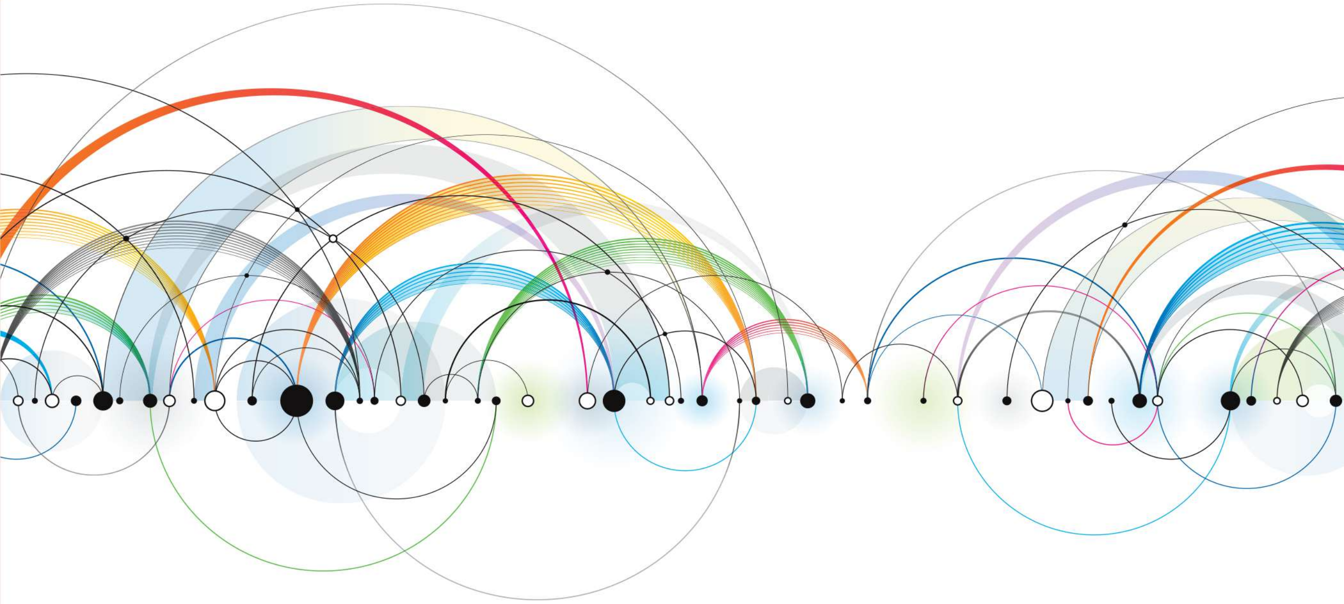
"A must-read resource for anyone who is serious about embracing the opportunity of big data."

—Craig Vaughan, Global Vice President, SAP

Data Science *for* Business

What You Need to Know About Data Mining and Data-Analytic Thinking



Foster Provost & Tom Fawcett

Data Science / Business

Data Science *for* Business

What You Need to Know About Data Mining and Data-Analytic Thinking

This broad, deep, but not-too-technical guide introduces you to the fundamental principles of data science and walks you through the "data-analytic thinking" necessary for extracting useful knowledge and business value from the data you collect. By learning data science principles, you will understand the many data mining techniques in use today. More importantly, these principles underpin the processes and strategies necessary to solve business problems through data mining techniques.

"This book goes beyond data analytics 101. It's the essential guide for those of us (all of us?) whose businesses are built on the ubiquity of data opportunities and the new mandate for data-driven decision-making."

—Tom Phillips, CEO Media6Degrees; former Head of Google Search and Analytics

"The authors, both renowned experts in data science before it had a name, have taken a complex topic and made it accessible to all levels. This is the first book of its kind, with a focus on data science concepts as applied to practical business problems. It is liberally sprinkled with compelling real-world examples outlining familiar, accessible problems in the business world: customer churn, targeted marketing, even whiskey analytics!

The book is unique in that it does not give a cookbook of algorithms, rather it helps the reader understand the underlying concepts behind data science, and most importantly how to approach and be successful at problem solving. Whether you are looking for a good comprehensive overview of data science or are a budding data scientist in need of the basics, this is a must-read."

—Chris Volinsky, Director, Statistics Research, AT&T Labs
Winner of the $1 Million Netflix Challenge

"Data is the foundation of new waves of productivity growth, innovation, and richer customer insight. Only recently viewed broadly as a source of competitive advantage, dealing well with data is rapidly becoming table stakes to stay in the game. The authors' deep applied experience makes this a must read—a window into your competitor's strategy."

—Alan Murray, Serial Entrepreneur; Partner Coriolis Ventures

US $34.99 **CAN $36.99**
ISBN: 978-1-449-36132-7
53499
9 781449 361327

Praise

"A must-read resource for anyone who is serious about embracing the opportunity of big data."

— *Craig Vaughan*
Global Vice President at SAP

"This timely book says out loud what has finally become apparent: in the modern world, Data is Business, and you can no longer think business without *thinking data.* Read this book and you will understand the Science behind thinking data."

— *Ron Bekkerman*
Chief Data Officer at Carmel Ventures

"A great book for business managers who lead or interact with data scientists, who wish to better understand the principals and algorithms available without the technical details of single-disciplinary books."

— *Ronny Kohavi*
Partner Architect at Microsoft Online Services Division

"Provost and Fawcett have distilled their mastery of both the art and science of real-world data analysis into an unrivalled introduction to the field."

—*Geoff Webb*
Editor-in-Chief of *Data Mining and Knowledge Discovery* Journal

"I would love it if everyone I had to work with had read this book."

— *Claudia Perlich*
Chief Scientist of M6D (Media6Degrees) and Advertising Research Foundation Innovation Award Grand Winner (2013)

“A foundational piece in the fast developing world of Data Science. A must read for anyone interested in the Big Data revolution."

—Justin Gapper
Business Unit Analytics Manager
at Teledyne Scientific and Imaging

“The authors, both renowned experts in data science before it had a name, have taken a complex topic and made it accessible to all levels, but mostly helpful to the budding data scientist. As far as I know, this is the first book of its kind—with a focus on data science concepts as applied to practical business problems. It is liberally sprinkled with compelling real-world examples outlining familiar, accessible problems in the business world: customer churn, targeted marking, even whiskey analytics!

The book is unique in that it does not give a cookbook of algorithms, rather it helps the reader understand the underlying concepts behind data science, and most importantly how to approach and be successful at problem solving. Whether you are looking for a good comprehensive overview of data science or are a budding data scientist in need of the basics, this is a must-read.”

— Chris Volinsky
Director of Statistics Research at AT&T Labs and Winning
Team Member for the $1 Million Netflix Challenge

“This book goes beyond data analytics 101. It’s the essential guide for those of us (all of us?) whose businesses are built on the ubiquity of data opportunities and the new mandate for data-driven decision-making.”

—Tom Phillips
CEO of Media6Degrees and Former Head of
Google Search and Analytics

“Intelligent use of data has become a force powering business to new levels of competitiveness. To thrive in this data-driven ecosystem, engineers, analysts, and managers alike must understand the options, design choices, and tradeoffs before them. With motivating examples, clear exposition, and a breadth of details covering not only the “hows” but the “whys”, *Data Science for Business* is the perfect primer for those wishing to become involved in the development and application of data-driven systems.”

—Josh Attenberg
Data Science Lead at Etsy

"Data is the foundation of new waves of productivity growth, innovation, and richer customer insight. Only recently viewed broadly as a source of competitive advantage, dealing well with data is rapidly becoming table stakes to stay in the game. The authors' deep applied experience makes this a must read—a window into your competitor's strategy."

— *Alan Murray*
Serial Entrepreneur; Partner at Coriolis Ventures

"One of the best data mining books, which helped me think through various ideas on liquidity analysis in the FX business. The examples are excellent and help you take a deep dive into the subject! This one is going to be on my shelf for lifetime!"

— *Nidhi Kathuria*
Vice President of FX at Royal Bank of Scotland

Data Science for Business

Foster Provost and Tom Fawcett

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Data Science for Business

by Foster Provost and Tom Fawcett

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Mike Loukides and Meghan Blanchette
Production Editor: Christopher Hearse
Proofreader: Kiel Van Horn
Indexer: WordCo Indexing Services, Inc.
Cover Designer: Mark Paglietti
Interior Designer: David Futato
Illustrator: Rebecca Demarest

July 2013: First Edition

Revision History for the First Edition:

2013-07-25: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449361327* for release details.

ISBN: 978-1-449-36132-7

[LSI]

Table of Contents

Preface

Data Science for Business is intended for several sorts of readers:

- Business people who will be working with data scientists, managing data science–oriented projects, or investing in data science ventures,
- Developers who will be implementing data science solutions, and
- Aspiring data scientists.

This is not a book about algorithms, nor is it a replacement for a book about algorithms. We deliberately avoided an algorithm-centered approach. We believe there is a relatively small set of fundamental concepts or principles that underlie techniques for extracting useful knowledge from data. These concepts serve as the *foundation* for many well-known algorithms of data mining. Moreover, these concepts underlie the analysis of data-centered business problems, the creation and evaluation of data science solutions, and the evaluation of general data science strategies and proposals. Accordingly, we organized the exposition around these general principles rather than around specific algorithms. Where necessary to describe procedural details, we use a combination of text and diagrams, which we think are more accessible than a listing of detailed algorithmic steps.

