Paul Hindle
Elinor Perry-Smith
Chris Smith

Commissioning Editor

Owen Roberts

Indexer

Mariammal Chettiyar

Acquisition Editor

Owen Roberts

Graphics

Sheetal Aute
Valentina D'silva
Disha Haria
Abhinash Sahu

Content Development Editor

Mohammed Fahad

Production Coordinator

Arvindkumar Gupta

Technical Editors

Madhuri Das
Taabish Khan

Cover Work

Arvindkumar Gupta

Copy Editors

Janbal Dharmaraj
Vikrant Phadkay

About the Author

Patrick R. Nicolas is a lead R&D engineer at Dell in Santa Clara, California. He has 25 years of experience in software engineering and building large-scale applications in C++, Java, and Scala, and has held several managerial positions. His interests include real-time analytics, modeling, and optimization.

Special thanks to the Packt Publishing team: Mohammed Fahad for his patience and encouragement, Owen Roberts for the opportunity, and the reviewers for their guidance and dedication.

About the Reviewers

Subhajit Datta is a passionate software developer.

He did his Bachelor of Engineering in Information Technology (BE in IT) from Indian Institute of Engineering Science and Technology, Shibpur (IIEST, Shibpur), formerly known as Bengal Engineering and Science University, Shibpur.

He completed his Master of Technology in Computer Science and Engineering (MTech CSE) from Indian Institute of Technology Bombay (IIT Bombay); his thesis focused on topics in natural language processing.

He has experience working in the investment banking domain and web application domain, and is a polyglot having worked on Java, Scala, Python, Unix shell scripting, VBScript, JavaScript, C#.Net, and PHP. He is interested in learning and applying new and different technologies.

He believes that choosing the right programming language, tool, and framework for the problem at hand is more important than trying to fit all problems in one technology.

He also has experience working in the Waterfall and Agile processes. He is excited about the Agile software development processes.

Rui Gonçalves is an all-round, hardworking, and dedicated software engineer. He is an enthusiast of software architecture, programming paradigms, algorithms, and data structures with the ambition of developing products and services that have a great impact on society.

He currently works at ShiftForward, where he is a software engineer in the online advertising field. He is focused on designing and implementing highly efficient, concurrent, and scalable systems as well as machine learning solutions. In order to achieve this, he uses Scala as the main development language of these systems on a day-to-day basis.

Patricia Hoffman, PhD, is a consultant at iCube Consulting Service Inc., with over 25 years of experience in modeling and simulation, of which the last six years concentrated on machine learning and data mining technologies. Her software development experience ranges from modeling stochastic partial differential equations to image processing. She is currently an adjunct faculty member at International Technical University, teaching machine learning courses. She also teaches machine learning and data mining at the University of California, Santa Cruz – Silicon Valley Campus. She was Chair of Association for Computing Machinery of the Data Mining Special Interest Group for the San Francisco Bay area for 5 years, organizing monthly lectures and five data mining conferences with over 350 participants.

Patricia has a long list of significant accomplishments. She developed the architecture and software development plan for a collaborative recommendation system while consulting as a data mining expert for Quantum Capital. While consulting for Revolution Analytics, she developed training materials for interfacing the R statistical language with IBM's Netezza data warehouse appliance.

She has also set up the systems used for communication and software development along with technical coordination for GTECH, a medical device start-up.

She has also technically directed, produced, and managed operations concepts and architecture analysis for hardware, software, and firmware. She has performed risk assessments and has written qualification letters, proposals, system specs, and interface control documents. Also, she has coordinated with subcontractors, associate contractors, and various Lockheed departments to produce analysis, documents, technology demonstrations, and integrated systems. She was the Chief Systems Engineer for a \$12 million image processing workstation development, and had scored 100 percent from the customer.

The various contributions of Patricia to the publications field are as follows:

- *A unified view on the rotational symmetry of equilibria of nematic polymers, dipolar nematic polymers, and polymers in higher dimensional space, Communications in Mathematical Sciences, Volume 6, 949-974*
- She worked as a technical editor on the book *Machine Learning in Action*, Peter Harrington, Manning Publications Co.
- *A Distributed Architecture for the C3 I (Command, Control, Communications, and Intelligence) Collection Management Expert System*, with Allen Rude, AIC Lockheed
- A book review of computer-supported cooperative work, *ACM/SIGCHI Bulletin, Volume 21, Issue 2, pages 125-128, ISSN:0736-6906, 1989*

Md Zahidul Islam is a software developer working for HSI Health and lives in Concord, California, with his wife.

He has a passion for functional programming, machine learning, and working with data. He is currently working with Scala, Apache Spark, MLlib, Ruby on Rails, ElasticSearch, MongoDB, and Backbone.js. Earlier in his career, he worked with C#, ASP.NET, and everything around the .NET ecosystem.

I would like to thank my wife, Sandra, who lovingly supports me in everything I do. I'd also like to thank Packt Publishing and its staff for the opportunity to contribute to this book.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit www.PacktPub.com.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

To Jennifer, for her kindness and support throughout this long journey.

Table of Contents

Preface	1
Chapter 1: Getting Started	9
Mathematical notation for the curious	10
Why machine learning?	10
Classification	10
Prediction	11
Optimization	11
Regression	11
Why Scala?	11
Abstraction	11
Scalability	12
Configurability	13
Maintainability	14
Computation on demand	14
Model categorization	14
Taxonomy of machine learning algorithms	15
Unsupervised learning	15
Clustering	15
Dimension reduction	16
Supervised learning	16
Generative models	16
Discriminative models	17
Reinforcement learning	18
Tools and frameworks	19
Java	19
Scala	20
Apache Commons Math	20
Description	20

Table of Contents

Licensing	20
Installation	21
JFreeChart	21
Description	21
Licensing	21
Installation	22
Other libraries and frameworks	22
Source code	22
Context versus view bounds	23
Presentation	23
Primitives and implicits	24
Primitive types	24
Type conversions	24
Operators	25
Immutability	25
Performance of Scala iterators	26
Let's kick the tires	26
Overview of computational workflows	26
Writing a simple workflow	28
Selecting a dataset	28
Loading the dataset	29
Preprocessing the dataset	30
Creating a model (learning)	34
Classify the data	36
Summary	37
Chapter 2: Hello World!	39
Modeling	39
A model by any other name	39
Model versus design	41
Selecting a model's features	41
Extracting features	42
Designing a workflow	42
The computational framework	44
The pipe operator	44
Monadic data transformation	45
Dependency injection	46
Workflow modules	48
The workflow factory	49
Examples of workflow components	51
The preprocessing module	51
The clustering module	52

