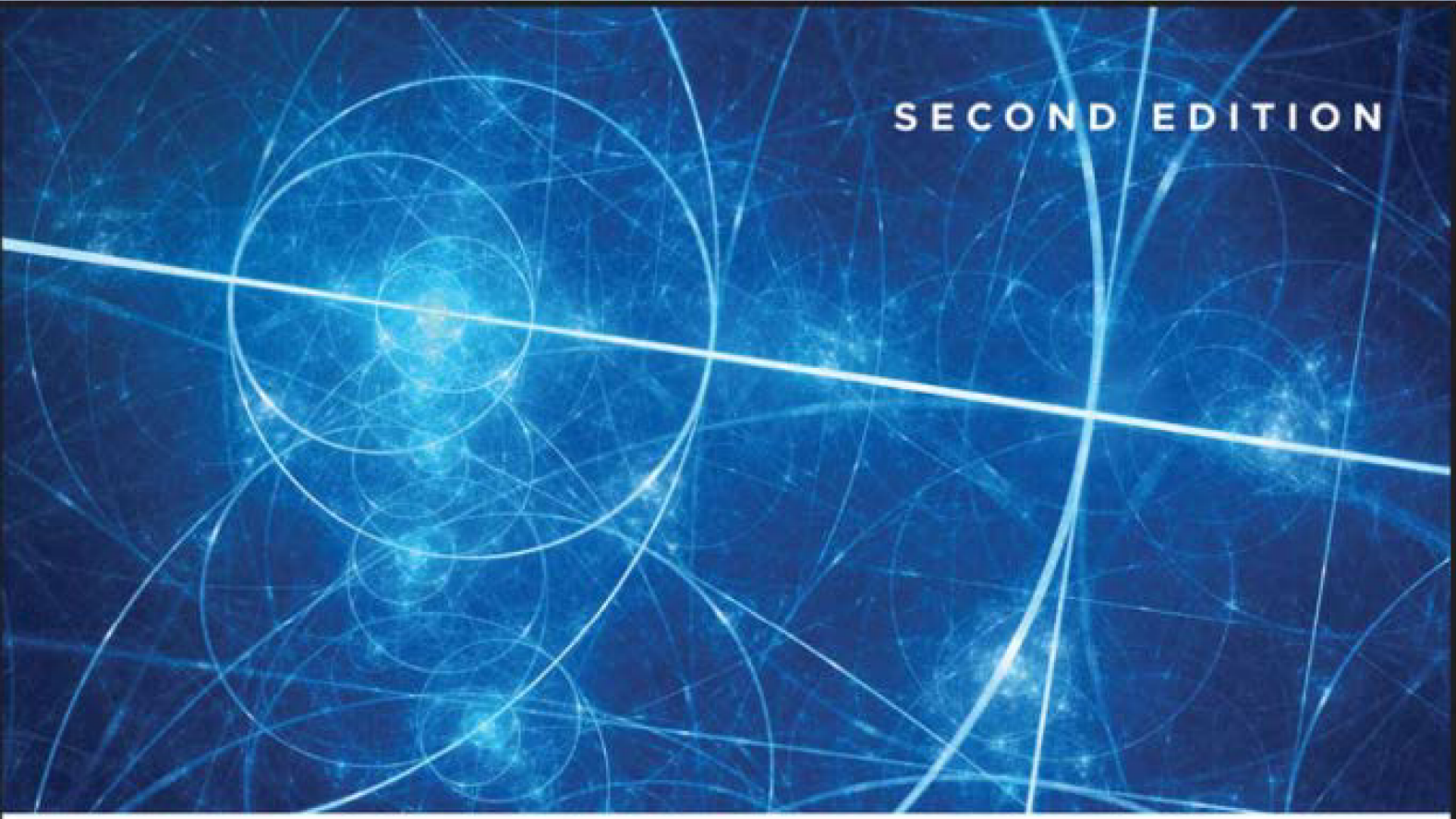


MACHINE LEARNING

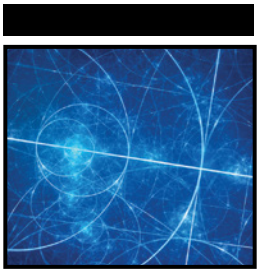
WITH SPARK™ AND PYTHON®

ESSENTIAL TECHNIQUES FOR PREDICTIVE ANALYTICS

MICHAEL BOWLES

WILEY

Machine Learning with Spark™ and Python®



Machine Learning with Spark™ and Python®

Essential Techniques for Predictive Analytics

Second Edition

Michael Bowles

WILEY

Machine Learning with Spark™ and Python®: Essential Techniques for Predictive Analytics, Second Edition

Published by
John Wiley & Sons, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

ISBN: 978-1-119-56193-4
ISBN: 978-1-119-56201-6 (ebk)
ISBN: 978-1-119-56195-8 (ebk)

Manufactured in the United States of America

Library of Congress Control Number: 2019940771

I dedicate this book to my expanding family of children and grandchildren, Scott, Seth, Cayley, Rees, and Lia. Being included in their lives is a constant source of joy for me. I hope it makes them smile to see their names in print. I also dedicate it to my close friend Dave, whose friendship remains steadfast in spite of my best efforts. I hope this makes him smile too.

Mike Bowles, Silicon Valley 2019

About the Author

Dr. Michael Bowles (Mike) holds bachelor's and master's degrees in mechanical engineering, an ScD in instrumentation, and an MBA. He has worked in academia, technology, and business. Mike currently works with companies where artificial intelligence or machine learning are integral to success. He serves variously as part of the management team, a consultant, or advisor. He also teaches machine learning courses at UC Berkeley and Hacker Dojo, a co-working space and startup incubator in Mountain View, CA.