The book does not presume a sophisticated mathematical background. However, by its very nature the material is somewhat technical—the goal is to impart a significant understanding of data science, not just to give a high-level overview. In general, we have tried to minimize the mathematics and make the exposition as "conceptual" as possible.

Colleagues in industry comment that the book is invaluable for helping to align the understanding of the business, technical/development, and data science teams. That observation is based on a small sample, so we are curious to see how general it truly is (see Chapter 5!). Ideally, we envision a book that any data scientist would give to his collaborators from the development or business teams, effectively saying: if you really want to design/implement top-notch data science solutions to business problems, we all need to have a common understanding of this material.

Colleagues also tell us that the book has been quite useful in an unforeseen way: for preparing to interview data science job candidates. The demand from business for hiring data scientists is strong and increasing. In response, more and more job seekers are presenting themselves as data scientists. Every data science job candidate should understand the fundamentals presented in this book. (Our industry colleagues tell us that they are surprised how many do not. We have half-seriously discussed a follow-up pamphlet "Cliff's Notes to Interviewing for Data Science Jobs.")

Our Conceptual Approach to Data Science

In this book we introduce a collection of the most important fundamental concepts of data science. Some of these concepts are "headliners" for chapters, and others are introduced more naturally through the discussions (and thus they are not necessarily labeled as fundamental concepts). The concepts span the process from envisioning the problem, to applying data science techniques, to deploying the results to improve decision-making. The concepts also undergird a large array of business analytics methods and techniques.

The concepts fit into three general types:

1. Concepts about how data science fits in the organization and the competitive landscape, including ways to attract, structure, and nurture data science teams; ways for thinking about how data science leads to competitive advantage; and tactical concepts for doing well with data science projects.
2. General ways of thinking data-analytically. These help in identifying appropriate data and consider appropriate methods. The concepts include the *data mining process* as well as the collection of different *high-level data mining tasks*.
3. General concepts for actually extracting knowledge from data, which undergird the vast array of data science tasks and their algorithms.

For example, one fundamental concept is that of determining the similarity of two entities described by data. This ability forms the basis for various specific tasks. It may be used directly to *find* customers similar to a given customer. It forms the core of several *prediction* algorithms that estimate a target value such as the expected resouce usage of a client or the probability of a customer to respond to an offer. It is also the basis for *clustering* techniques, which group entities by their shared features without a focused objective. Similarity forms the basis of *information retrieval*, in which documents or webpages relevant to a search query are retrieved. Finally, it underlies several common algorithms for *recommendation*. A traditional algorithm-oriented book might present each of these tasks in a different chapter, under different names, with common aspects buried in algorithm details or mathematical propositions. In this book we instead focus on the unifying concepts, presenting specific tasks and algorithms as natural manifestations of them.

As another example, in evaluating the utility of a pattern, we see a notion of *lift*— how much more prevalent a pattern is than would be expected by chance—recurring broadly across data science. It is used to evaluate very different sorts of patterns in different contexts. Algorithms for targeting advertisements are evaluated by computing the lift one gets for the targeted population. Lift is used to judge the weight of evidence for or against a conclusion. Lift helps determine whether a co-occurrence (an association) in data is interesting, as opposed to simply being a natural consequence of popularity.

We believe that explaining data science around such fundamental concepts not only aids the reader, it also facilitates communication between business stakeholders and data scientists. It provides a shared vocabulary and enables both parties to understand each other better. The shared concepts lead to deeper discussions that may uncover critical issues otherwise missed.

To the Instructor

This book has been used successfully as a textbook for a very wide variety of data science courses. Historically, the book arose from the development of Foster's multidisciplinary Data Science classes at the Stern School at NYU, starting in the fall of 2005.[1] The original class was nominally for MBA students and MSIS students, but drew students from schools across the university. The most interesting aspect of the class was not that it appealed to MBA and MSIS students, for whom it was designed. More interesting, it also was found to be very valuable by students with strong backgrounds in machine learning and other technical disciplines. Part of the reason seemed to be that the focus on fundamental principles and other issues besides algorithms was missing from their curricula.

At NYU we now use the book in support of a variety of data science–related programs: the original MBA and MSIS programs, undergraduate business analytics, NYU/Stern's new MS in Business Analytics program, and as the Introduction to Data Science for NYU's new MS in Data Science. In addition, (prior to publication) the book has been adopted by more than a dozen other universities for programs in seven countries (and counting), in business schools, in computer science programs, and for more general introductions to data science.

Stay tuned to the books' websites (see below) for information on how to obtain helpful instructional material, including lecture slides, sample homework questions and prob-

1. Of course, each author has the distinct impression that he did the majority of the work on the book.

lems, example project instructions based on the frameworks from the book, exam questions, and more to come.

> We keep an up-to-date list of known adoptees on the book's website. Click *Who's Using It* at the top.

Other Skills and Concepts

There are many other concepts and skills that a practical data scientist needs to know besides the fundamental principles of data science. These skills and concepts will be discussed in Chapter 1 and Chapter 2. The interested reader is encouraged to visit the book's website for pointers to material for learning these additional skills and concepts (for example, scripting in Python, Unix command-line processing, datafiles, common data formats, databases and querying, big data architectures and systems like MapReduce and Hadoop, data visualization, and other related topics).

Sections and Notation

In addition to occasional footnotes, the book contains boxed "sidebars." These are essentially extended footnotes. We reserve these for material that we consider interesting and worthwhile, but too long for a footnote and too much of a digression for the main text.

> **A note on the starred, "curvy road" sections**
> The occasional mathematical details are relegated to optional "starred" sections. These section titles will have asterisk prefixes, and they will include the "curvy road" graphic you see to the left to indicate that the section contains more detailed mathematics or technical details than elsewhere. The book is written so that these sections may be skipped without loss of continuity, although in a few places we remind readers that details appear there.

Constructions in the text like (Smith and Jones, 2003) indicate a reference to an entry in the bibliography (in this case, the 2003 article or book by Smith and Jones); "Smith and Jones (2003)" is a similar reference. A single bibliography for the entire book appears in the endmatter.

In this book we try to keep math to a minimum, and what math there is we have simplified as much as possible without introducing confusion. For our readers with technical backgrounds, a few comments may be in order regarding our simplifying choices.

1. We avoid Sigma (Σ) and Pi (Π) notation, commonly used in textbooks to indicate sums and products, respectively. Instead we simply use equations with ellipses like this:

   $$f(x) = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

2. Statistics books are usually careful to distinguish between a value and its estimate by putting a "hat" on variables that are estimates, so in such books you'll typically see a true probability denoted p and its estimate denoted $\hat{p}$. In this book we are almost always talking about estimates from data, and putting hats on everything makes equations verbose and ugly. Everything should be assumed to be an estimate from data unless we say otherwise.
3. We simplify notation and remove extraneous variables where we believe they are clear from context. For example, when we discuss classifiers mathematically, we are technically dealing with decision predicates over feature vectors. Expressing this formally would lead to equations like:

   $$\hat{f}_R(\mathbf{x}) = x_{\text{Age}} \times -1 + 0.7 \times x_{\text{Balance}} + 60$$

   Instead we opt for the more readable:

   $$f(\mathbf{x}) = Age \times -1 + 0.7 \times \text{Balance} + 60$$

   with the understanding that **x** is a vector and *Age* and *Balance* are components of it.

We have tried to be consistent with typography, reserving fixed-width typewriter fonts like `sepal_width` to indicate attributes or keywords in data. For example, in the text-mining chapter, a word like *'discussing'* designates a word in a document while `discuss` might be the resulting token in the data.

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Examples

In addition to being an introduction to data science, this book is intended to be useful in discussions of and day-to-day work in the field. Answering a question by citing this book and quoting examples does not require permission. We appreciate, but do not require, attribution. Formal attribution usually includes the title, author, publisher, and ISBN. For example: "*Data Science for Business* by Foster Provost and Tom Fawcett (O'Reilly). Copyright 2013 Foster Provost and Tom Fawcett, 978-1-449-36132-7."

If you feel your use of examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Safari® Books Online

How to Contact Us

Please address comments and questions concerning this book to the publisher:

> O'Reilly Media, Inc.
> 1005 Gravenstein Highway North
> Sebastopol, CA 95472
> 800-998-9938 (in the United States or Canada)
> 707-829-0515 (international or local)
> 707-829-0104 (fax)

We have two web pages for this book, where we list errata, examples, and any additional information. You can access the publisher's page at *http://oreil.ly/data-science* and the authors' page at *http://www.data-science-for-biz.com*.

To comment or ask technical questions about this book, send email to *bookques tions@oreilly.com*.

For more information about O'Reilly Media's books, courses, conferences, and news, see their website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Acknowledgments

Thanks to all the many colleagues and others who have provided invaluable feedback, criticism, suggestions, and encouragement based on many prior draft manuscripts. At the risk of missing someone, let us thank in particular: Panos Adamopoulos, Manuel Arriaga, Josh Attenberg, Solon Barocas, Ron Bekkerman, Josh Blumenstock, Aaron Brick, Jessica Clark, Nitesh Chawla, Peter Devito, Vasant Dhar, Jan Ehmke, Theos Evgeniou, Justin Gapper, Tomer Geva, Daniel Gillick, Shawndra Hill, Nidhi Kathuria, Ronny Kohavi, Marios Kokkodis, Tom Lee, David Martens, Sophie Mohin, Lauren Moores, Alan Murray, Nick Nishimura, Balaji Padmanabhan, Jason Pan, Claudia Perlich, Gregory Piatetsky-Shapiro, Tom Phillips, Kevin Reilly, Maytal Saar-Tsechansky, Evan Sadler, Galit Shmueli, Roger Stein, Nick Street, Kiril Tsemekhman, Craig Vaughan, Chris Volinsky, Wally Wang, Geoff Webb, and Rong Zheng. We would also like to thank more generally the students from Foster's classes, Data Mining for Business Analytics, Practical Data Science, and the Data Science Research Seminar. Questions and issues that arose when using prior drafts of this book provided substantive feedback for improving it.