Assessing a model	54
Validation	54
Key metrics	54
Implementation	56
K-fold cross-validation	57
Bias-variance decomposition	58
Overfitting	61
Summary	62
Chapter 3: Data Preprocessing	63
Time series	63
Moving averages	66
The simple moving average	67
The weighted moving average	68
The exponential moving average	69
Fourier analysis	73
Discrete Fourier transform (DFT)	73
DFT-based filtering	79
Detection of market cycles	82
The Kalman filter	85
The state space estimation	86
The transition equation	86
The measurement equation	87
The recursive algorithm	87
Prediction	89
Correction	91
Kalman smoothing	92
Experimentation	93
Alternative preprocessing techniques	97
Summary	97
Chapter 4: Unsupervised Learning	99
Clustering	100
K-means clustering	101
Measuring similarity	101
Overview of the K-means algorithm	103
Step 1 – cluster configuration	103
Step 2 – cluster assignment	107
Step 3 – iterative reconstruction	108
Curse of dimensionality	109
Experiment	111
Tuning the number of clusters	114
Validation	117

Expectation-maximization (EM) algorithm	118
Gaussian mixture model	119
EM overview	120
Implementation	120
Testing	123
Online EM	126
Dimension reduction	126
Principal components analysis (PCA)	127
Algorithm	128
Implementation	129
Test case	130
Evaluation	131
Other dimension reduction techniques	133
Performance considerations	133
K-means	133
EM	134
PCA	134
Summary	135
Chapter 5: Naïve Bayes Classifiers	137
Probabilistic graphical models	137
Naïve Bayes classifiers	139
Introducing the multinomial Naïve Bayes	139
Formalism	141
The frequentist perspective	142
The predictive model	144
The zero-frequency problem	145
Implementation	145
Software design	145
Training	146
Classification	151
Labeling	152
Results	154
Multivariate Bernoulli classification	155
Model	155
Implementation	156
Naïve Bayes and text mining	156
Basics of information retrieval	158
Implementation	159
Extraction of terms	160
Scoring of terms	161
Testing	163
Retrieving textual information	163
Evaluation	166

Pros and cons	168
Summary	168
Chapter 6: Regression and Regularization	169
Linear regression	169
One-variate linear regression	170
Implementation	170
Test case	171
Ordinary least squares (OLS) regression	173
Design	173
Implementation	174
Test case 1 – trending	175
Test case 2 – features selection	178
Regularization	184
L_n roughness penalty	184
The ridge regression	186
Implementation	186
The test case	188
Numerical optimization	191
The logistic regression	192
The logit function	192
Binomial classification	193
Software design	196
The training workflow	197
Configuring the least squares optimizer	198
Computing the Jacobian matrix	199
Defining the exit conditions	200
Defining the least squares problem	201
Minimizing the loss function	201
Test	202
Classification	203
Summary	205
Chapter 7: Sequential Data Models	207
Markov decision processes	207
The Markov property	208
The first-order discrete Markov chain	208
The hidden Markov model (HMM)	209
Notation	211
The lambda model	212
HMM execution state	214
Evaluation (CF-1)	216
Alpha class (the forward variable)	217
Beta class (the backward variable)	220

Training (CF-2)	222
Baum-Welch estimator (EM)	222
Decoding (CF-3)	226
The Viterbi algorithm	226
Putting it all together	228
Test case	230
The hidden Markov model for time series analysis	232
Conditional random fields	232
Introduction to CRF	233
Linear chain CRF	235
CRF and text analytics	237
The feature functions model	238
Software design	240
Implementation	241
Building the training set	242
Generating tags	243
Extracting data sequences	244
CRF control parameters	244
Putting it all together	245
Tests	246
The training convergence profile	247
Impact of the size of the training set	247
Impact of the L_2 regularization factor	248
Comparing CRF and HMM	249
Performance consideration	250
Summary	250
Chapter 8: Kernel Models and Support Vector Machines	251
Kernel functions	252
Overview	252
Common discriminative kernels	254
The support vector machine (SVM)	256
The linear SVM	256
The separable case (hard margin)	257
The nonseparable case (soft margin)	258
The nonlinear SVM	260
Max-margin classification	260
The kernel trick	261
Support vector classifier (SVC)	262
The binary SVC	262
LIBSVM	262
Software design	263
Configuration parameters	264
SVM implementation	267

C-penalty and margin	269
Kernel evaluation	272
Application to risk analysis	277
Anomaly detection with one-class SVC	282
Support vector regression (SVR)	284
Overview	284
SVR versus linear regression	285
Performance considerations	288
Summary	288
Chapter 9: Artificial Neural Networks	289
Feed-forward neural networks (FFNN)	289
The Biological background	290
The mathematical background	291
The multilayer perceptron (MLP)	293
The activation function	294
The network architecture	295
Software design	296
Model definition	297
Layers	298
Synapses	299
Connections	299
Training cycle/epoch	300
Step 1 – input forward propagation	301
Step 2 – sum of squared errors	305
Step 3 – error backpropagation	305
Step 4 – synapse/weights adjustment	308
Step 5 – convergence criteria	309
Configuration	309
Putting all together	310
Training strategies and classification	312
Online versus batch training	312
Regularization	313
Model instantiation	313
Prediction	314
Evaluation	315
Impact of learning rate	315
Impact of the momentum factor	316
Test case	317
Implementation	319
Models evaluation	321
Impact of hidden layers architecture	323
Benefits and limitations	324
Summary	326

Chapter 10: Genetic Algorithms	327
Evolution	327
The origin	328
NP problems	328
Evolutionary computing	329
Genetic algorithms and machine learning	330
Genetic algorithm components	330
Encodings	331
Value encoding	331
Predicate encoding	332
Solution encoding	333
The encoding scheme	334
Genetic operators	335
Selection	336
Crossover	338
Mutation	339
Fitness score	340
Implementation	340
Software design	340
Key components	341
Selection	344
Controlling population growth	345
GA configuration	345
Crossover	345
Population	346
Chromosomes	347
Genes	348
Mutation	349
Population	349
Chromosomes	349
Genes	349
The reproduction cycle	350
GA for trading strategies	351
Definition of trading strategies	352
Trading operators	353
The cost/unfitness function	353
Trading signals	354
Trading strategies	355
Signal encoding	356
Test case	357
Data extraction	358
Initial population	358
Configuration	359
GA instantiation	359

GA execution	360
Tests	360
Advantages and risks of genetic algorithms	363
Summary	364
Chapter 11: Reinforcement Learning	365
Introduction	365
The problem	366
A solution – Q-learning	366
Terminology	367
Concept	368
Value of policy	369
Bellman optimality equations	370
Temporal difference for model-free learning	371
Action-value iterative update	372
Implementation	373
Software design	373
States and actions	374
Search space	375
Policy and action-value	376
The Q-learning training	378
Tail recursion to the rescue	380
Prediction	381
Option trading using Q-learning	382
Option property	383
Option model	384
Function approximation	385
Constrained state-transition	386
Putting it all together	387
Evaluation	389
Pros and cons of reinforcement learning	391
Learning classifier systems	391
Introduction to LCS	392
Why LCS	393
Terminology	394
Extended learning classifier systems (XCS)	395
XCS components	396
Application to portfolio management	396
XCS core data	398
XCS rules	399
Covering	401
Example of implementation	401
Benefits and limitation of learning classifier systems	402
Summary	403