Mike was born in Oklahoma and took his bachelor's and master's degrees there, then after a stint in Southeast Asia went to Cambridge for ScD and C. Stark Draper Chair at MIT after graduation. Mike left Boston to work on communications satellites at Hughes Aircraft Company in Southern California, and then after completing an MBA at UCLA moved to the San Francisco Bay Area to take roles as founder and CEO of two successful venture-backed startups.

Mike remains actively involved in technical and startup-related work. Recent projects include the use of machine learning in industrial inspection and automation, financial prediction, predicting biological outcomes on the basis of molecular graph structures, and financial risk estimation. He has participated in due diligence work on companies in the artificial intelligence and machine learning arenas. Mike can be reached through `mbowles.com`.

About the Technical Editor

James York-Winegar is an Infrastructure Principal with Accenture Enkitec Group. James helps companies of all sizes from startups to enterprises with their data lifecycle by helping them bridge the gap between systems management and data science. He started his career in physics, where he did large-scale quantum chemistry simulations on supercomputers, and went into technology. He holds a master's in Data Science from Berkeley.

Acknowledgments

I'd like to acknowledge the splendid support that people at Wiley have offered during the course of writing this book and making the revisions for this second edition. It began with Robert Elliot, the acquisitions editor who first contacted me about writing a book—very easy to work with. Tom Dinse has done a splendid job editing this second edition. He's been responsive, thorough, flexible, and completely professional, as I've come to expect from Wiley. I thank you.

I'd also like to acknowledge the enormous comfort that comes from having such a quick, capable computer scientist as James Winegar doing the technical editing on the book. James has brought a more consistent style and has made a number of improvements that will make the code that comes along with the book easier to use and understand. Thank you for that.

The example problems used in the book come from the University of California at Irvine's data repository. UCI does the machine learning community a great service by gathering these data sets, curating them, and making them freely available. The reference for this material is:

Bache, K. & Lichman, M. (2013). UCI Machine Learning Repository (`http://archive.ics.uci.edu/ml`). Irvine, CA: University of California, School of Information and Computer Science.

Contents at a Glance

Introduction		xxi
Chapter 1	The Two Essential Algorithms for Making Predictions	1
Chapter 2	Understand the Problem by Understanding the Data	23
Chapter 3	Predictive Model Building: Balancing Performance, Complexity, and Big Data	77
Chapter 4	Penalized Linear Regression	129
Chapter 5	Building Predictive Models Using Penalized Linear Methods	169
Chapter 6	Ensemble Methods	221
Chapter 7	Building Ensemble Models with Python	265
Index		329

Contents

Introduction **xxi**

Chapter 1 The Two Essential Algorithms for Making Predictions **1**
Why Are These Two Algorithms So Useful? 2
What Are Penalized Regression Methods? 7
What Are Ensemble Methods? 9
How to Decide Which Algorithm to Use 11
The Process Steps for Building a Predictive Model 13
Framing a Machine Learning Problem 15
Feature Extraction and Feature Engineering 17
Determining Performance of a Trained Model 18
Chapter Contents and Dependencies 18
Summary 20

Chapter 2 Understand the Problem by Understanding the Data **23**
The Anatomy of a New Problem 24
Different Types of Attributes and Labels Drive Modeling Choices 26
Things to Notice about Your New Data Set 27
Classification Problems: Detecting Unexploded Mines Using Sonar 28
Physical Characteristics of the Rocks Versus Mines Data Set 29
Statistical Summaries of the Rocks Versus Mines Data Set 32
Visualization of Outliers Using a Quantile-Quantile Plot 34
Statistical Characterization of Categorical Attributes 35
How to Use Python Pandas to Summarize the Rocks Versus Mines Data Set 36
Visualizing Properties of the Rocks Versus Mines Data Set 39
Visualizing with Parallel Coordinates Plots 39
Visualizing Interrelationships between Attributes and Labels 41

Visualizing Attribute and Label Correlations Using a Heat Map 48
Summarizing the Process for Understanding the Rocks Versus Mines Data Set 50
Real-Valued Predictions with Factor Variables: How Old Is Your Abalone? 50
Parallel Coordinates for Regression Problems—Visualize Variable Relationships for the Abalone Problem 55
How to Use a Correlation Heat Map for Regression—Visualize Pair-Wise Correlations for the Abalone Problem 59
Real-Valued Predictions Using Real-Valued Attributes: Calculate How Your Wine Tastes 61
Multiclass Classification Problem: What Type of Glass Is That? 67
Using PySpark to Understand Large Data Sets 72
Summary 75

Chapter 3 Predictive Model Building: Balancing Performance, Complexity, and Big Data 77
The Basic Problem: Understanding Function Approximation 78
Working with Training Data 79
Assessing Performance of Predictive Models 81
Factors Driving Algorithm Choices and Performance—Complexity and Data 82
Contrast between a Simple Problem and a Complex Problem 82
Contrast between a Simple Model and a Complex Model 85
Factors Driving Predictive Algorithm Performance 89
Choosing an Algorithm: Linear or Nonlinear? 90
Measuring the Performance of Predictive Models 91
Performance Measures for Different Types of Problems 91
Simulating Performance of Deployed Models 105
Achieving Harmony between Model and Data 107
Choosing a Model to Balance Problem Complexity, Model Complexity, and Data Set Size 107
Using Forward Stepwise Regression to Control Overfitting 109
Evaluating and Understanding Your Predictive Model 114
Control Overfitting by Penalizing Regression Coefficients—Ridge Regression 116
Using PySpark for Training Penalized Regression Models on Extremely Large Data Sets 124
Summary 127