Thanks to David Stillwell, Thore Graepel, and Michal Kosinski for providing the Facebook Like data for some of the examples. Thanks to Nick Street for providing the cell nuclei data and for letting us use the cell nuclei image in Chapter 4. Thanks to David Martens for his help with the mobile locations visualization. Thanks to Chris Volinsky for providing data from his work on the Netflix Challenge. Thanks to Sonny Tambe for early access to his results on big data technologies and productivity. Thanks to Patrick Perry for pointing us to the bank call center example used in Chapter 12. Thanks to Geoff Webb for the use of the Magnum Opus association mining system.

Most of all we thank our families for their love, patience and encouragement.

A great deal of open source software was used in the preparation of this book and its examples. The authors wish to thank the developers and contributors of:

- Python and Perl
- Scipy, Numpy, Matplotlib, and Scikit-Learn
- Weka
- The Machine Learning Repository at the University of California at Irvine (Bache & Lichman, 2013)

Finally, we encourage readers to check our website for updates to this material, new chapters, errata, addenda, and accompanying slide sets.

—Foster Provost and Tom Fawcett

CHAPTER 1

Introduction: Data-Analytic Thinking

Dream no small dreams for they have no power to move the hearts of men.

—Johann Wolfgang von Goethe

The past fifteen years have seen extensive investments in business infrastructure, which have improved the ability to collect data throughout the enterprise. Virtually every aspect of business is now open to data collection and often even instrumented for data collection: operations, manufacturing, supply-chain management, customer behavior, marketing campaign performance, workflow procedures, and so on. At the same time, information is now widely available on external events such as market trends, industry news, and competitors' movements. This broad availability of data has led to increasing interest in methods for extracting useful information and knowledge from data—the realm of data science.

The Ubiquity of Data Opportunities

With vast amounts of data now available, companies in almost every industry are focused on exploiting data for competitive advantage. In the past, firms could employ teams of statisticians, modelers, and analysts to explore datasets manually, but the volume and variety of data have far outstripped the capacity of manual analysis. At the same time, computers have become far more powerful, networking has become ubiquitous, and algorithms have been developed that can connect datasets to enable broader and deeper analyses than previously possible. The convergence of these phenomena has given rise to the increasingly widespread business application of data science principles and data-mining techniques.

Probably the widest applications of data-mining techniques are in marketing for tasks such as targeted marketing, online advertising, and recommendations for cross-selling.

Data mining is used for general customer relationship management to analyze customer behavior in order to manage attrition and maximize expected customer value. The finance industry uses data mining for credit scoring and trading, and in operations via fraud detection and workforce management. Major retailers from Walmart to Amazon apply data mining throughout their businesses, from marketing to supply-chain management. Many firms have differentiated themselves strategically with data science, sometimes to the point of evolving into data mining companies.

The primary goals of this book are to help you view business problems from a data perspective and understand principles of extracting useful knowledge from data. There is a fundamental structure to data-analytic thinking, and basic principles that should be understood. There are also particular areas where intuition, creativity, common sense, and domain knowledge must be brought to bear. A data perspective will provide you with structure and principles, and this will give you a framework to systematically analyze such problems. As you get better at data-analytic thinking you will develop intuition as to how and where to apply creativity and domain knowledge.

Throughout the first two chapters of this book, we will discuss in detail various topics and techniques related to data science and data mining. The terms “data science” and “data mining” often are used interchangeably, and the former has taken a life of its own as various individuals and organizations try to capitalize on the current hype surrounding it. At a high level, *data science* is a set of fundamental principles that guide the extraction of knowledge from data. Data mining is the extraction of knowledge from data, via technologies that incorporate these principles. As a term, “data science” often is applied more broadly than the traditional use of “data mining,” but data mining techniques provide some of the clearest illustrations of the principles of data science.

It is important to understand data science even if you never intend to apply it yourself. Data-analytic thinking enables you to evaluate proposals for data mining projects. For example, if an employee, a consultant, or a potential investment target proposes to improve a particular business application by extracting knowledge from data, you should be able to assess the proposal systematically and decide whether it is sound or flawed. This does not mean that you will be able to tell whether it will actually succeed—for data mining projects, that often requires trying—but you should be able to spot obvious flaws, unrealistic assumptions, and missing pieces.

Throughout the book we will describe a number of fundamental data science principles, and will illustrate each with at least one data mining technique that embodies the principle. For each principle there are usually many specific techniques that embody it, so in this book we have chosen to emphasize the basic principles in preference to specific techniques. That said, we will not make a big deal about the difference between data science and data mining, except where it will have a substantial effect on understanding the actual concepts.

Let's examine two brief case studies of analyzing data to extract predictive patterns.

Example: Hurricane Frances

Consider an example from a *New York Times* story from 2004:

> Hurricane Frances was on its way, barreling across the Caribbean, threatening a direct hit on Florida's Atlantic coast. Residents made for higher ground, but far away, in Bentonville, Ark., executives at Wal-Mart Stores decided that the situation offered a great opportunity for one of their newest data-driven weapons … predictive technology.
>
> A week ahead of the storm's landfall, Linda M. Dillman, Wal-Mart's chief information officer, pressed her staff to come up with forecasts based on what had happened when Hurricane Charley struck several weeks earlier. Backed by the trillions of bytes' worth of shopper history that is stored in Wal-Mart's data warehouse, she felt that the company could 'start predicting what's going to happen, instead of waiting for it to happen,' as she put it. (Hays, 2004)

Consider *why* data-driven prediction might be useful in this scenario. It might be useful to predict that people in the path of the hurricane would buy more bottled water. Maybe, but this point seems a bit obvious, and why would we need data science to discover it? It might be useful to project the *amount of increase* in sales due to the hurricane, to ensure that local Wal-Marts are properly stocked. Perhaps mining the data could reveal that a particular DVD sold out in the hurricane's path—but maybe it sold out that week at Wal-Marts across the country, not just where the hurricane landing was imminent. The prediction could be somewhat useful, but is probably more general than Ms. Dillman was intending.

It would be more valuable to discover patterns due to the hurricane that were not obvious. To do this, analysts might examine the huge volume of Wal-Mart data from prior, similar situations (such as Hurricane Charley) to identify *unusual* local demand for products. From such patterns, the company might be able to anticipate unusual demand for products and rush stock to the stores ahead of the hurricane's landfall.

Indeed, that is what happened. *The New York Times* (Hays, 2004) reported that: "… the experts mined the data and found that the stores would indeed need certain products —and not just the usual flashlights. 'We didn't know in the past that strawberry Pop-Tarts increase in sales, like seven times their normal sales rate, ahead of a hurricane,' Ms. Dillman said in a recent interview. 'And the pre-hurricane top-selling item was beer.'"[1]

1. Of course! What goes better with strawberry Pop-Tarts than a nice cold beer?

Example: Predicting Customer Churn

How are such data analyses performed? Consider a second, more typical business scenario and how it might be treated from a data perspective. This problem will serve as a running example that will illuminate many of the issues raised in this book and provide a common frame of reference.

Assume you just landed a great analytical job with MegaTelCo, one of the largest telecommunication firms in the United States. They are having a major problem with customer retention in their wireless business. In the mid-Atlantic region, 20% of cell phone customers leave when their contracts expire, and it is getting increasingly difficult to acquire new customers. Since the cell phone market is now saturated, the huge growth in the wireless market has tapered off. Communications companies are now engaged in battles to attract each other's customers while retaining their own. Customers switching from one company to another is called *churn*, and it is expensive all around: one company must spend on incentives to attract a customer while another company loses revenue when the customer departs.

You have been called in to help understand the problem and to devise a solution. Attracting new customers is much more expensive than retaining existing ones, so a good deal of marketing budget is allocated to prevent churn. Marketing has already designed a special retention offer. Your task is to devise a precise, step-by-step plan for how the data science team should use MegaTelCo's vast data resources to decide which customers should be offered the special retention deal prior to the expiration of their contracts.

Think carefully about what data you might use and how they would be used. Specifically, how should MegaTelCo choose a set of customers to receive their offer in order to best reduce churn for a particular incentive budget? Answering this question is much more complicated than it may seem initially. We will return to this problem repeatedly through the book, adding sophistication to our solution as we develop an understanding of the fundamental data science concepts.

In reality, customer retention has been a major use of data mining technologies—especially in telecommunications and finance businesses. These more generally were some of the earliest and widest adopters of data mining technologies, for reasons discussed later.