Chapter 12: Scalable Frameworks	405
Overview	406
Scala	407
Controlling object creation	407
Parallel collections	407
Processing a parallel collection	408
Benchmark framework	409
Performance evaluation	410
Scalability with Actors	413
The Actor model	413
Partitioning	415
Beyond actors – reactive programming	415
Akka	415
Master-workers	417
Messages exchange	417
Worker actors	418
The workflow controller	419
The master Actor	419
Master with routing	421
Distributed discrete Fourier transform	422
Limitations	425
Futures	425
The Actor life cycle	426
Blocking on futures	426
Handling future callbacks	428
Putting all together	430
Apache Spark	431
Why Spark	432
Design principles	433
In-memory persistency	433
Laziness	433
Transforms and Actions	434
Shared variables	436
Experimenting with Spark	437
Deploying Spark	437
Using Spark shell	438
MLlib	439
RDD generation	439
K-means using Spark	440
Performance evaluation	442
Tuning parameters	442
Tests	443
Performance considerations	444
Pros and cons	445
Oxdata Sparkling Water	446
Summary	446

Appendix A: Basic Concepts	447
Scala programming	447
List of libraries	447
Format of code snippets	448
Encapsulation	449
Class constructor template	449
Companion objects versus case classes	450
Enumerations versus case classes	450
Overloading	451
Design template for classifiers	452
Data extraction	453
Data sources	454
Extraction of documents	455
Matrix class	456
Mathematics	457
Linear algebra	457
QR Decomposition	458
LU factorization	458
LDL decomposition	458
Cholesky factorization	458
Singular value decomposition	459
Eigenvalue decomposition	459
Algebraic and numerical libraries	459
First order predicate logic	460
Jacobian and Hessian matrices	461
Summary of optimization techniques	462
Gradient descent methods	462
Quasi-Newton algorithms	463
Nonlinear least squares minimization	464
Lagrange multipliers	465
Overview of dynamic programming	466
Finances 101	467
Fundamental analysis	467
Technical analysis	468
Terminology	468
Trading signals and strategy	469
Price patterns	471
Options trading	471
Financial data sources	472
Suggested online courses	473
References	473
Index	475

Preface

Not a single day passes by that we do not hear about Big Data in the news media, technical conferences, and even coffee shops. The ever-increasing amount of data collected in process monitoring, research, or simple human behavior becomes valuable only if you extract knowledge from it. Machine learning is the essential tool to mine data for gold (knowledge).

This book covers the "what", "why", and "how" of machine learning:

- What are the objectives and the mathematical foundation of machine learning?
- Why is Scala the ideal programming language to implement machine learning algorithms?
- How can you apply machine learning to solve real-world problems?

Throughout this book, machine learning algorithms are described with diagrams, mathematical formulation, and documented snippets of Scala code, allowing you to understand these key concepts in your own unique way.

What this book covers

Chapter 1, Getting Started, introduces the basic concepts of statistical analysis, classification, regression, prediction, clustering, and optimization. This chapter covers the Scala languages features and libraries, followed by the implementation of a simple application.

Chapter 2, Hello World!, describes a typical workflow for classification, the concept of bias/variance trade-off, and validation using the Scala dependency injection applied to the technical analysis of financial markets.

Chapter 3, Data Preprocessing, covers time series analyses and leverages Scala to implement data preprocessing and smoothing techniques such as moving averages, discrete Fourier transform, and the Kalman recursive filter.

Chapter 4, Unsupervised Learning, focuses on the implementation of some of the most widely used clustering techniques, such as K-means, the expectation-maximization, and the principal component analysis as a dimension reduction method.

Chapter 5, Naïve Bayes Classifiers, introduces probabilistic graphical models, and then describes the implementation of the Naïve Bayes and the multivariate Bernoulli classifiers in the context of text mining.

Chapter 6, Regression and Regularization, covers a typical implementation of the linear and least squares regression, the ridge regression as a regularization technique, and finally, the logistic regression.

Chapter 7, Sequential Data Models, introduces the Markov processes followed by a full implementation of the hidden Markov model, and conditional random fields applied to pattern recognition in financial market data.

Chapter 8, Kernel Models and Support Vector Machines, covers the concept of kernel functions with implementation of support vector machine classification and regression, followed by the application of the one-class SVM to anomaly detection.

Chapter 9, Artificial Neural Networks, describes feed-forward neural networks followed by a full implementation of the multilayer perceptron classifier.

Chapter 10, Genetic Algorithms, covers the basics of evolutionary computing and the implementation of the different components of a multipurpose genetic algorithm.

Chapter 11, Reinforcement Learning, introduces the concept of reinforcement learning with an implementation of the Q-learning algorithm followed by a template to build a learning classifier system.

Chapter 12, Scalable Frameworks, covers some of the artifacts and frameworks to create scalable applications for machine learning such as Scala parallel collections, Akka, and the Apache Spark framework.

Appendix A, Basic Concepts, covers the Scala constructs used throughout the book, elements of linear algebra, and an introduction to investment and trading strategies.

Appendix B, References, provides a chapter-wise list of references for [source entry] in the respective chapters. This appendix is available as an online chapter at https://www.packtpub.com/sites/default/files/downloads/87420S_AppendixB_References.pdf.

Short test applications using financial data illustrate the large variety of predictive, regression, and classification models.

The interdependencies between chapters are kept to a minimum. You can easily delve into any chapter once you complete *Chapter 1, Getting Started*, and *Chapter 2, Hello World!*.

What you need for this book

A decent command of the Scala programming language is a prerequisite. Reading through a mathematical formulation, conveniently defined in an information box, is optional. However, some basic knowledge of mathematics and statistics might be helpful to understand the inner workings of some algorithms.

The book uses the following libraries:

- Scala 2.10.3 or higher
- Java JDK 1.7.0_45 or 1.8.0_25
- SBT 0.13 or higher
- JFreeChart 1.0.1
- Apache Commons Math library 3.3 (*Chapter 3, Data Preprocessing*, *Chapter 4, Unsupervised Learning*, and *Chapter 6, Regression and Regularization*)
- Indian Institute of Technology Bombay CRF 0.2 (*Chapter 7, Sequential Data Models*)
- LIBSVM 0.1.6 (*Chapter 8, Kernel Models and Support Vector Machines*)
- Akka 2.2.4 or higher (or Typesafe activator 1.2.10 or higher) (*Chapter 12, Scalable Frameworks*)
- Apache Spark 1.0.2 or higher (*Chapter 12, Scalable Frameworks*)



Understanding the mathematical formulation of a model is optional.