Chapter 4 Penalized Linear Regression 129
Why Penalized Linear Regression Methods Are So Useful 130
Extremely Fast Coefficient Estimation 130
Variable Importance Information 131
Extremely Fast Evaluation When Deployed 131
Reliable Performance 131
Sparse Solutions 132

Problem May Require Linear Model 132
When to Use Ensemble Methods 132
Penalized Linear Regression: Regulating Linear Regression for Optimum Performance 132
Training Linear Models: Minimizing Errors and More 135
Adding a Coefficient Penalty to the OLS Formulation 136
Other Useful Coefficient Penalties—Manhattan and ElasticNet 137
Why Lasso Penalty Leads to Sparse Coefficient Vectors 138
ElasticNet Penalty Includes Both Lasso and Ridge 140
Solving the Penalized Linear Regression Problem 141
Understanding Least Angle Regression and Its Relationship to Forward Stepwise Regression 141
How LARS Generates Hundreds of Models of Varying Complexity 145
Choosing the Best Model from the Hundreds LARS Generates 147
Using Glmnet: Very Fast and Very General 152
Comparison of the Mechanics of Glmnet and LARS Algorithms 153
Initializing and Iterating the Glmnet Algorithm 153
Extension of Linear Regression to Classification Problems 157
Solving Classification Problems with Penalized Regression 157
Working with Classification Problems Having More Than Two Outcomes 161
Understanding Basis Expansion: Using Linear Methods on Nonlinear Problems 161
Incorporating Non-Numeric Attributes into Linear Methods 163
Summary 166

Chapter 5 Building Predictive Models Using Penalized Linear Methods 169
Python Packages for Penalized Linear Regression 170
Multivariable Regression: Predicting Wine Taste 171
Building and Testing a Model to Predict Wine Taste 172
Training on the Whole Data Set before Deployment 175
Basis Expansion: Improving Performance by Creating New Variables from Old Ones 179
Binary Classification: Using Penalized Linear Regression to Detect Unexploded Mines 182
Build a Rocks Versus Mines Classifier for Deployment 191
Multiclass Classification: Classifying Crime Scene Glass Samples 200
Linear Regression and Classification Using PySpark 203
Using PySpark to Predict Wine Taste 204
Logistic Regression with PySpark: Rocks Versus Mines 208
Incorporating Categorical Variables in a PySpark Model: Predicting Abalone Rings 213

Multiclass Logistic Regression with Meta Parameter Optimization 217
Summary 219

Chapter 6 Ensemble Methods 221
Binary Decision Trees 222
How a Binary Decision Tree Generates Predictions 224
How to Train a Binary Decision Tree 225
Tree Training Equals Split Point Selection 227
How Split Point Selection Affects Predictions 228
Algorithm for Selecting Split Points 229
Multivariable Tree Training—Which Attribute to Split? 229
Recursive Splitting for More Tree Depth 230
Overfitting Binary Trees 231
Measuring Overfit with Binary Trees 231
Balancing Binary Tree Complexity for Best Performance 232
Modifications for Classification and Categorical Features 235
Bootstrap Aggregation: "Bagging" 235
How Does the Bagging Algorithm Work? 236
Bagging Performance—Bias Versus Variance 239
How Bagging Behaves on Multivariable Problem 241
Bagging Needs Tree Depth for Performance 245
Summary of Bagging 246
Gradient Boosting 246
Basic Principle of Gradient Boosting Algorithm 246
Parameter Settings for Gradient Boosting 249
How Gradient Boosting Iterates toward a Predictive Model 249
Getting the Best Performance from Gradient Boosting 250
Gradient Boosting on a Multivariable Problem 253
Summary for Gradient Boosting 256
Random Forests 256
Random Forests: Bagging Plus Random Attribute Subsets 259
Random Forests Performance Drivers 260
Random Forests Summary 261
Summary 262

Chapter 7 Building Ensemble Models with Python 265
Solving Regression Problems with Python Ensemble Packages 265
Using Gradient Boosting to Predict Wine Taste 266
Using the Class Constructor for GradientBoostingRegressor 266
Using GradientBoostingRegressor to Implement a Regression Model 268
Assessing the Performance of a Gradient Boosting Model 271
Building a Random Forest Model to Predict Wine Taste 272
Constructing a RandomForestRegressor Object 273

Modeling Wine Taste with RandomForestRegressor 275
Visualizing the Performance of a Random Forest Regression Model 279
Incorporating Non-Numeric Attributes in Python Ensemble Models 279
Coding the Sex of Abalone for Gradient Boosting Regression in Python 280
Assessing Performance and the Importance of Coded Variables with Gradient Boosting 282
Coding the Sex of Abalone for Input to Random Forest Regression in Python 284
Assessing Performance and the Importance of Coded Variables 287
Solving Binary Classification Problems with Python Ensemble Methods 288
Detecting Unexploded Mines with Python Gradient Boosting 288
Determining the Performance of a Gradient Boosting Classifier 291
Detecting Unexploded Mines with Python Random Forest 292
Constructing a Random Forest Model to Detect Unexploded Mines 294
Determining the Performance of a Random Forest Classifier 298
Solving Multiclass Classification Problems with Python Ensemble Methods 300
Dealing with Class Imbalances 301
Classifying Glass Using Gradient Boosting 301
Determining the Performance of the Gradient Boosting Model on Glass Classification 306
Classifying Glass with Random Forests 307
Determining the Performance of the Random Forest Model on Glass Classification 310
Solving Regression Problems with PySpark Ensemble Packages 311
Predicting Wine Taste with PySpark Ensemble Methods 312
Predicting Abalone Age with PySpark Ensemble Methods 317
Distinguishing Mines from Rocks with PySpark Ensemble Methods 321
Identifying Glass Types with PySpark Ensemble Methods 325
Summary 327

Index **329**

Introduction

Extracting actionable information from data is changing the fabric of modern business in ways that directly affect programmers. One way is the demand for new programming skills. Market analysts predict demand for people with advanced statistics and machine learning skills will exceed supply by 140,000 to 190,000 by 2018. That means good salaries and a wide choice of interesting projects for those who have the requisite skills. Another development that affects programmers is progress in developing core tools for statistics and machine learning. This relieves programmers of the need to program intricate algorithms for themselves each time they want to try a new one. Among general-purpose programming languages, Python developers have been in the forefront, building state-of-the-art machine learning tools, but there is a gap between having the tools and being able to use them efficiently.