Data Science, Engineering, and Data-Driven Decision Making

Data science involves principles, processes, and techniques for understanding phenomena via the (automated) analysis of data. In this book, we will view the ultimate goal

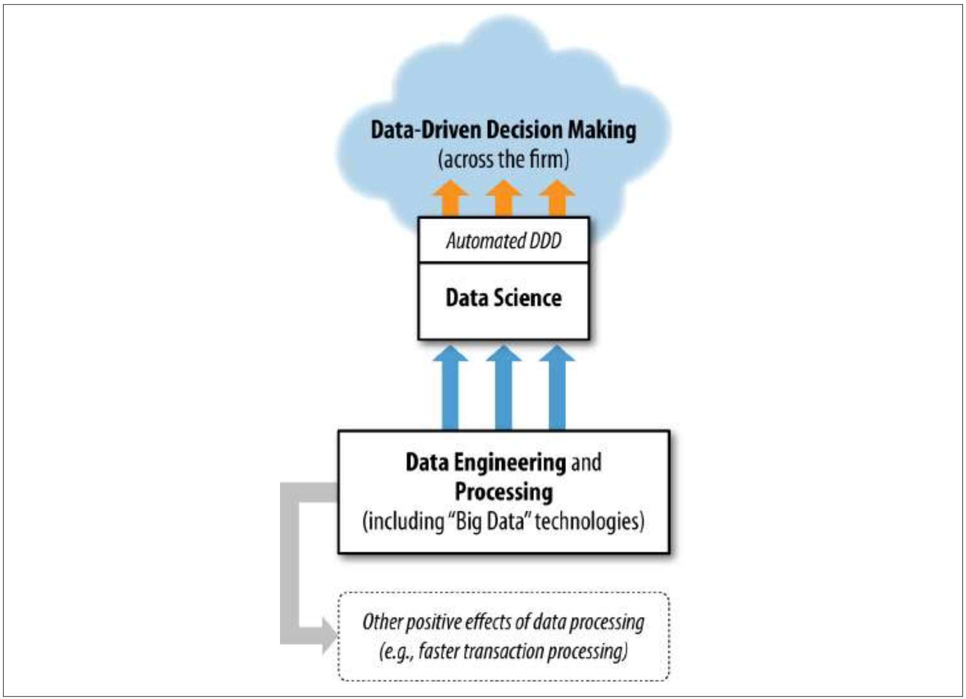

Data-Driven Decision Making
(across the firm)
Automated DDD
Data Science
Data Engineering and Processing
(including "Big Data" technologies)
Other positive effects of data processing
(e.g., faster transaction processing)

Figure 1-1. Data science in the context of various data-related processes in the organization.

of data science as improving decision making, as this generally is of direct interest to business.

Figure 1-1 places data science in the context of various other closely related and data-related processes in the organization. It distinguishes data science from other aspects of data processing that are gaining increasing attention in business. Let's start at the top.

Data-driven decision-making (DDD) refers to the practice of basing decisions on the analysis of data, rather than purely on intuition. For example, a marketer could select advertisements based purely on her long experience in the field and her eye for what will work. Or, she could base her selection on the analysis of data regarding how consumers react to different ads. She could also use a combination of these approaches. DDD is not an all-or-nothing practice, and different firms engage in DDD to greater or lesser degrees.

The benefits of data-driven decision-making have been demonstrated conclusively. Economist Erik Brynjolfsson and his colleagues from MIT and Penn's Wharton School conducted a study of how DDD affects firm performance (Brynjolfsson, Hitt, & Kim, 2011). They developed a measure of DDD that rates firms as to how strongly they use data to make decisions across the company. They show that statistically, the more data-driven a firm is, the more productive it is—even controlling for a wide range of possible confounding factors. And the differences are not small. One standard deviation higher on the DDD scale is associated with a 4%–6% increase in productivity. DDD also is correlated with higher return on assets, return on equity, asset utilization, and market value, and the relationship seems to be causal.

The sort of decisions we will be interested in in this book mainly fall into two types: (1) decisions for which "discoveries" need to be made within data, and (2) decisions that repeat, especially at massive scale, and so decision-making can benefit from even small increases in decision-making accuracy based on data analysis. The Walmart example above illustrates a type 1 problem: Linda Dillman would like to discover knowledge that will help Walmart prepare for Hurricane Frances's imminent arrival.

In 2012, Walmart's competitor Target was in the news for a data-driven decision-making case of its own, also a type 1 problem (Duhigg, 2012). Like most retailers, Target cares about consumers' shopping habits, what drives them, and what can influence them. Consumers tend to have inertia in their habits and getting them to change is very difficult. Decision makers at Target knew, however, that the arrival of a new baby in a family is one point where people do change their shopping habits significantly. In the Target analyst's words, "As soon as we get them buying diapers from us, they're going to start buying everything else too." Most retailers know this and so they compete with each other trying to sell baby-related products to new parents. Since most birth records are public, retailers obtain information on births and send out special offers to the new parents.

However, Target wanted to get a jump on their competition. They were interested in whether they could *predict* that people *are expecting* a baby. If they could, they would gain an advantage by making offers before their competitors. Using techniques of data science, Target analyzed historical data on customers who *later* were revealed to have been pregnant, and were able to extract information that could predict which consumers were pregnant. For example, pregnant mothers often change their diets, their wardrobes, their vitamin regimens, and so on. These indicators could be extracted from historical data, assembled into predictive models, and then deployed in marketing campaigns. We will discuss predictive models in much detail as we go through the book. For the time being, it is sufficient to understand that a predictive model abstracts away most of the complexity of the world, focusing in on a particular set of indicators that correlate in some way with a quantity of interest (who will churn, or who will purchase, who is pregnant, etc.). Importantly, in both the Walmart and the Target examples, the data analysis was not testing a simple hypothesis. Instead, the data were explored with the hope that something useful would be discovered.[2]

Our churn example illustrates a type 2 DDD problem. MegaTelCo has hundreds of millions of customers, each a candidate for defection. Tens of millions of customers have contracts expiring each month, so each one of them has an increased likelihood of defection in the near future. If we can improve our ability to estimate, for a given customer, how profitable it would be for us to focus on her, we can potentially reap large benefits by applying this ability to the millions of customers in the population. This same logic applies to many of the areas where we have seen the most intense application of data science and data mining: direct marketing, online advertising, credit scoring, financial trading, help-desk management, fraud detection, search ranking, product recommendation, and so on.

The diagram in Figure 1-1 shows data science supporting data-driven decision-making, but also overlapping with data-driven decision-making. This highlights the often overlooked fact that, increasingly, business decisions are being made *automatically* by computer systems. Different industries have adopted automatic decision-making at different rates. The finance and telecommunications industries were early adopters, largely because of their precocious development of data networks and implementation of massive-scale computing, which allowed the aggregation and modeling of data at a large scale, as well as the application of the resultant models to decision-making.

In the 1990s, automated decision-making changed the banking and consumer credit industries dramatically. In the 1990s, banks and telecommunications companies also implemented massive-scale systems for managing data-driven fraud control decisions. As retail systems were increasingly computerized, merchandising decisions were automated. Famous examples include Harrah's casinos' reward programs and the automated recommendations of Amazon and Netflix. Currently we are seeing a revolution in advertising, due in large part to a huge increase in the amount of time consumers are spending online, and the ability online to make (literally) split-second advertising decisions.

Data Processing and "Big Data"

It is important to digress here to address another point. There is a lot to data processing that is not data science—despite the impression one might get from the media. Data engineering and processing are critical to support data science, but they are more general. For example, these days many data processing skills, systems, and technologies often are mistakenly cast as data science. To understand data science and data-driven

2. Target was successful enough that this case raised ethical questions on the deployment of such techniques. Concerns of ethics and privacy are interesting and very important, but we leave their discussion for another time and place.

businesses it is important to understand the differences. Data science needs access to data and it often benefits from sophisticated data engineering that data processing technologies may facilitate, but these technologies are not data science technologies per se. They support data science, as shown in Figure 1-1, but they are useful for much more. Data processing technologies are very important for many data-oriented business tasks that do not involve extracting knowledge or data-driven decision-making, such as efficient transaction processing, modern web system processing, and online advertising campaign management.

"Big data" technologies (such as Hadoop, HBase, and MongoDB) have received considerable media attention recently. *Big data* essentially means datasets that are too large for traditional data processing systems, and therefore require new processing technologies. As with the traditional technologies, big data technologies are used for many tasks, including data engineering. Occasionally, big data technologies are actually used for *implementing* data mining techniques. However, much more often the well-known big data technologies are used for data processing *in support of* the data mining techniques and other data science activities, as represented in Figure 1-1.

Previously, we discussed Brynjolfsson's study demonstrating the benefits of data-driven decision-making. A separate study, conducted by economist Prasanna Tambe of NYU's Stern School, examined the extent to which *big data* technologies seem to help firms (Tambe, 2012). He finds that, after controlling for various possible confounding factors, using big data technologies is associated with significant additional productivity growth. Specifically, one standard deviation higher utilization of big data technologies is associated with 1%–3% higher productivity than the average firm; one standard deviation lower in terms of big data utilization is associated with 1%–3% lower productivity. This leads to potentially very large productivity differences between the firms at the extremes.

From Big Data 1.0 to Big Data 2.0

One way to think about the state of big data technologies is to draw an analogy with the business adoption of Internet technologies. In Web 1.0, businesses busied themselves with getting the basic internet technologies in place, so that they could establish a web presence, build electronic commerce capability, and improve the efficiency of their operations. We can think of ourselves as being in the era of Big Data 1.0. Firms are busying themselves with building the capabilities to process large data, largely in support of their current operations—for example, to improve efficiency.

Once firms had incorporated Web 1.0 technologies thoroughly (and in the process had driven down prices of the underlying technology) they started to look further. They began to ask what the Web could do for them, and how it could improve things they'd always done—and we entered the era of Web 2.0, where new systems and companies began taking advantage of the interactive nature of the Web. The changes brought on by this shift in thinking are pervasive; the most obvious are the incorporation of social-networking components, and the rise of the "voice" of the individual consumer (and citizen).