Who this book is for

This book is for software developers with a background in Scala programming who want to learn how to create, validate, and apply machine learning algorithms.

The book is also beneficial to data scientists who want to explore functional programming or improve the scalability of their existing applications using Scala.

This book is designed as a tutorial with comparative hands-on exercises using technical analysis of financial markets.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Finally, the environment variables `JAVA_HOME`, `PATH`, and `CLASSPATH` have to be updated accordingly."

A block of code is set as follows:

```
[default]
val lsp = builder.model(lrJacobian)
                    .weight(wMatrix)
                    .target(labels)
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
[default]
val lsp = builder.model(lrJacobian)
                    .weight(wMatrix)
                    .target(labels)
```

The source code block is described using a reference number embedded as a code comment:


```
[default]
val lsp = builder.model(lrJacobian) //1
                    .weight(wMatrix)
                    .target(labels)
```


The reference number is used in the chapter as follows: "The model instance is initialized with the Jacobian matrix, `lrJacobian` (line 1)."


Any command-line input or output is written as follows:

```
sbt/sbt assembly
```

New terms and **important words** are shown in bold. Words that you see on the screen, for example, in menus or dialog boxes, appear in the text like this: "The loss function is then known as the **hinge loss**."

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

 Mathematical formulas (optional to read) appear in a box like this

For the sake of readability, the elements of the Scala code that are not essential to the understanding of an algorithm such as class, variable, and method qualifiers and validation of arguments, exceptions, or logging are omitted. The convention for code snippets is detailed in the *Format of code snippets* section in *Appendix A, Basic Concepts*.

You will be provided with in-text citation of papers, conference, books, and instructional videos throughout the book. The sources are listed in the *Appendix B, References* using in the following format:

[In-text citation]

For example, in the chapter, you will find an instance as follows:

This time around RSS increases with λ before reaching a maximum for $\lambda > 60$. This behavior is consistent with other findings [6:12].

The respective [source entry] is mentioned in *Appendix B, References*, as follows:

[6:12] *Model selection and assessment* H. Bravo, R. Irizarry, 2010, available at <http://www.cbcb.umd.edu/~hcorrada/PracticalML/pdf/lectures/selection.pdf>.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail feedback@packtpub.com, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at questions@packtpub.com, and we will do our best to address the problem.

1

Getting Started

It is critical for any computer scientist to understand the different classes of machine learning algorithms and be able to select the ones that are relevant to the domain of their expertise and dataset. However, the application of these algorithms represents a small fraction of the overall effort needed to extract an accurate and performing model from input data. A common data mining workflow consists of the following sequential steps:


1. Loading the data.
2. Preprocessing, analyzing, and filtering the input data.
3. Discovering patterns, affinities, clusters, and classes.
4. Selecting the model features and the appropriate machine learning algorithm(s).
5. Refining and validating the model.
6. Improving the computational performance of the implementation.

As we will emphasize throughout this book, each stage of the process is critical to build the *right* model.

This first chapter introduces you to the taxonomy of machine learning algorithms, the tools and frameworks used in the book, and a simple application of logistic regression to get your feet wet.

Mathematical notation for the curious

Each chapter contains a small section dedicated to the formulation of the algorithms for those interested in the mathematical concepts behind the science and art of machine learning. These sections are optional and defined within a tip box. For example, the mathematical expression of the mean and the variance of a variable X mentioned in a tip box will be as follows:



Mean value of a variable $X = \{x\}$ is defined as:

$$E(X) = \frac{1}{n} \sum x_j$$

The variance of a variable $X = \{x\}$ is defined as:

$$Var(X) = \frac{\sum (E(X) - x_j)^2}{n - 1}$$

Why machine learning?

The explosion in the number of digital devices generates an ever-increasing amount of data. The best analogy I can find to describe the need, desire, and urgency to extract knowledge from large datasets is the process of extracting a precious metal from a mine, and in some cases, extracting blood from a stone.

Knowledge is quite often defined as a model that can be constantly updated or tweaked as new data comes into play. Models are obviously domain-specific ranging from credit risk assessment, face recognition, maximization of quality of service, classification of pathological symptoms of disease, optimization of computer networks, and security intrusion detection, to customers' online behavior and purchase history.

Machine learning problems are categorized as classification, prediction, optimization, and regression.

Classification

The purpose of classification is to extract knowledge from historical data. For instance, a classifier can be built to identify a disease from a set of symptoms. The scientist collects information regarding the body temperature (continuous variable), congestion (discrete variables HIGH, MEDIUM, and LOW), and the actual diagnostic (flu). This dataset is used to create a model such as `IF temperature > 102 AND congestion = HIGH THEN patient has the flu (probability 0.72)`, which doctors can use in their diagnostic.

Prediction

Once the model is extracted and validated against the past data, it can be used to draw inference from the future data. A doctor collects symptoms from a patient, such as body temperature and nasal congestion, and anticipates the state of his/her health.

Optimization

Some global optimization problems are intractable using traditional linear and non-linear optimization methods. Machine learning techniques improve the chances that the optimization method converges toward a solution (intelligent search). You can imagine that fighting the spread of a new virus requires optimizing a process that may evolve over time as more symptoms and cases are uncovered.

Regression

Regression is a classification technique that is particularly suitable for a continuous model. Linear (least square), polynomial, and logistic regressions are among the most commonly used techniques to *fit* a parametric model, or function, $y = f(x_j)$, to a dataset. Regression is sometimes regarded as a specialized case of classification for which the output variables are continuous instead of categorical.

Why Scala?

Like most functional languages, Scala provides developers and scientists with a toolbox to implement iterative computations that can be easily woven dynamically into a coherent dataflow. To some extent, Scala can be regarded as an extension of the popular MapReduce model for distributed computation of large amounts of data. Among the capabilities of the language, the following features are deemed essential to machine learning and statistical analysis.

Abstraction

Monoids and monads are important concepts in functional programming.

Monads are derived from the category and group theory allowing developers to create a high-level abstraction as illustrated in **Twitter's Algebird** (<https://github.com/twitter/algebird>) or **Google's Breeze Scala** (<https://github.com/dlwh/breeze>) libraries.

A **monoid** defines a binary operation op on a dataset \mathbb{T} with the property of closure, identity operation, and associativity.

Let's consider the $+$ operation is defined for a set T using the following monoidal representation:

```
trait Monoid[T] {  
  def zero: T  
  def op(a: T, b: T): T  
}
```

Monoids are associative operations. For instance, if `ts1`, `ts2`, and `ts3` are three time series, then the property `ts1 + (ts2 + ts3) = (ts1 + ts2) + ts3` is true. The associativity of a monoid operator is critical in regards to parallelization of computational workflows.