Programmers can gain general knowledge about machine learning in a number of ways: online courses, a number of well-written books, and so on. Many of these give excellent surveys of machine learning algorithms and examples of their use, but because of the availability of so many different algorithms, it's difficult to cover the details of their usage in a survey.

This leaves a gap for the practitioner. The number of algorithms available requires making choices that a programmer new to machine learning might not be equipped to make until trying several, and it leaves the programmer to fill in the details of the usage of these algorithms in the context of overall problem formulation and solution.

This book attempts to close that gap. The approach taken is to restrict the algorithms covered to two families of algorithms that have proven to give optimum performance for a wide variety of problems. This assertion is supported by their dominant usage in machine learning competitions, their early inclusion in

newly developed packages of machine learning tools, and their performance in comparative studies (as discussed in Chapter 1, "The Two Essential Algorithms for Making Predictions"). Restricting attention to two algorithm families makes it possible to provide good coverage of the principles of operation and to run through the details of a number of examples showing how these algorithms apply to problems with different structures.

The book largely relies on code examples to illustrate the principles of operation for the algorithms discussed. I've discovered in the classes I have taught at University of California, Berkeley, Galvanize, University of New Haven, and Hacker Dojo, that programmers generally grasp principles more readily by seeing simple code illustrations than by looking at math.

This book focuses on Python because it offers a good blend of functionality and specialized packages containing machine learning algorithms. Python is an often-used language that is well known for producing compact, readable code. That fact has led a number of leading companies to adopt Python for prototyping and deployment. Python developers are supported by a large community of fellow developers, development tools, extensions, and so forth. Python is widely used in industrial applications and in scientific programming, as well. It has a number of packages that support computationally intensive applications like machine learning, and it is a good collection of the leading machine learning algorithms (so you don't have to code them yourself). Python is a better general-purpose programming language than specialized statistical languages such as R or SAS (Statistical Analysis System). Its collection of machine learning algorithms incorporates a number of top-flight algorithms and continues to expand.

Who This Book Is For

This book is intended for Python programmers who want to add machine learning to their repertoire, either for a specific project or as part of keeping their toolkit relevant. Perhaps a new problem has come up at work that requires machine learning. With machine learning being covered so much in the news these days, it's a useful skill to claim on a resume.

This book provides the following for Python programmers:

- A description of the basic problems that machine learning attacks
- Several state-of-the-art algorithms
- The principles of operation for these algorithms
- Process steps for specifying, designing, and qualifying a machine learning system

- Examples of the processes and algorithms
- Hackable code

To get through this book easily, your primary background requirements include an understanding of programming or computer science and the ability to read and write code. The code examples, libraries, and packages are all Python, so the book will prove most useful to Python programmers. In some cases, the book runs through code for the core of an algorithm to demonstrate the operating principles, but then uses a Python package incorporating the algorithm to apply the algorithm to problems. Seeing code often gives programmers an intuitive grasp of an algorithm in the way that seeing the math does for others. Once the understanding is in place, examples will use developed Python packages with the bells and whistles that are important for efficient use (error checking, handling input and output, developed data structures for the models, defined predictor methods incorporating the trained model, and so on).

In addition to having a programming background, some knowledge of math and statistics will help get you through the material easily. Math requirements include some undergraduate-level differential calculus (knowing how to take a derivative and a little bit of linear algebra), matrix notation, matrix multiplication, and matrix inverse. The main use of these will be to follow the derivations of some of the algorithms covered. Many times, that will be as simple as taking a derivative of a simple function or doing some basic matrix manipulations. Being able to follow the calculations at a conceptual level may aid your understanding of the algorithm. Understanding the steps in the derivation can help you to understand the strengths and weaknesses of an algorithm and can help you to decide which algorithm is likely to be the best choice for a particular problem.

This book also uses some general probability and statistics. The requirements for these include some familiarity with undergraduate-level probability and concepts such as the mean value of a list of real numbers, variance, and correlation. You can always look through the code if some of the concepts are rusty for you.

This book covers two broad classes of machine learning algorithms: penalized linear regression (for example, Ridge and Lasso) and ensemble methods (for example, Random Forest and Gradient Boosting). Each of these families contains variants that will solve regression and classification problems. (You learn the distinction between classification and regression early in the book.)

Readers who are already familiar with machine learning and are only interested in picking up one or the other of these can skip to the two chapters covering that family. Each method gets two chapters—one covering principles of operation and the other running through usage on different types of problems. Penalized linear regression is covered in Chapter 4, "Penalized Linear Regression," and Chapter 5, "Building Predictive Models Using Penalized Linear

Methods." Ensemble methods are covered in Chapter 6, "Ensemble Methods," and Chapter 7, "Building Ensemble Models with Python." To familiarize yourself with the problems addressed in the chapters on usage of the algorithms, you might find it helpful to skim Chapter 2, "Understand the Problem by Understanding the Data," which deals with data exploration. Readers who are just starting out with machine learning and want to go through from start to finish might want to save Chapter 2 until they start looking at the solutions to problems in later chapters.