We should expect a Big Data 2.0 phase to follow Big Data 1.0. Once firms have become capable of processing massive data in a flexible fashion, they should begin asking: "*What can I now do that I couldn't do before, or do better than I could do before?*" This is likely to be the golden era of data science. The principles and techniques we introduce in this book will be applied far more broadly and deeply than they are today.

> It is important to note that in the Web 1.0 era some precocious companies began applying Web 2.0 ideas far ahead of the mainstream. Amazon is a prime example, incorporating the consumer's "voice" early on, in the rating of products, in product reviews (and deeper, in the rating of product reviews). Similarly, we see some companies already applying Big Data 2.0. Amazon again is a company at the forefront, providing data-driven recommendations from massive data. There are other examples as well. Online advertisers must process extremely large volumes of data (billions of ad impressions per day is not unusual) and maintain a very high throughput (real-time bidding systems make decisions in tens of milliseconds). We should look to these and similar industries for hints at advances in big data and data science that subsequently will be adopted by other industries.

Data and Data Science Capability as a Strategic Asset

The prior sections suggest one of the fundamental principles of data science: *data, and the capability to extract useful knowledge from data, should be regarded as key strategic assets.* Too many businesses regard data analytics as pertaining mainly to realizing value from some existing data, and often without careful regard to whether the business has the appropriate analytical talent. Viewing these as assets allows us to think explicitly about the extent to which one should invest in them. Often, we don't have exactly the right data to best make decisions and/or the right talent to best support making decisions from the data. Further, thinking of these as assets should lead us to the realization that they are *complementary*. The best data science team can yield little value without the appropriate data; the right data often cannot substantially improve decisions without suitable data science talent. As with all assets, it is often necessary to make investments. Building a top-notch data science team is a nontrivial undertaking, but can make a huge difference for decision-making. We will discuss strategic considerations involving data science in detail in Chapter 13. Our next case study will introduce the idea that thinking explicitly about how to invest in data assets very often pays off handsomely.

The classic story of little Signet Bank from the 1990s provides a case in point. Previously, in the 1980s, data science had transformed the business of consumer credit. Modeling the probability of default had changed the industry from personal assessment of the likelihood of default to strategies of massive scale and market share, which brought along concomitant economies of scale. It may seem strange now, but at the time, credit cards essentially had uniform pricing, for two reasons: (1) the companies did not have adequate information systems to deal with differential pricing at massive scale, and (2) bank management believed customers would not stand for price discrimination. Around 1990, two strategic visionaries (Richard Fairbanks and Nigel Morris) realized that information technology was powerful enough that they could do more sophisticated predictive modeling—using the sort of techniques that we discuss throughout this book—and offer different terms (nowadays: pricing, credit limits, low-initial-rate balance transfers, cash back, loyalty points, and so on). These two men had no success persuading the big banks to take them on as consultants and let them try. Finally, after running out of big banks, they succeeded in garnering the interest of a small regional Virginia bank: Signet Bank. Signet Bank's management was convinced that modeling profitability, not just default probability, was the right strategy. They knew that a small proportion of customers actually account for *more than* 100% of a bank's profit from credit card operations (because the rest are break-even or money-losing). If they could model profitability, they could make better offers to the best customers and "skim the cream" of the big banks' clientele.

But Signet Bank had one really big problem in implementing this strategy. They did not have the appropriate data to model profitability with the goal of offering different terms to different customers. No one did. Since banks were offering credit with a specific set of terms and a specific default model, they had the data to model profitability (1) for the terms they actually have offered in the past, and (2) for the sort of customer who was actually offered credit (that is, those who were deemed worthy of credit by the existing model).

What could Signet Bank do? They brought into play a fundamental strategy of data science: acquire the necessary data at a cost. Once we view data as a business asset, we should think about whether and how much we are willing to invest. In Signet's case, data could be generated on the profitability of customers given different credit terms by conducting experiments. Different terms were offered at random to different customers. This may seem foolish outside the context of data-analytic thinking: you're likely to lose money! This is true. In this case, losses are the cost of data acquisition. The data-analytic thinker needs to consider whether she expects the data to have sufficient value to justify the investment.

So what happened with Signet Bank? As you might expect, when Signet began randomly offering terms to customers for data acquisition, the number of bad accounts soared. Signet went from an industry-leading "charge-off" rate (2.9% of balances went unpaid) to almost 6% charge-offs. Losses continued for a few years while the data scientists worked to build predictive models from the data, evaluate them, and deploy them to improve profit. Because the firm viewed these losses as investments in data, they persisted despite complaints from stakeholders. Eventually, Signet's credit card operation turned around and became so profitable that it was spun off to separate it from the bank's other operations, which now were overshadowing the consumer credit success.

Fairbanks and Morris became Chairman and CEO and President and COO, and proceeded to apply data science principles throughout the business—not just customer acquisition but retention as well. When a customer calls looking for a better offer, data-driven models calculate the potential profitability of various possible actions (different offers, including sticking with the status quo), and the customer service representative's computer presents the best offers to make.

You may not have heard of little Signet Bank, but if you're reading this book you've probably heard of the spin-off: Capital One. Fairbanks and Morris's new company grew to be one of the largest credit card issuers in the industry with one of the lowest charge-off rates. In 2000, the bank was reported to be carrying out 45,000 of these "scientific tests" as they called them.[3]

Studies giving clear quantitative demonstrations of the value of a data asset are hard to find, primarily because firms are hesitant to divulge results of strategic value. One exception is a study by Martens and Provost (2011) assessing whether data on the specific transactions of a bank's consumers can improve models for deciding what product offers to make. The bank built models from data to decide whom to target with offers for different products. The investigation examined a number of different types of data and their effects on predictive performance. Sociodemographic data provide a substantial ability to model the sort of consumers that are more likely to purchase one product or another. However, sociodemographic data only go so far; after a certain volume of data, no additional advantage is conferred. In contrast, detailed data on customers' individual (anonymized) transactions improve performance substantially over just using sociodemographic data. The relationship is clear and striking and—significantly, for the point here—the predictive performance continues to improve as more data are used, increasing throughout the range investigated by Martens and Provost with no sign of abating. This has an important implication: banks with bigger data assets may have an important strategic advantage over their smaller competitors. If these trends generalize, and the banks are able to apply sophisticated analytics, banks with bigger data assets should be better able to identify the best customers for individual products. The net result will be either increased adoption of the bank's products, decreased cost of customer acquisition, or both.

The idea of data as a strategic asset is certainly not limited to Capital One, nor even to the banking industry. Amazon was able to gather data early on online customers, which has created significant switching costs: consumers find value in the rankings and recommendations that Amazon provides. Amazon therefore can retain customers more

3. You can read more about Capital One's story (Clemons & Thatcher, 1998; McNamee 2001).

easily, and can even charge a premium (Brynjolfsson & Smith, 2000). Harrah's casinos famously invested in gathering and mining data on gamblers, and moved itself from a small player in the casino business in the mid-1990s to the acquisition of Caesar's Entertainment in 2005 to become the world's largest gambling company. The huge valuation of Facebook has been credited to its vast and unique data assets (Sengupta, 2012), including both information about individuals and their likes, as well as information about the structure of the social network. Information about network structure has been shown to be important to predicting and has been shown to be remarkably helpful in building models of who will buy certain products (Hill, Provost, & Volinsky, 2006). It is clear that Facebook has a remarkable data asset; whether they have the right data science strategies to take full advantage of it is an open question.

In the book we will discuss in more detail many of the fundamental concepts behind these success stories, in exploring the principles of data mining and data-analytic thinking.

Data-Analytic Thinking

Analyzing case studies such as the churn problem improves our ability to approach problems "data-analytically." Promoting such a perspective is a primary goal of this book. When faced with a business problem, you should be able to assess whether and how data can improve performance. We will discuss a set of fundamental concepts and principles that facilitate careful thinking. We will develop frameworks to structure the analysis so that it can be done systematically.

As mentioned above, it is important to understand data science even if you never intend to do it yourself, because data analysis is now so critical to business strategy. Businesses increasingly are driven by data analytics, so there is great professional advantage in being able to interact competently with and within such businesses. Understanding the fundamental concepts, and having frameworks for organizing data-analytic thinking not only will allow one to interact competently, but will help to envision opportunities for improving data-driven decision-making, or to see data-oriented competitive threats.

Firms in many traditional industries are exploiting new and existing data resources for competitive advantage. They employ data science teams to bring advanced technologies to bear to increase revenue and to decrease costs. In addition, many new companies are being developed with data mining as a key strategic component. Facebook and Twitter, along with many other "Digital 100" companies (*Business Insider*, 2012), have high valuations due primarily to data assets they are committed to capturing or creating.[4] Increasingly, managers need to oversee analytics teams and analysis projects, marketers

4. Of course, this is not a new phenomenon. Amazon and Google are well-established companies that get tremendous value from their data assets.

have to organize and understand data-driven campaigns, venture capitalists must be able to invest wisely in businesses with substantial data assets, and business strategists must be able to devise plans that exploit data.

As a few examples, if a consultant presents a proposal to mine a data asset to improve your business, you should be able to assess whether the proposal makes sense. If a competitor announces a new data partnership, you should recognize when it may put you at a strategic disadvantage. Or, let's say you take a position with a venture firm and your first project is to assess the potential for investing in an advertising company. The founders present a convincing argument that they will realize significant value from a unique body of data they will collect, and on that basis are arguing for a substantially higher valuation. Is this reasonable? With an understanding of the fundamentals of data science you should be able to devise a few probing questions to determine whether their valuation arguments are plausible.