Monads are structures that can be seen either as containers by programmers or as a generalization of Monoids. The collections bundled with the Scala standard library (`list`, `map`, and so on) are constructed as monads [1:1]. Monads provide the ability for those collections to perform the following functions:

1. Create the collection.
2. Transform the elements of the collection.
3. Flatten nested collections.

A common categorical representation of a monad in Scala is a trait, `Monad`, parameterized with a container type `M`:

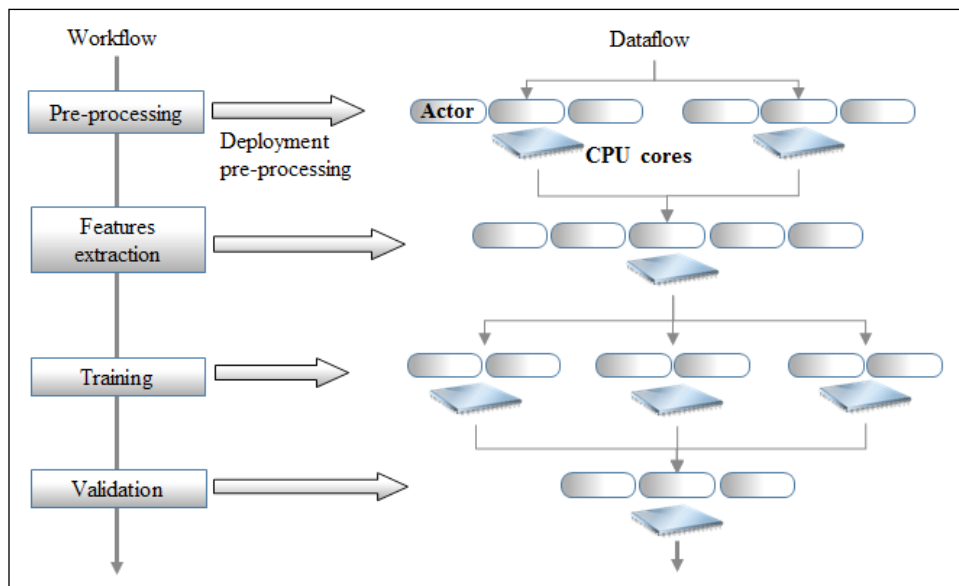
```
trait Monad[M[_]] {  
  def apply[T](a: T): M[T]  
  def flatMap[T, U](m: M[T])(f: T => M[U]): M[U]  
}
```

Monads allow those collections or containers to be chained to generate a workflow. This property is applicable to any scientific computation [1:2].

Scalability

As seen previously, monoids and monads enable parallelization and chaining of data processing functions by leveraging the Scala higher-order methods. In terms of implementation, **Actors** are the core elements that make Scala scalable. Actors act as coroutines, managing the underlying threads pool. Actors communicate through passing asynchronous messages. A distributed computing Scala framework such as Akka and Spark extends the capabilities of the Scala standard library to support computation on very large datasets. Akka and Spark are described in detail in the last chapter of this book [1:3].

In a nutshell, a workflow is implemented as a sequence of activities or computational tasks. Those tasks consist of high-order Scala methods such as `flatMap`, `map`, `fold`, `reduce`, `collect`, `join`, or `filter` applied to a large collection of observations. Scala allows these observations to be partitioned by executing those tasks through a cluster of actors. Scala also supports message dispatching and routing of messages between local and remote actors. The engineers can decide to execute a workflow either locally or distributed across CPU cores and servers with no code or very little code changes.



Deployment of a workflow as a distributed computation

In this diagram, a controller, that is, the master node, manages the sequence of tasks 1 to 4 similar to a scheduler. These tasks are actually executed over multiple worker nodes that are implemented by the Scala actors. The master node exchanges messages with the workers to manage the state of the execution of the workflow as well as its reliability. High availability of these tasks is implemented through a hierarchy of supervising actors.

Configurability

Scala supports **dependency injection** using a combination of abstract variables, self-referenced composition, and stackable traits. One of the most commonly used dependency injection patterns, the **cake pattern**, is used throughout this book to create dynamic computation workflows and plots.

Maintainability

Scala embeds **Domain Specific Languages (DSL)** natively. DSLs are syntactic layers built on top of Scala native libraries. DSLs allow software developers to abstract computation in terms that are easily understood by scientists. The most notorious application of DSLs is the definition of the emulation of the syntax used in the MATLAB program, which data scientists are familiar with.

Computation on demand

Lazy methods and values allow developers to execute functions and allocate computing resources on demand. The Spark framework relies on lazy variables and methods to chain **Resilient Distributed Datasets (RDD)**.

Model categorization

A model can be predictive, descriptive, or adaptive.

Predictive models discover patterns in historical data and extract fundamental trends and relationships between factors. They are used to predict and classify future events or observations. Predictive analytics is used in a variety of fields such as marketing, insurance, and pharmaceuticals. Predictive models are created through supervised learning using a preselected training set.

Descriptive models attempt to find unusual patterns or affinities in data by grouping observations into clusters with similar properties. These models define the first level in knowledge discovery. They are generated through unsupervised learning.

A third category of models, known as **adaptive modeling**, is generated through reinforcement learning. **Reinforcement learning** consists of one or several decision-making agents that recommend and possibly execute actions in the attempt of solving a problem, optimizing an objective function, or resolving constraints.

Taxonomy of machine learning algorithms

The purpose of machine learning is to teach computers to execute tasks without human intervention. An increasing number of applications such as genomics, social networking, advertising, or risk analysis generate a very large amount of data that can be analyzed or mined to extract knowledge or provide insight into a process, a customer, or an organization. Ultimately, machine learning algorithms consist of identifying and validating models to optimize a performance criterion using historical, present, and future data [1:4].

Data mining is the process of extracting or identifying patterns in a dataset.

Unsupervised learning

The goal of **unsupervised learning** is to discover patterns of regularities and irregularities in a set of observations. The process known as density estimation in statistics is broken down into two categories: discovery of data clusters and discovery of latent factors. The methodology consists of processing input data to understand patterns similar to the natural learning process in infants or animals. Unsupervised learning does not require labeled data, and therefore, is easy to implement and execute because no expertise is needed to validate an output. However, it is possible to label the output of a clustering algorithm and use it for future classification.