What This Book Covers

As mentioned earlier, this book covers two algorithm families that are relatively recent developments and that are still being actively researched. They both depend on, and have somewhat eclipsed, earlier technologies.

Penalized linear regression represents a relatively recent development in ongoing research to improve on ordinary least squares regression. Penalized linear regression has several features that make it a top choice for predictive analytics. Penalized linear regression introduces a tunable parameter that makes it possible to balance the resulting model between overfitting and underfitting. It also yields information on the relative importance of the various inputs to the predictions it makes. Both of these features are vitally important to the process of developing predictive models. In addition, penalized linear regression yields the best prediction performance in some classes of problems, particularly underdetermined problems and problems with very many input parameters such as genetics and text mining. Furthermore, there's been a great deal of recent development of coordinate descent methods, making training penalized linear regression models extremely fast.

To help you understand penalized linear regression, this book recapitulates ordinary linear regression and other extensions to it, such as stepwise regression. The hope is that these will help cultivate intuition.

Ensemble methods are one of the most powerful predictive analytics tools available. They can model extremely complicated behavior, especially for problems that are vastly overdetermined, as is often the case for many web-based prediction problems (such as returning search results or predicting ad click-through rates). Many seasoned data scientists use ensemble methods as their first try because of their performance. They are relatively simple to use, and they also rank variables in terms of predictive performance.

Ensemble methods have followed a development path parallel to penalized linear regression. Whereas penalized linear regression evolved from overcoming the limitations of ordinary regression, ensemble methods evolved to overcome the limitations of binary decision trees. Correspondingly, this book's

coverage of ensemble methods covers some background on binary decision trees because ensemble methods inherit some of their properties from binary decision trees. Understanding them helps cultivate intuition about ensemble methods.

What Has Changed Since the First Edition

In the three years since the first edition was published, Python has more firmly established itself as the primary language for data science. Developers of platforms like Spark for big data or TensorFlow and Torch for deep learning have adopted Python interfaces to reach the widest set of data scientists. The two classes of algorithms emphasized in the first edition continue to be heavy favorites and are now available as part of PySpark.

The beauty of this marriage is that the code required to build machine learning models on truly gargantuan data sets is no more complicated than what's required on smaller data sets.

PySpark illustrates several important developments, making it cleaner and easier to invoke very powerful machine learning tools through relatively simple easy to read and write Python code. When the first edition of this book was written, building machine learning models on very large data sets required spinning up hundreds of processors, which required vast knowledge of data center processes and programming. It was cumbersome and frankly not very effective. Spark architecture was developed to correct this difficulty.

Spark made it possible to easily rent and employ large numbers of processors for machine learning. PySpark added a Python interface. The result is that the code to run a machine learning algorithm in PySpark is not much more complicated than to run the plain Python versions of programs. The algorithms that were the focus of the first edition continue to be heavily used favorites and are available in Spark. So it seemed natural to add PySpark examples alongside the Python examples in order to familiarize readers with PySpark.

In this edition all the code examples are in Python 3, since Python 2 is due to fall out of support and, in addition to providing the code in text form, the code is also available in Jupyter notebooks for each chapter. The notebook code when executed will draw graphs and tables you see in the figures.

How This Book Is Structured

This book follows the basic order in which you would approach a new prediction problem. The beginning involves developing an understanding of the data and determining how to formulate the problem, and then proceeds to try an algorithm and measure the performance. In the midst of this sequence, the book outlines

the methods and reasons for the steps as they come up. Chapter 1 gives a more thorough description of the types of problems that this book covers and the methods that are used. The book uses several data sets from the UC Irvine data repository as examples, and Chapter 2 exhibits some of the methods and tools that you can use for developing insight into a new data set. Chapter 3, "Predictive Model Building: Balancing Performance, Complexity, and Big Data," talks about the difficulties of predictive analytics and techniques for addressing them. It outlines the relationships between problem complexity, model complexity, data set size, and predictive performance. It discusses overfitting and how to reliably sense overfitting. It talks about performance metrics for different types of problems. Chapters 4 and 5, respectively, cover the background on penalized linear regression and its application to problems explored in Chapter 2. Chapters 6 and 7 cover background and application for ensemble methods.

What You Need to Use This Book

To run the code examples in the book, you need to have Python 3.x, SciPy, numpy, pandas, and scikit-learn and PySpark. These can be difficult to install due to cross-dependencies and version issues. To make the installation easy, I've used a free distribution of these packages that's available from Continuum Analytics (`http://continuum.io/`). Its Anaconda product is a free download and includes Python 3.x and all the packages you need to run the code in this book (and more). I've run the examples on Ubuntu 14.04 Linux but haven't tried them on other operating systems.

PySpark will need a Linux environment. If you're not running on Linux, then probably the easiest way to run the examples will be to use a virtual machine. Virtual Box is a free open source virtual machine—follow the directions to download Virtual Box and then install Ubuntu 18.05 and use Anaconda to install Python, PySpark, etc. You'll only need to employ a VM to run the PySpark examples. The non-Spark code will run anywhere you can open a Jupyter notebook.

Reader Support for This Book

Source code available in the book's repository can help you speed your learning. The chapters include installation instructions so that you can get coding along with reading the book.

Source Code

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book.

All the source code used in this book is available for download from `http://www.wiley.com/go/pythonmachinelearning2e`. You will find the code snippets from the source code are accompanied by a download icon and note indicating the name of the program so that you know it's available for download and can easily locate it in the download file.