On a scale less grand, but probably more common, data analytics projects reach into all business units. Employees throughout these units must interact with the data science team. If these employees do not have a fundamental grounding in the principles of data-analytic thinking, they will not really understand what is happening in the business. This lack of understanding is much more damaging in data science projects than in other technical projects, because the data science is supporting improved decision-making. As we will describe in the next chapter, this requires a close interaction between the data scientists and the business people responsible for the decision-making. Firms where the business people do not understand what the data scientists are doing are at a substantial disadvantage, because they waste time and effort or, worse, because they ultimately make wrong decisions.

The need for managers with data-analytic skills

The consulting firm McKinsey and Company estimates that "there will be a shortage of talent necessary for organizations to take advantage of big data. By 2018, the United States alone could face a shortage of 140,000 to 190,000 people with deep analytical skills as well as 1.5 million managers and analysts with the know-how to use the analysis of big data to make effective decisions." (Manyika, 2011). Why 10 times as many managers and analysts than those with deep analytical skills? Surely data scientists aren't so difficult to manage that they need 10 managers! The reason is that a business can get leverage from a data science team for making better decisions in multiple areas of the business. However, as McKinsey is pointing out, the managers in those areas need to understand the fundamentals of data science to effectively get that leverage.

This Book

This book concentrates on the fundamentals of data science and data mining. These are a set of principles, concepts, and techniques that structure thinking and analysis. They allow us to understand data science processes and methods surprisingly deeply, without needing to focus in depth on the large number of specific data mining algorithms.

There are many good books covering data mining algorithms and techniques, from practical guides to mathematical and statistical treatments. This book instead focuses on the fundamental concepts and how they help us to think about problems where data mining may be brought to bear. That doesn't mean that we will ignore the data mining techniques; many algorithms are exactly the embodiment of the basic concepts. But with only a few exceptions we will not concentrate on the deep technical details of how the techniques actually work; we will try to provide just enough detail so that you will understand what the techniques do, and how they are based on the fundamental principles.

Data Mining and Data Science, Revisited

This book devotes a good deal of attention to the extraction of useful (nontrivial, hopefully actionable) patterns or models from large bodies of data (Fayyad, Piatetsky-Shapiro, & Smyth, 1996), and to the fundamental data science principles underlying such data mining. In our churn-prediction example, we would like to *take the data* on prior churn and *extract patterns*, for example patterns of behavior, *that are useful*—that can help us to predict those customers who are more likely to leave in the future, or that can help us to design better services.

The fundamental concepts of data science are drawn from many fields that study data analytics. We introduce these concepts throughout the book, but let's briefly discuss a few now to get the basic flavor. We will elaborate on all of these and more in later chapters.

Fundamental concept: *Extracting useful knowledge from data to solve business problems can be treated systematically by following a process with reasonably well-defined stages.* The Cross Industry Standard Process for Data Mining, abbreviated CRISP-DM (CRISP-DM Project, 2000), is one codification of this process. Keeping such a process in mind provides a framework to structure our thinking about data analytics problems. For example, in actual practice one repeatedly sees analytical "solutions" that are not based on careful analysis of the problem or are not carefully evaluated. Structured thinking about analytics emphasizes these often under-appreciated aspects of supporting decision-making with data. Such structured thinking also contrasts critical points where human creativity is necessary versus points where high-powered analytical tools can be brought to bear.

Fundamental concept: *From a large mass of data, information technology can be used to find informative descriptive attributes of entities of interest.* In our churn example, a customer would be an entity of interest, and each customer might be described by a large number of attributes, such as usage, customer service history, and many other factors. Which of these actually gives us information on the customer's likelihood of leaving the company when her contract expires? How much information? Sometimes this process is referred to roughly as finding variables that "correlate" with churn (we will discuss this notion precisely). A business analyst may be able to hypothesize some and test them, and there are tools to help facilitate this experimentation (see "Other Analytics Techniques and Technologies" on page 35). Alternatively, the analyst could apply information technology to automatically discover informative attributes—essentially doing large-scale automated experimentation. Further, as we will see, this concept can be applied recursively to build models to predict churn based on multiple attributes.

Fundamental concept: *If you look too hard at a set of data, you will find something—but it might not generalize beyond the data you're looking at.* This is referred to as *overfitting* a dataset. Data mining techniques can be very powerful, and the need to detect and avoid overfitting is one of the most important concepts to grasp when applying data mining to real problems. The concept of overfitting and its avoidance permeates data science processes, algorithms, and evaluation methods.

Fundamental concept: *Formulating data mining solutions and evaluating the results involves thinking carefully about the context in which they will be used.* If our goal is the extraction of potentially *useful* knowledge, how can we formulate what is useful? It depends critically on the application in question. For our churn-management example, how exactly are we going to use the patterns extracted from historical data? Should the value of the customer be taken into account in addition to the likelihood of leaving? More generally, does the pattern lead to better decisions than some reasonable alternative? How well would one have done by chance? How well would one do with a smart "default" alternative?

These are just four of the fundamental concepts of data science that we will explore. By the end of the book, we will have discussed a dozen such fundamental concepts in detail, and will have illustrated how they help us to structure data-analytic thinking and to understand data mining techniques and algorithms, as well as data science applications, quite generally.

Chemistry Is Not About Test Tubes: Data Science Versus the Work of the Data Scientist

Before proceeding, we should briefly revisit the engineering side of data science. At the time of this writing, discussions of data science commonly mention not just analytical skills and techniques for understanding data but popular tools used. Definitions of data scientists (and advertisements for positions) specify not just areas of expertise but also specific programming languages and tools. It is common to see job advertisements mentioning data mining techniques (e.g., random forests, support vector machines), specific application areas (recommendation systems, ad placement optimization), alongside popular software tools for processing big data (Hadoop, MongoDB). There is often little distinction between the science and the technology for dealing with large datasets.

We must point out that data science, like computer science, is a young field. The particular concerns of data science are fairly new and general principles are just beginning to emerge. The state of data science may be likened to that of chemistry in the mid-19th century, when theories and general principles were being formulated and the field was largely experimental. Every good chemist had to be a competent lab technician. Similarly, it is hard to imagine a working data scientist who is not proficient with certain sorts of software tools.

Having said this, this book focuses on the science and not on the technology. You will not find instructions here on how best to run massive data mining jobs on Hadoop clusters, or even what Hadoop is or why you might want to learn about it.[5] We focus here on the general principles of data science that have emerged. In 10 years' time the predominant technologies will likely have changed or advanced enough that a discussion here would be obsolete, while the general principles are the same as they were 20 years ago, and likely will change little over the coming decades.

Summary

This book is about the extraction of useful information and knowledge from large volumes of data, in order to improve business decision-making. As the massive collection of data has spread through just about every industry sector and business unit, so have the opportunities for mining the data. Underlying the extensive body of techniques for mining data is a much smaller set of fundamental concepts comprising *data science*. These concepts are general and encapsulate much of the essence of data mining and business analytics.

Success in today's data-oriented business environment requires being able to think about how these fundamental concepts apply to particular business problems—to think data-analytically. For example, in this chapter we discussed the principle that data should be thought of as a business asset, and once we are thinking in this direction we start to ask whether (and how much) we should invest in data. Thus, an understanding of these fundamental concepts is important not only for data scientists themselves, but for any-

5. OK: Hadoop is a widely used open source architecture for doing highly parallelizable computations. It is one of the current "big data" technologies for processing massive datasets that exceed the capacity of relational database systems. Hadoop is based on the MapReduce parallel processing framework introduced by Google.

one working with data scientists, employing data scientists, investing in data-heavy ventures, or directing the application of analytics in an organization.

Thinking data-analytically is aided by conceptual frameworks discussed throughout the book. For example, the automated extraction of patterns from data is a process with well-defined stages, which are the subject of the next chapter. Understanding the process and the stages helps to structure our data-analytic thinking, and to make it more systematic and therefore less prone to errors and omissions.

There is convincing evidence that data-driven decision-making and big data technologies substantially improve business performance. Data science supports data-driven decision-making—and sometimes conducts such decision-making automatically—and depends upon technologies for "big data" storage and engineering, but its principles are separate. The data science principles we discuss in this book also differ from, and are complementary to, other important technologies, such as statistical hypothesis testing and database querying (which have their own books and classes). The next chapter describes some of these differences in more detail.

CHAPTER 2

Business Problems and Data Science Solutions

Fundamental concepts: *A set of canonical data mining tasks; The data mining process; Supervised versus unsupervised data mining.*

An important principle of data science is that data mining is a *process* with fairly well-understood stages. Some involve the application of information technology, such as the automated discovery and evaluation of patterns from data, while others mostly require an analyst's creativity, business knowledge, and common sense. Understanding the whole process helps to structure data mining projects, so they are closer to systematic analyses rather than heroic endeavors driven by chance and individual acumen.

Since the data mining process breaks up the overall task of finding patterns from data into a set of well-defined subtasks, it is also useful for structuring discussions about data science. In this book, we will use the process as an overarching framework for our discussion. This chapter introduces the data mining process, but first we provide additional context by discussing common types of data mining tasks. Introducing these allows us to be more concrete when presenting the overall process, as well as when introducing other concepts in subsequent chapters.

We close the chapter by discussing a set of important business analytics subjects that are not the focus of this book (but for which there are many other helpful books), such as databases, data warehousing, and basic statistics.