Clustering

The purpose of data clustering is to partition a collection of data into a number of clusters or data segments. Practically, a clustering algorithm is used to organize observations into clusters by minimizing the observations within a cluster and maximizing the observations between clusters. A clustering algorithm consists of the following steps:

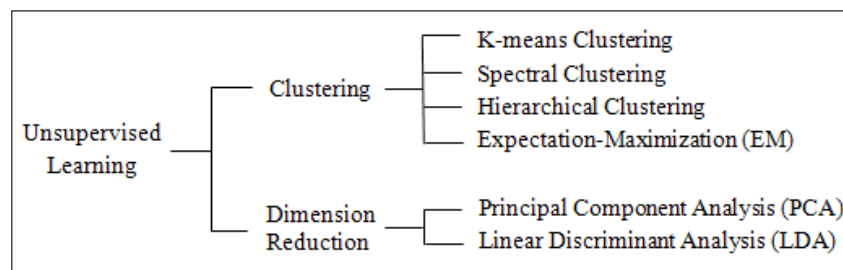
1. Creating a model by making an assumption on the input data.
2. Selecting the objective function or goal of the clustering.
3. Evaluating one or more algorithms to optimize the objective function.

Data clustering is also known as data segmentation or data partitioning.

Dimension reduction

Dimension reduction techniques aim at finding the smallest but most relevant set of features that models dataset reliability. There are many reasons for reducing the number of features or parameters in a model, from avoiding overfitting to reducing computation costs.

There are many ways to classify the different techniques used to extract knowledge from data using unsupervised learning. The following taxonomy breaks down these techniques according to their purpose, although the list is far from being exhaustive, as shown in the following diagram:



Supervised learning

The best analogy for supervised learning is function approximation or curve fitting. In its simplest form, supervised learning attempts to extract a relation or function f $x \rightarrow y$ from a training set $\{x, y\}$. Supervised learning is far more accurate and reliable than any other learning strategy. However, a domain expert may be required to label (tag) data as a training set for certain types of problems.


Supervised machine learning algorithms can be broken into two categories:

- Generative models
- Discriminative models

Generative models

In order to simplify the description of statistics formulas, we adopt the following simplification: the probability of an event X is the same as the probability of the discrete random variable X to have a value x , $p(X) = p(X=x)$. The notation of joint probability (resp. conditional probability) becomes $p(X, Y) = p(X=x, Y=y)$ (resp. $p(X|Y)=p(X=x | Y=y)$).

Generative models attempt to fit a joint probability distribution, $p(X, Y)$, of two events (or random variables), X and Y , representing two sets of observed and hidden (latent) variables x and y . Discriminative models learn the conditional probability $p(Y|X)$ of an event or random variable Y of hidden variables y , given an event or random variable X of observed variables x . Generative models are commonly introduced through the Bayes' rule. The conditional probability of an event Y , given an event X , is computed as the product of the conditional probability of the event X , given the event Y , and the probability of the event X normalized by the probability of event Y [1:5].



Join probability (if X and Y are independent):

$$p(X, Y) = p(X \cap Y) = p(X) \cdot p(Y)$$

Conditional probability:

$$p(Y|X) = P(Y, X)/P(X)$$

The Bayes' rule:

$$P(Y|X) = P(X|Y) \cdot P(X)/P(Y)$$

The Bayes' rule is the foundation of the Naïve Bayes classifier, which is the topic of *Chapter 5, Naïve Bayes Classifiers*.

Discriminative models

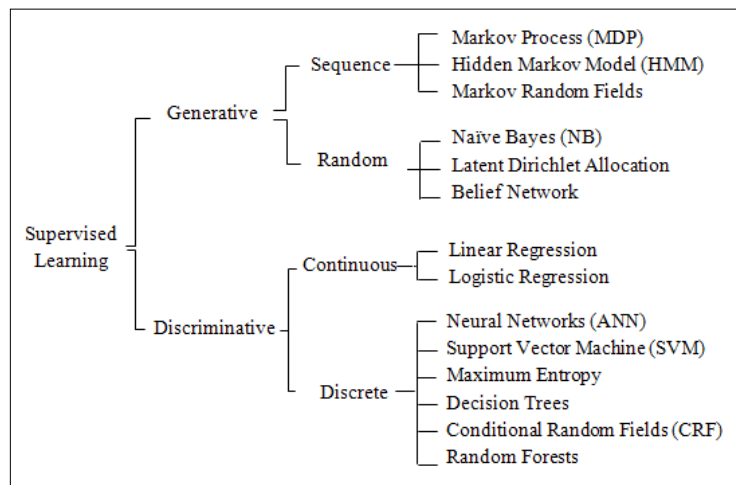
Contrary to generative models, discriminative models compute the conditional probability $p(Y|X)$ directly, using the same algorithm for training and classification.

Generative and discriminative models have their respective advantages and drawbacks. Novice data scientists learn to match the appropriate algorithm to each problem through experimentation. Here is a brief guideline describing which type of models makes sense according to the objective or criteria of the project:

Objective	Generative models	Discriminative models
Accuracy	Highly dependent on the training set.	Probability estimates tend to be more accurate.
Modeling requirements	There is a need to model both observed and hidden variables, which requires a significant amount of training.	The quality of the training set does not have to be as rigorous as for generative models.

Objective	Generative models	Discriminative models
Computation cost	This is usually low. For example, any graphical method derived from the Bayes' rule has low overhead.	Most algorithms rely on optimization of a convex that introduces significant performance overhead.
Constraints	These models assume some degree of independence among the model features.	Most discriminative algorithms accommodate dependencies between features.

We can further refine the taxonomy of supervised learning algorithms by segregating between sequential and random variables for generative models and breaking down discriminative methods as applied to continuous processes (regression) and discrete processes (classification):

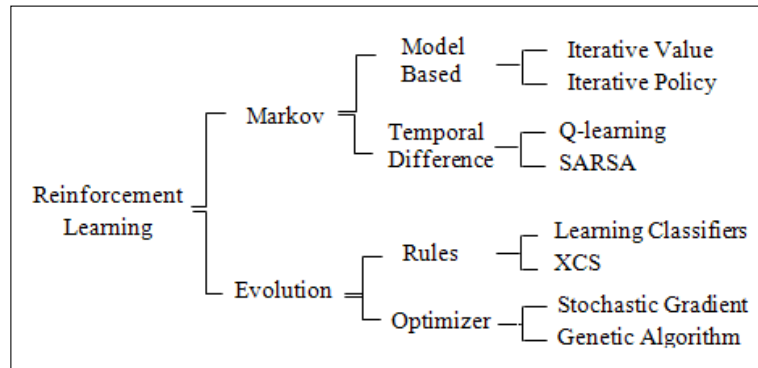


Reinforcement learning

Reinforcement learning is not as well understood as supervised and unsupervised learning outside the realms of robotics or game strategy. However, since the 90s, genetic-algorithms-based classifiers have become increasingly popular to solve problems that require collaboration with a domain expert. For some types of applications, reinforcement learning algorithms output a set of recommended actions for the *adaptive* system to execute. In its simplest form, these algorithms compute or estimate the best course of action. Most complex systems based on reinforcement learning establish and update policies that can be vetoed by an expert. The foremost challenge developers of reinforcement learning systems face is that the recommended action or policy may depend on partially observable states and how to deal with uncertainty.