Besides providing the code in text form, it is also included in a Python notebook. If you know how to run a Jupyter notebook, you can run the code cell-by-cell. The output will appear in the notebook, the figures will get drawn, and printed output will appear below the code block.

After you download the code, just decompress it with your favorite compression tool.

How to Contact the Publisher

If you believe you've found a mistake in this book, please bring it to our attention. At John Wiley & Sons, we understand how important it is to provide our customers with accurate content, but even with our best efforts an error may occur.

In order to submit your possible errata, please email it to our Customer Service Team at `wileysupport@wiley.com` with the subject line "Possible Book Errata Submission".

CHAPTER

1

The Two Essential Algorithms for Making Predictions

This book focuses on the machine learning process and so covers just a few of the most effective and widely used algorithms. It does not provide a survey of machine learning techniques. Too many of the algorithms that might be included in a survey are not actively used by practitioners.

This book deals with one class of machine learning problems, generally referred to as *function approximation*. Function approximation is a subset of problems that are called *supervised learning* problems. Linear regression and its classifier cousin, logistic regression, provide familiar examples of algorithms for function approximation problems. Function approximation problems include an enormous breadth of practical classification and regression problems in all sorts of arenas, including text classification, search responses, ad placements, spam filtering, predicting customer behavior, diagnostics, and so forth. The list is almost endless.

Broadly speaking, this book covers two classes of algorithms for solving function approximation problems: penalized linear regression methods and ensemble methods. This chapter introduces you to both of these algorithms, outlines some of their characteristics, and reviews the results of comparative studies of algorithm performance in order to demonstrate their consistent high performance.

This chapter then discusses the process of building predictive models. It describes the kinds of problems that you'll be able to address with the tools covered here and the flexibilities that you have in how you set up your problem

Machine Learning with Spark™ and Python®: Essential Techniques for Predictive Analytics, Second Edition. Michael Bowles.

and define the features that you'll use for making predictions. It describes process steps involved in building a predictive model and qualifying it for deployment.

Why Are These Two Algorithms So Useful?

Several factors make the penalized linear regression and ensemble methods a useful collection. Stated simply, they will provide optimum or near-optimum performance on the vast majority of predictive analytics (function approximation) problems encountered in practice, including big data sets, little data sets, wide data sets, tall skinny data sets, complicated problems, and simple problems. Evidence for this assertion can be found in two papers by Rich Caruana and his colleagues:

- "An Empirical Comparison of Supervised Learning Algorithms," by Rich Caruana and Alexandru Niculescu-Mizil[1]
- "An Empirical Evaluation of Supervised Learning in High Dimensions," by Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina[2]

In those two papers, the authors chose a variety of classification problems and applied a variety of different algorithms to build predictive models. The models were run on test data that were not included in training the models, and then the algorithms included in the studies were ranked on the basis of their performance on the problems. The first study compared 9 different basic algorithms on 11 different machine learning (binary classification) problems. The problems used in the study came from a wide variety of areas, including demographic data, text processing, pattern recognition, physics, and biology. Table 1.1 lists the data sets used in the study using the same names given by the study authors. The table shows how many attributes were available for predicting outcomes for each of the data sets, and it shows what percentage of the examples were positive.

Table 1.1: Sketch of Problems in Machine Learning Comparison Study

DATA SET NAME	NUMBER OF ATTRIBUTES	% OF EXAMPLES THAT ARE POSITIVE
Adult	14	25
Bact	11	69
Cod	15	50
Calhous	9	52
Cov_Type	54	36
HS	200	24

DATA SET NAME	NUMBER OF ATTRIBUTES	% OF EXAMPLES THAT ARE POSITIVE
Letter.p1	16	3
Letter.p2	16	53
Medis	63	11
Mg	124	17
Slac	59	50

The term *positive example* in a classification problem means an experiment (a line of data from the input data set) in which the outcome is positive. For example, if the classifier is being designed to determine whether a radar return signal indicates the presence of an airplane, then the positive example would be those returns where there was actually an airplane in the radar's field of view. The term *positive* comes from this sort of example where the two outcomes represent presence or absence. Other examples include presence or absence of disease in a medical test or presence or absence of cheating on a tax return.

Not all classification problems deal with presence or absence. For example, determining the gender of an author by machine-reading his or her text or machine-analyzing a handwriting sample has two classes—male and female—but there's no sense in which one is the absence of the other. In these cases, there's some arbitrariness in the assignment of the designations "positive" and "negative." The assignments of positive and negative can be arbitrary, but once chosen must be used consistently.

Some of the problems in the first study had many more examples of one class than the other. These are called *unbalanced*. For example, the two data sets Letter.p1 and Letter.p2 pose closely related problems in correctly classifying typed uppercase letters in a wide variety of fonts. The task with Letter.p1 is to correctly classify the letter *O* in a standard mix of letters. The task with Letter .p2 is to correctly classify *A–M* versus *N–Z*. The percentage of positives shown in Table 1.1 reflects this difference.

Table 1.1 also shows the number of "attributes" in each of the data sets. Attributes are the variables you have available to base a prediction on. For example, to predict whether or not an airplane will arrive at its destination on time, you might incorporate attributes such as the name of the airline company, the make and year of the airplane, the level of precipitation at the destination airport, the wind speed and direction along the flight path, and so on. Having a lot of attributes upon which to base a prediction can be a blessing and a curse. Attributes that relate directly to the outcomes being predicted are a blessing. Attributes that are unrelated to the outcomes are a curse. Telling the difference between blessed and cursed attributes requires data. Chapter 3, "Predictive Model Building: Balancing Performance, Complexity, and Big Data," goes into that in more detail.