From Business Problems to Data Mining Tasks

Each data-driven business decision-making problem is unique, comprising its own combination of goals, desires, constraints, and even personalities. As with much engineering, though, there are sets of common tasks that underlie the business problems. In collaboration with business stakeholders, data scientists decompose a business problem into subtasks. The solutions to the subtasks can then be composed to solve the overall problem. Some of these subtasks are unique to the particular business problem, but others are common data mining tasks. For example, our telecommunications churn problem is unique to MegaTelCo: there are specifics of the problem that are different from churn problems of any other telecommunications firm. However, a subtask that will likely be part of the solution to any churn problem is to estimate from historical data the probability of a customer terminating her contract shortly after it has expired. Once the idiosyncratic MegaTelCo data have been assembled into a particular format (described in the next chapter), this probability estimation fits the mold of one very common data mining task. We know a lot about solving the common data mining tasks, both scientifically and practically. In later chapters, we also will provide data science frameworks to help with the decomposition of business problems and with the re-composition of the solutions to the subtasks.

A critical skill in data science is the ability to decompose a data-analytics problem into pieces such that each piece matches a known task for which tools are available. Recognizing familiar problems and their solutions avoids wasting time and resources reinventing the wheel. It also allows people to focus attention on more interesting parts of the process that require human involvement—parts that have not been automated, so human creativity and intelligence must come into play.

Despite the large number of specific data mining algorithms developed over the years, there are only a handful of fundamentally different types of tasks these algorithms address. It is worth defining these tasks clearly. The next several chapters will use the first two (classification and regression) to illustrate several fundamental concepts. In what follows, the term "an individual" will refer to an entity about which we have data, such as a customer or a consumer, or it could be an inanimate entity such as a business. We will make this notion more precise in Chapter 3. In many business analytics projects, we want to find "correlations" between a particular variable describing an individual and other variables. For example, in historical data we may know which customers left the company after their contracts expired. We may want to find out which other variables correlate with a customer leaving in the near future. Finding such correlations are the most basic examples of classification and regression tasks.

1. *Classification* and class *probability estimation* attempt to predict, for each individual in a population, which of a (small) set of classes this individual belongs to. Usually the classes are mutually exclusive. An example classification question would be: "Among all the customers of MegaTelCo, which are likely to respond to a given offer?" In this example the two classes could be called `will respond` and `will not respond`.

For a classification task, a data mining procedure produces a model that, given a new individual, determines which class that individual belongs to. A closely related task is *scoring* or class *probability estimation*. A scoring model applied to an individual produces, instead of a class prediction, a score representing the probability (or some other quantification of likelihood) that that individual belongs to each class. In our customer response scenario, a scoring model would be able to evaluate each individual customer and produce a score of how likely each is to respond to the offer. Classification and scoring are very closely related; as we shall see, a model that can do one can usually be modified to do the other.

2. *Regression* ("value estimation") attempts to estimate or predict, for each individual, the numerical value of some variable for that individual. An example regression question would be: "How much will a given customer use the service?" The property (variable) to be predicted here is *service usage*, and a model could be generated by looking at other, similar individuals in the population and their historical usage. A regression procedure produces a model that, given an individual, estimates the value of the particular variable specific to that individual.

   Regression is related to classification, but the two are different. Informally, classification predicts *whether* something will happen, whereas regression predicts *how much* something will happen. The difference will become clearer as the book progresses.

3. *Similarity matching* attempts to *identify* similar individuals based on data known about them. Similarity matching can be used directly to find similar entities. For example, IBM is interested in finding companies similar to their best business customers, in order to focus their sales force on the best opportunities. They use similarity matching based on "firmographic" data describing characteristics of the companies. Similarity matching is the basis for one of the most popular methods for making product recommendations (finding people who are similar to you in terms of the products they have liked or have purchased). Similarity measures underlie certain solutions to other data mining tasks, such as classification, regression, and clustering. We discuss similarity and its uses at length in Chapter 6.

4. *Clustering* attempts to *group* individuals in a population together by their similarity, but not driven by any specific purpose. An example clustering question would be: "Do our customers form natural groups or segments?" Clustering is useful in preliminary domain exploration to see which natural groups exist because these groups in turn may suggest other data mining tasks or approaches. Clustering also is used as input to decision-making processes focusing on questions such as: *What products should we offer or develop*? *How should our customer care teams (or sales teams) be structured*? We discuss clustering in depth in Chapter 6.

5. *Co-occurrence grouping* (also known as frequent itemset mining, association rule discovery, and market-basket analysis) attempts to find *associations* between entities based on transactions involving them. An example co-occurrence question

would be: *What items are commonly purchased together*? While clustering looks at similarity between objects based on the objects' attributes, co-occurrence grouping considers similarity of objects based on their appearing together in transactions. For example, analyzing purchase records from a supermarket may uncover that ground meat is purchased together with hot sauce much more frequently than we might expect. Deciding how to act upon this discovery might require some creativity, but it could suggest a special promotion, product display, or combination offer. Co-occurrence of products in purchases is a common type of grouping known as market-basket analysis. Some *recommendation* systems also perform a type of affinity grouping by finding, for example, pairs of books that are purchased frequently by the same people ("people who bought X also bought Y").

The result of co-occurrence grouping is a description of items that occur together. These descriptions usually include statistics on the frequency of the co-occurrence and an estimate of how surprising it is.

6. *Profiling* (also known as behavior description) attempts to characterize the typical behavior of an individual, group, or population. An example profiling question would be: "What is the typical cell phone usage of this customer segment?" Behavior may not have a simple description; profiling cell phone usage might require a complex description of night and weekend airtime averages, international usage, roaming charges, text minutes, and so on. Behavior can be described generally over an entire population, or down to the level of small groups or even individuals.

   Profiling is often used to establish behavioral norms for anomaly detection applications such as fraud detection and monitoring for intrusions to computer systems (such as someone breaking into your iTunes account). For example, if we know what kind of purchases a person typically makes on a credit card, we can determine whether a new charge on the card fits that profile or not. We can use the degree of mismatch as a suspicion score and issue an alarm if it is too high.

7. *Link prediction* attempts to predict connections between data items, usually by suggesting that a link should exist, and possibly also estimating the strength of the link. Link prediction is common in social networking systems: "Since you and Karen share 10 friends, maybe you'd like to be Karen's friend?" Link prediction can also estimate the strength of a link. For example, for recommending movies to customers one can think of a graph between customers and the movies they've watched or rated. Within the graph, we search for links that do *not* exist between customers and movies, but that we predict should exist and should be strong. These links form the basis for recommendations.

8. *Data reduction* attempts to take a large set of data and replace it with a smaller set of data that contains much of the important information in the larger set. The smaller dataset may be easier to deal with or to process. Moreover, the smaller dataset may better reveal the information. For example, a massive dataset on consumer movie-viewing preferences may be reduced to a much smaller dataset re-

vealing the consumer taste preferences that are latent in the viewing data (for example, viewer genre preferences). Data reduction usually involves loss of information. What is important is the trade-off for improved insight.

9. *Causal modeling* attempts to help us understand what events or actions actually *influence* others. For example, consider that we use predictive modeling to target advertisements to consumers, and we observe that indeed the targeted consumers purchase at a higher rate subsequent to having been targeted. Was this because the advertisements influenced the consumers to purchase? Or did the predictive models simply do a good job of identifying those consumers who would have purchased anyway? Techniques for causal modeling include those involving a substantial investment in data, such as randomized controlled experiments (e.g., so-called "A/B tests"), as well as sophisticated methods for drawing causal conclusions from observational data. Both experimental and observational methods for causal modeling generally can be viewed as "counterfactual" analysis: they attempt to understand what would be the difference between the situations—which cannot both happen —where the "treatment" event (e.g., showing an advertisement to a particular individual) were to happen, and were not to happen.

   In all cases, a careful data scientist should always include with a causal conclusion the exact assumptions that must be made in order for the causal conclusion to hold (there *always* are such assumptions—always ask). When undertaking causal modeling, a business needs to weigh the trade-off of increasing investment to reduce the assumptions made, versus deciding that the conclusions are good enough given the assumptions. Even in the most careful randomized, controlled experimentation, assumptions are made that could render the causal conclusions invalid. The discovery of the "placebo effect" in medicine illustrates a notorious situation where an assumption was overlooked in carefully designed randomized experimentation.

Discussing all of these tasks in detail would fill multiple books. In this book, we present a collection of the most fundamental data science principles—principles that together underlie all of these types of tasks. We will illustrate the principles mainly using classification, regression, similarity matching, and clustering, and will discuss others when they provide important illustrations of the fundamental principles (toward the end of the book).

Consider which of these types of tasks might fit our churn-prediction problem. Often, practitioners formulate churn prediction as a problem of finding *segments* of customers who are more or less likely to leave. This segmentation problem sounds like a classification problem, or possibly clustering, or even regression. To decide the best formulation, we first need to introduce some important distinctions.

Supervised Versus Unsupervised Methods

Consider two similar questions we might ask about a customer population. The first is: "Do our customers naturally fall into different groups?" Here no specific purpose or *target* has been specified for the grouping. When there is no such target, the data mining problem is referred to as *unsupervised*. Contrast this with a slightly different question: "Can we find groups of customers who have particularly high likelihoods of canceling their service soon after their contracts expire?" Here there is a specific target defined: will a customer leave when her contract expires? In this case, segmentation is being done for a specific reason: to take action based on likelihood of churn. This is called a *supervised* data mining problem.