Genetic algorithms are not usually considered part of the reinforcement learning toolbox. However, advanced models such as learning classifier systems use genetic algorithms to classify and reward the rules and policies.

As with the two previous learning strategies, reinforcement learning models can be categorized as Markovian or evolutionary:



This is a brief overview of machine learning algorithms with a suggested taxonomy. There are almost as many ways to introduce machine learning as there are data and computer scientists. We encourage you to browse through the list of references at the end of the book and find the documentation appropriate to your level of interest and understanding.

Tools and frameworks

Before getting your hands dirty, you need to download and deploy a minimum set of tools and libraries so as not to reinvent the wheel. A few key components have to be installed in order to compile and run the source code described throughout the book. We focus on open source and commonly available libraries, although you are invited to experiment with equivalent tools of your choice. The learning curve for the frameworks described here is minimal.

Java

The code described in the book has been tested with **JDK 1.7.0_45** and **JDK 1.8.0_25** on **Windows x64** and **MacOS X x64**. You need to install the Java Development Kit if you have not already done so. Finally, the environment variables `JAVA_HOME`, `PATH`, and `CLASSPATH` have to be updated accordingly.

Scala

The code has been tested with Scala 2.10.4. We recommend using Scala version 2.10.3 or higher and SBT 0.13 or higher. Let's assume that Scala runtime (REPL) and libraries have been properly installed and environment variables `SCALA_HOME` and `PATH` have been updated. The description and installation instructions of the Scala plugin for Eclipse are available at <http://scala-ide.org/docs/user/gettingstarted.html>.

You can also download the Scala plugin for IntelliJ IDEA from the JetBrains website at <http://confluence.jetbrains.com/display/SCA/>.

The ubiquitous **simple build tool** (**sbt**) will be our primary building engine. The syntax of the build file `sbt/build.sbt` conforms to version 0.13, and is used to compile and assemble the source code presented throughout this book.

Apache Commons Math

Apache Commons Math is a Java library for numerical processing, algebra, statistics, and optimization [1:6].

Description

This is a lightweight library that provides developers with a foundation of small, ready-to-use Java classes that can be easily weaved into a machine learning problem. The examples used throughout the book require version 3.3 or higher.

The main components of Apache Commons Math are:

- Functions, differentiation, and integral and ordinary differential equations
- Statistics distribution
- Linear and nonlinear optimization
- Dense and Sparse vectors and matrices
- Curve fitting, correlation, and regression

For more information, visit <http://commons.apache.org/proper/commons-math>.

Licensing

We need Apache Public License 2.0; the terms are available at <http://www.apache.org/licenses/LICENSE-2.0>.

Installation

The installation and deployment of the Commons Math library are quite simple:

1. Go to the download page, http://commons.apache.org/proper/commons-math/download_math.cgi.
2. Download the latest .jar files in the **Binaries** section, commons-math3-3.3-bin.zip (for version 3.3, for instance).
3. Unzip and install the .jar files.
4. Add commons-math3-3.3.jar to classpath as follows:
 - For Mac OS X, use the command `export CLASSPATH=$CLASSPATH:/Commons_Math_path/commons-math3-3.3.jar`
 - For Windows, navigate to **System property | Advanced system settings | Advanced | Environment variables...**, then edit the entry of the CLASSPATH variable
5. Add the commons-math3-3.3.jar file to your IDE environment if needed (that is, for Eclipse, navigate to **Project | Properties | Java Build Path | Libraries | Add External JARs**).

You can also download commons-math3-3.3-src.zip from the **Source** section.

JFreeChart

JFreeChart is an open source chart and plotting Java library, widely used in the Java programmer community. It was originally created by David Gilbert [1:7].

Description

The library supports a variety of configurable plots and charts (scatter, dial, pie, area, bar, box and whisker, stacked, and 3D). We use JFreeChart to display the output of data processing and algorithms throughout the book, but you are encouraged to explore this great library on your own, as time permits.

Licensing

It is distributed under the terms of the **GNU Lesser General Public License (LGPL)**, which permits its use in proprietary applications.

Installation

To install and deploy JFreeChart, perform the following steps:

1. Visit <http://www.jfree.org/jfreechart>.
2. Download the latest version from Source Forge at <http://sourceforge.net/projects/jfreechart/files>.
3. Unzip and install the .jar file.
4. Add `jfreechart-1.0.17.jar` (for version 1.0.17) to classpath as follows:
 - For Mac OS, update the classpath by using export
`CLASSPATH=$CLASSPATH:/JFreeChart_path/ jfreechart-1.0.17.jar`
 - For Windows, go to **System property | Advanced system settings | Advanced | Environment variables...** and then edit the entry of the `CLASSPATH` variable
5. Add the `jfreechart-1.0.17.jar` file to your IDE environment, if needed.

Other libraries and frameworks

Libraries and tools that are specific to a single chapter are introduced along with the topic. Scalable frameworks are presented in the last chapter along with the instructions to download them. Libraries related to the conditional random fields and support vector machines are described in the respective chapters.



Why not use Scala algebra and numerical libraries

Libraries such as **Breeze**, **ScalaNLP**, and **Algebird** are great Scala frameworks for linear algebra, numerical analysis, and machine learning. They provide even the most seasoned Scala programmer with a high-quality layer of abstraction. However, this book is designed as a tutorial that allows developers to write algorithms from the ground up using simple common Java libraries [1:8].

Source code

The Scala programming language is used to implement and evaluate the machine learning techniques presented in this book. Only a subset of the source code used to implement the techniques are presented in the book. The formal implementation of these algorithms is available on the website of Packt Publishing (<http://www.packtpub.com>).



Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Context versus view bounds

Most Scala classes discussed in the book are parameterized with the type associated to the discrete/categorical value (`Int`) or continuous value (`Double`). Context bounds would require that any type used by the client code has `Int` or `Double` as upper bounds:

```
class MyClassInt[T <: Int]
class MyClassFloat[T <: Double]
```

Such a design introduces constraints on the client to inherit from simple types and to deal with covariance and contravariance for container types [1:9].

For this book, view bounds are used instead of context bounds only where they require an implicit conversion to the parameterized type to be defined:

```
Class MyClassFloat[T <% Double]
implicit def T2Double(t : T): Double
```

Presentation

For the sake of readability of the implementation of algorithms, all nonessential code such as error checking, comments, exceptions, or imports are omitted. The following code elements are discarded in the code snippet presented in the book:

- Code comments
- Validation of class parameters and method arguments:


```
class BaumWelchEM(val lambda: HMMLambda ...) {
  require( lambda != null, "Lambda model is undefined")
```
- Exceptions and an exception handler:


```
try { .. }
catch {
  case e: ArrayIndexOutOfBoundsException =>println(e.
toString)
}
```


- Nonessential annotation:
`@inline def mean = ..`
- Logging and debugging code:
`m_logger.debug(...)`
- Private and nonessential methods

Primitives and implicits

The algorithms presented in this book share the same primitive types, generic operators, and implicit conversions.