Table 1.2 shows how the algorithms covered in this book fared relative to the other algorithms used in the study. Table 1.2 shows which algorithms showed the top five performance scores for each of the problems listed in Table 1.1. Algorithms covered in this book are spelled out (boosted decision trees, Random Forests, bagged decision trees, and logistic regression). The first three of these are ensemble methods. Penalized regression was not fully developed when the study was done and wasn't evaluated. Logistic regression is a close relative and is used to gauge the success of regression methods. Each of the 9 algorithms used in the study had 3 different data reduction techniques applied, for a total of 27 combinations. The top five positions represent roughly the top 20 percent of performance scores. The row next to the heading Covt indicates that the boosted decision trees algorithm was the first and second best relative to performance, the Random Forests algorithm was the fourth and fifth best, and the bagged decision trees algorithm was the third best. In the cases where algorithms not covered here were in the top five, an entry appears in the Other column. The algorithms that show up there are *k-nearest neighbors* (KNNs), *artificial neural nets* (ANNs), and *support vector machines* (SVMs).

Table 1.2: How the Algorithms Covered in This Book Compare on Different Problems

ALGORITHM	BOOSTED DECISION TREES	RANDOM FORESTS	BAGGED DECISION TREES	LOGISTIC REGRESSION	OTHER
Covt	1, 2	4, 5	3		
Adult	1, 4	2	3, 5		
LTR.P1	1				SVM, KNN
LTR.P2	1, 2	4, 5			SVM
MEDIS		1, 3		5	ANN
SLAC		1, 2, 3	4, 5		
HS	1, 3				ANN
MG		2, 4, 5	1, 3		
CALHOUS	1, 2	5	3, 4		
COD	1, 2		3, 4, 5		
BACT	2, 5		1, 3, 4		

Logistic regression captures top-five honors in only one case in Table 1.2. The reason for that is that these data sets have few attributes (at most 200) relative to examples (5,000 in each data set). There's plenty of data to resolve a model with

so few attributes, and yet the training sets are small enough that the training time is not excessive.

As you'll see in Chapter 3 and in the examples covered in Chapter 5, "Building Predictive Models Using Penalized Linear Methods," and Chapter 7, "Building Ensemble Models with Python," the penalized regression methods perform best relative to other algorithms when there are numerous attributes and not enough examples or time to train a more complicated ensemble model.

Caruana et al. have run a newer study (2008) to address how these algorithms compare when the number of attributes increases. That is, how do these algorithms compare on big data? A number of fields have significantly more attributes than the data sets in the first study. For example, genomic problems have several tens of thousands of attributes (one attribute per gene), and text mining problems can have millions of attributes (one attribute per distinct word or per distinct pair of words). Table 1.3 shows how linear regression and ensemble methods fare as the number of attributes grows. The results in Table 1.3 show the ranking of the algorithms used in the second study. The table shows the performance on each of the problems individually and in the far right column shows the ranking of each algorithm's average score across all the problems. The algorithms used in the study are broken into two groups. The top group of algorithms are ones that will be covered in this book. The bottom group will not be covered.

The problems shown in Table 1.3 are arranged in order of their number of attributes, ranging from 761 to 685,569. Linear (logistic) regression is in the top three for 5 of the 11 test cases used in the study. Those superior scores were concentrated among the larger data sets. Notice that boosted decision tree (denoted by BSTDT in Table 1.3) and Random Forests (denoted by RF in Table 1.3) algorithms still perform near the top. They come in first and second for overall score on these problems.

The algorithms covered in this book have other advantages besides raw predictive performance. An important benefit of the penalized linear regression models that the book covers is the speed at which they train. On big problems, training speed can become an issue. In some problems, model training can take days or weeks. This time frame can be an intolerable delay, particularly early in development when iterations are required to home in on the best approach. Besides training very quickly, after being deployed a trained linear model can produce predictions very quickly—quickly enough for high-speed trading or Internet ad insertions. The study demonstrates that penalized linear regression can provide the best answers available in many cases and be near the top even in cases where they are not the best.

Table 1.3: How the Algorithms Covered in This Book Compare on Big Data Problems

DIM	761 STURN	761 CALAM	780 DIGITS	927 TIS	1344 CRYST	3448 KDD98	20958 R-S	105354 CITE	195203 DSE	405333 SPAM	685569 IMDB	MEAN
BSTDT	8	1	2	6	1	3	8	1	7	6	3	1
RF	9	4	3	3	2	1	6	5	3	1	3	2
BAGDT	5	2	6	4	3	1	9	1	6	7	3	4
BSTST	2	3	7	7	7	1	7	4	8	8	5	7
LR	4	8	9	1	4	1	2	2	2	4	4	6
SVM	3	5	5	2	5	2	1	1	5	5	3	3
ANN	6	7	4	5	8	1	4	2	1	3	3	5
KNN	1	6	1	9	6	2	10	1	7	9	6	8
PRC	7	9	8	8	7	1	3	3	4	2	2	9
NB	10	10	10	10	9	1	5	1	9	10	7	10

In addition, these algorithms are reasonably easy to use. They do not have very many tunable parameters. They have well-defined and well-structured input types. They solve several types of problems in regression and classification. It is not unusual to be able to arrange the input data and generate a first trained model and performance predictions within an hour or two of starting a new problem.

One of their most important features is that they indicate which of their input variables is most important for producing predictions. This turns out to be an invaluable feature in a machine learning algorithm. One of the most time-consuming steps in the development of a predictive model is what is sometimes called *feature selection* or *feature engineering*. This is the process whereby the data scientist chooses the variables that will be used to predict outcomes. By ranking features according to importance, the algorithms covered in this book aid in the feature-engineering process by taking some of the guesswork out of the development process and making the process more sure.

What Are Penalized Regression Methods?

Penalized linear regression is a derivative of *ordinary least squares* (OLS) regression—a method developed by Gauss and Legendre roughly 200 years ago. Penalized linear regression methods were designed to overcome some basic limitations of OLS regression. The basic problem with OLS is that sometimes it overfits the problem. Think of OLS as fitting a line through a group of points, as in Figure 1.1. This is a simple prediction problem: predicting y, the target value given a single attribute x. For example, the problem might be to predict men's salaries using only their heights. Height is slightly predictive of salaries for men (but not for women).

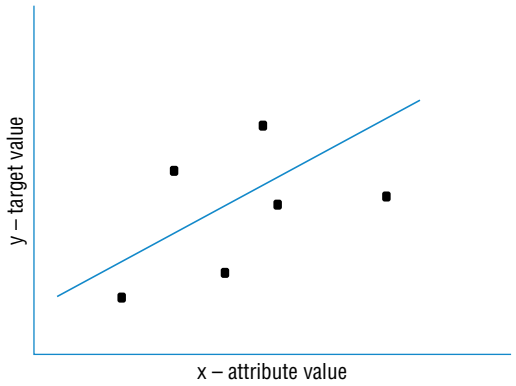


Figure 1.1: Ordinary least squares fit

The points represent men's salaries versus their heights. The line in Figure 1.1 represents the OLS solution to this prediction problem. In some sense, the line is the best predictive model for men's salaries given their heights. The data set has six points in it. Suppose that the data set had only two points in it. Imagine that there's a population of points, like the ones in Figure 1.1, but that you do not get to see all the points. Maybe they are too expensive to generate, like the genetic data mentioned earlier. There are enough humans available to isolate the gene that is the culprit; the problem is that you do not have gene sequences for many of them because of cost.

To simulate this in the simple example, imagine that instead of six points you're given only two of the six points. How would that change the nature of the line fit to those points? It would depend on which two points you happened to get. To see how much effect that would have, pick any two points from Figure 1.1 and imagine a line through them. Figure 1.2 shows some of the possible lines through pairs of points from Figure 1.1. Notice how much the lines vary depending on the choice of points.

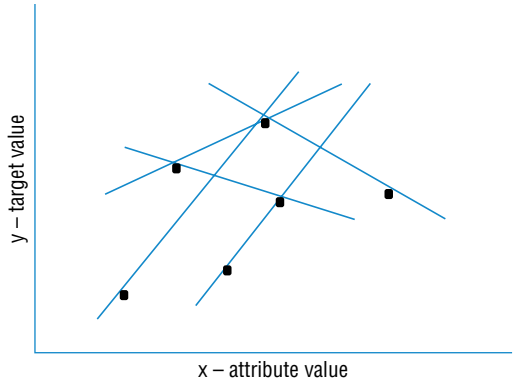


Figure 1.2: Fitting lines with only two points

The problem with having only two points to fit a line is that there is not enough data for the number of degrees of freedom. A line has two degrees of freedom. Having two degrees of freedom means that there are two independent parameters that uniquely determine a line. You can imagine grabbing hold of a line in the plane and sliding it up and down in the plane or twisting it to change its slope. So, vertical position and slope are independent. They can be changed separately, and together they completely specify a line. The degrees of freedom of a line can be expressed in several equivalent ways (where it intercepts the y-axis and its slope, two points that are on the line, and so on). All of these representations of a line require two parameters to specify.

When the number of degrees of freedom is equal to the number of points, the predictions are not very good. The lines hit the points used to draw them, but there is a lot of variation among lines drawn with different pairs of points. You

cannot place much faith in a prediction that has as many degrees of freedom as the number of points in your data set. The plot in Figure 1.1 had six points and fit a line (two degrees of freedom) through them. That is six points and two degrees of freedom. The thought problem of determining the genes causing a heritable condition illustrated that having more genes to choose from makes it necessary to have more data in order to isolate a cause from among the 20,000 or so possible human genes. The 20,000 different genes represent 20,000 degrees of freedom. Data from even 20,000 different persons will not suffice to get a reliable answer, and in many cases, all that can be afforded within the scope of a reasonable study is a sample from 500 or so persons. That is where penalized linear regression may be the best algorithm choice.

Penalized linear regression provides a way to systematically reduce degrees of freedom to match the amount of data available and the complexity of the underlying phenomena. These methods have become very popular for problems with very many degrees of freedom. They are a favorite for genetic problems where the number of degrees of freedom (that is, the number of genes) can be several tens of thousands and for problems like text classification where the number of degrees of freedom can be more than a million. Chapter 4, "Penalized Linear Regression," gives more detail on how these methods work, sample code that illustrates the mechanics of these algorithms, and examples of the process for implementing machine learning systems using available Python packages.

What Are Ensemble Methods?

The other family of algorithms covered in this book is ensemble methods. The basic idea with ensemble methods is to build a horde of different predictive models and then combine their outputs—by averaging the outputs or taking the majority answer (voting). The individual models are called *base learners*. Some results from computational learning theory show that if the base learners are just slightly better than random guessing, the performance of the ensemble can be very good if there is a sufficient number of independent models.

One of the problems spurring the development of ensemble methods has been the observation that some particular machine learning algorithms exhibit instability. For example, the addition of fresh data to the data set might result in a radical change in the resulting model or its performance. Binary decision trees and traditional neural nets exhibit this sort of instability. This instability causes high variance in the performance of models, and averaging many models can be viewed as a way to reduce the variance. The trick is how to generate large numbers of independent models, particularly if they are all using the same base learner. Chapter 6, "Ensemble Methods," will get into the details of how this is done. The techniques are ingenious, and it is relatively easy to understand their basic principles of operation. Here is a preview of what's in store.