Primitive types

For the sake of readability of the code, the following primitive types will be used:

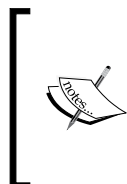
```
type XY = (Double, Double)
type XYTSeries = Array[(Double, Double)]
type DMatrix[T] = Array[Array[T]]
type DVector[T] = Array[T]
type DblMatrix = DMatrix[Double]
type DblVector = Array[Double]
```

The types have the behavior (methods) of their primitive counterpart (array). However, adding a new functionality to vectors, matrices, and time series requires classes of their own right. These classes will be introduced in the next chapter.

Type conversions

Implicit conversion is an important feature of the Scala programming language because it allows developers to specify a type conversion for an entire library in a single place. Here are a few of the implicit type conversions used throughout the book:

```
implicit def int2Double(n: Int): Double = n.toDouble
implicit def vectorT2DblVector[T <% Double] (vt: DVector[T]): DblVector
= vt.map( t => t.toDouble)
implicit def double2DblVector(x: Double): DblVector = Array[Double] (x)
implicit def dblPair2DblVector(x: (Double, Double)): DblVector =
Array[Double] (x._1,x._2)
implicit def dblPairs2DblRows(x: (Double, Double)): DblMatrix =
Array[Array[Double]] (Array[Double] (x._1, x._2))
...
```



Library-specific conversion

The conversion between the primitive type listed here and types introduced in a particular library (such as Apache Commons Math) is declared in future chapters the first time those libraries are used.

Operators

Lastly, some operations are applied by multiple machine learning or preprocessing algorithms. They need to be defined implicitly. The operation on a pair of a vector of arbitrary type and vector of `Double` is defined as follows:

```
def Op[T <% Double](v: DVector[T], w: DblVector, op: (T, Double) =>
Double): DblVector =
  v.zipWithIndex.map(x => op(x._1, w(x._2)))
```

It is also convenient to define the following operators that are included in the Scala standard library:

```
implicit def /(v: DblVector, n: Int):DblVector = v.map( x => x/n)
implicit def /(m: DblMatrix, col: Int, z: Double): DblMatrix = { (0
until m(n).size).foreach(i => m(n)(i) /= z) }
```

We won't have to redefine the types, conversions, and operators from now on.

Immutability

It is usually a good idea to reduce the number of states of an object. Method invocation transitions an object from one state to another. The larger the number of methods or states, the more cumbersome the testing process becomes.

There is no point in creating a model that is not defined (trained). Therefore, making the training of a model as part of the constructor of the class it implements makes a lot of sense. Therefore, the only public methods of a machine learning algorithm are:

- Classification or prediction
- Validation
- Retrieval of model parameters (weights, latent variables, hidden states, and so on), if needed

Performance of Scala iterators

The evaluation of the performance of Scala high-order iterative methods is beyond the scope of this book. However, it is important to be aware of the trade-off of each method.

The `for` loop construct is to be avoided as a counting iterator except if it is used in conjunction with `yield`. It is designed to implement the for-comprehension monad (`map-flatMap`). The source code presented in this book uses the `while` and `foreach` constructs.

Scala reducer methods `reduce` and `fold` are also frequently used for their efficiency.

Let's kick the tires

This final section introduces the key elements of the training and classification workflow. A test case using a simple logistic regression is used to illustrate each step of the computational workflow.

Overview of computational workflows

In its simplest form, a computational workflow to perform runtime processing of a dataset is composed of the following stages:

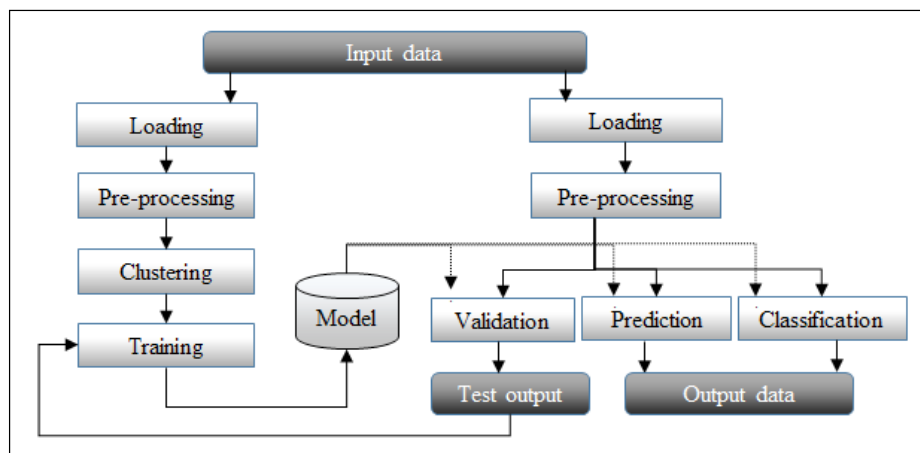
1. Loading the dataset from files, databases, or any streaming devices.
2. Splitting the dataset for parallel data processing.
3. Preprocessing data using filtering techniques, analysis of variance, and applying penalty and normalization functions whenever necessary.
4. Applying the model, either a set of clusters or classes to classify new data.
5. Assessing the quality of the model.

A similar sequence of tasks is used to extract a model from a training dataset:

1. Loading the dataset from files, databases, or any streaming devices.
2. Splitting the dataset for parallel data processing.
3. Applying filtering techniques, analysis of variance, and penalty and normalization functions to the raw dataset whenever necessary.
4. Selecting the training, testing, and validation set from the cleansed input data.
5. Extracting key features, establishing affinity between a similar group of observations using clustering techniques or supervised learning algorithms.

6. Reducing the number of features to a manageable set of attributes to avoid overfitting the training set.
7. Validating the model and tuning the model by iterating steps 5, 6, and 7 until the error meets criteria.
8. Storing the model into the file or database to be loaded for runtime processing of new observations.

Data clustering and data classification can be performed independent of each other or as part of a workflow that uses clustering techniques as a preprocessing stage of the training phase of a supervised learning algorithm. Data clustering does not require a model to be extracted from a training set, while classification can be performed only if a model has been built from the training set. The following image gives an overview of training and classification:



A generic data flow for training and running a model

This diagram is an overview of a typical data mining processing pipeline. The first phase consists of extracting the model through clustering or training of a supervised learning algorithm. The model is then validated against test data, for which the source is the same as the training set but with different observations. Once the model is created and validated, it can be used to classify real-time data or predict future behavior. In reality, real-world workflows are more complex and require being dynamically configurable to allow experimentation of different models. Several alternative classifiers can be used to perform a regression and different filtering algorithms are applied against input data depending of the latent noise in the raw data.