A note on the terms: Supervised and unsupervised learning

The terms *supervised* and *unsupervised* were inherited from the field of machine learning. Metaphorically, a teacher "supervises" the learner by carefully providing target information along with a set of examples. An unsupervised learning task might involve the same set of examples but would not include the target information. The learner would be given no information about the purpose of the learning, but would be left to form its own conclusions about what the examples have in common.

The difference between these questions is subtle but important. If a specific target can be provided, the problem can be phrased as a supervised one. Supervised tasks require different techniques than unsupervised tasks do, and the results often are much more useful. A supervised technique is given a specific purpose for the grouping—predicting the target. Clustering, an unsupervised task, produces groupings based on similarities, but there is no guarantee that these similarities are meaningful or will be useful for any particular purpose.

Technically, another condition must be met for supervised data mining: there must be *data* on the target. It is not enough that the target information exist in principle; it must also exist in the data. For example, it might be useful to know whether a given customer will stay for at least six months, but if in historical data this retention information is missing or incomplete (if, say, the data are only retained for two months) the target values cannot be provided. Acquiring data on the target often is a key data science investment. The value for the target variable for an individual is often called the individual's *label*, emphasizing that often (not always) one must incur expense to actively label the data.

Classification, regression, and causal modeling generally are solved with supervised methods. Similarity matching, link prediction, and data reduction could be either. Clustering, co-occurrence grouping, and profiling generally are unsupervised. The fundamental principles of data mining that we will present underlie all these types of technique.

Two main subclasses of *supervised* data mining, classification and regression, are distinguished by the type of target. Regression involves a numeric target while classification involves a categorical (often binary) target. Consider these similar questions we might address with supervised data mining:

"Will this customer purchase service S1 if given incentive I?"
: This is a classification problem because it has a binary target (the customer either purchases or does not).

"Which service package (S1, S2, or none) will a customer likely purchase if given incentive I?"
: This is also a classification problem, with a three-valued target.

"How much will this customer use the service?"
: This is a regression problem because it has a numeric target. The target variable is the amount of usage (actual or predicted) per customer.

There are subtleties among these questions that should be brought out. For business applications we often want a numerical *prediction* over a categorical target. In the churn example, a basic yes/no prediction of whether a customer is likely to continue to subscribe to the service may not be sufficient; we want to model the *probability* that the customer will continue. This is still considered classification modeling rather than regression because the underlying target is categorical. Where necessary for clarity, this is called "class probability estimation."

A vital part in the early stages of the data mining process is (i) to decide whether the line of attack will be supervised or unsupervised, and (ii) if supervised, to produce a precise definition of a target variable. This variable must be a specific quantity that will be the focus of the data mining (and for which we can obtain values for some example data). We will return to this in Chapter 3.

Data Mining and Its Results

There is another important distinction pertaining to mining data: the difference between (1) mining the data to find patterns and build models, and (2) *using* the results of data mining. Students often confuse these two processes when studying data science, and managers sometimes confuse them when discussing business analytics. The use of data mining results should influence and inform the data mining process itself, but the two should be kept distinct.

In our churn example, consider the deployment scenario in which the results will be used. We want to use the model to predict which of our customers will leave. Specifically, assume that data mining has created a class probability estimation model *M*. Given each

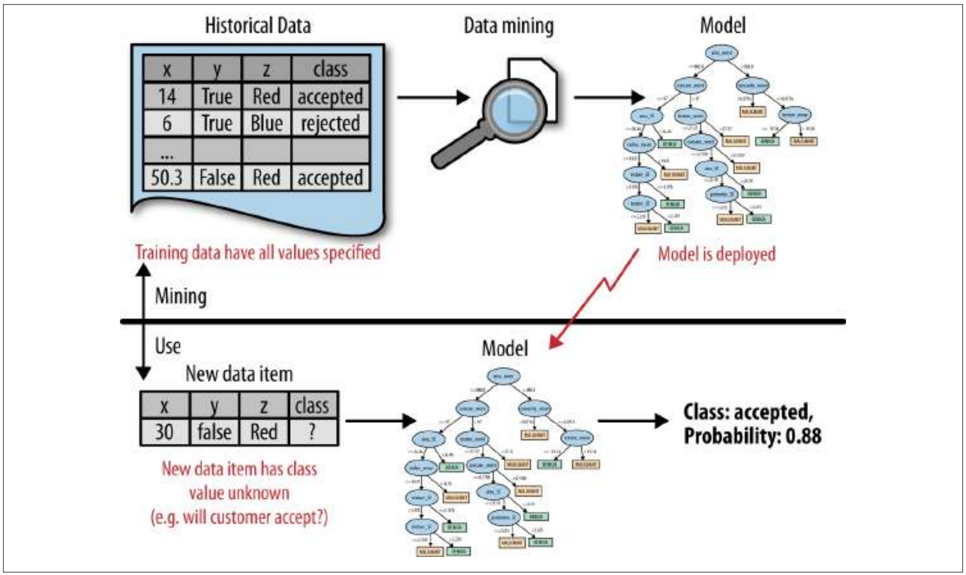

Historical Data
x | y | z | class
14 | True | Red | accepted
6 | True | Blue | rejected
...
50.3 | False | Red | accepted
Data mining
Model
Training data have all values specified
Model is deployed
Mining
Use
Model
New data item
x | y | z | class
30 | false | Red | ?
New data item has class value unknown (e.g. will customer accept?)
Class: accepted, Probability: 0.88

Figure 2-1. Data mining versus the use of data mining results. The upper half of the figure illustrates the mining of historical data to produce a model. Importantly, the historical data have the target ("class") value specified. The bottom half shows the result of the data mining in use, where the model is applied to new data for which we do not know the class value. The model predicts both the class value and the probability that the class variable will take on that value.

existing customer, described using a set of characteristics, *M* takes these characteristics as input and produces a score or probability estimate of attrition. This is the *use* of the results of data mining. The data mining produces the model *M* from some other, often historical, data.

Figure 2-1 illustrates these two phases. Data mining produces the probability estimation model, as shown in the top half of the figure. In the use phase (bottom half), the model is applied to a new, unseen case and it generates a probability estimate for it.

The Data Mining Process

Data mining is a craft. It involves the application of a substantial amount of science and technology, but the proper application still involves art as well. But as with many mature crafts, there is a well-understood process that places a structure on the problem, allowing reasonable consistency, repeatability, and objectiveness. A useful codification of the data mining process is given by the Cross Industry Standard Process for Data Mining (CRISP-DM; Shearer, 2000), illustrated in Figure 2-2.[1]

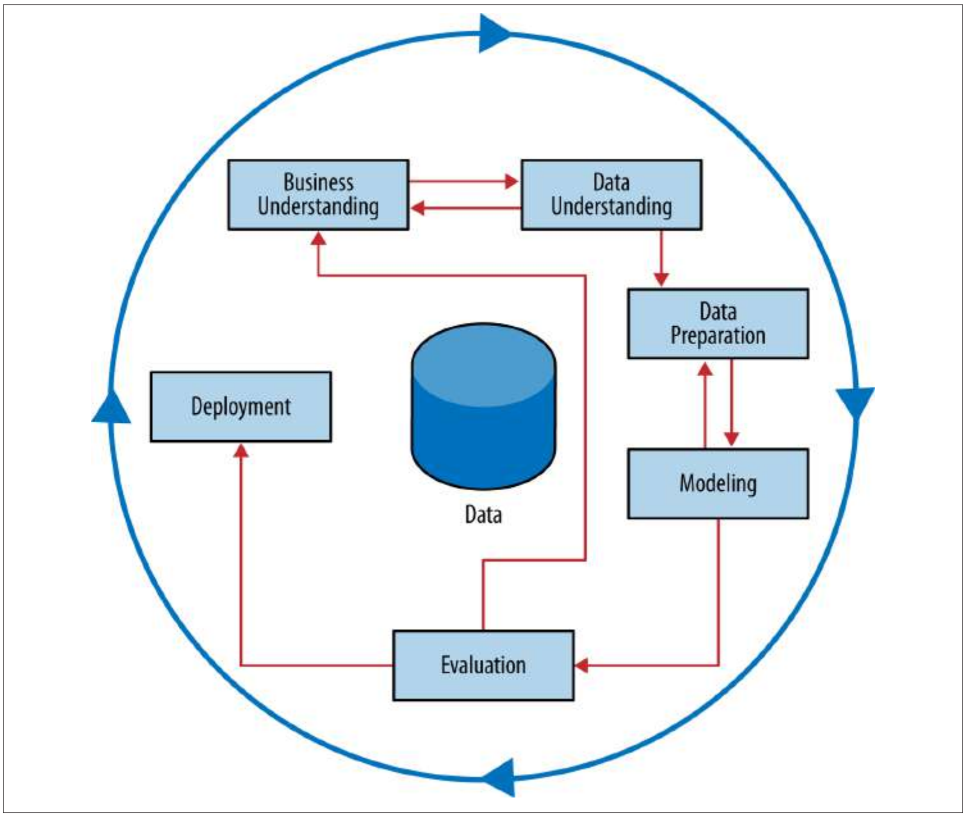


Figure 2-2. The CRISP data mining process.

This process diagram makes explicit the fact that iteration is the rule rather than the exception. Going through the process once without having solved the problem is, generally speaking, not a failure. Often the entire process is an exploration of the data, and after the first iteration the data science team knows much more. The next iteration can be much more well-informed. Let's now discuss the steps in detail.

Business Understanding

Initially, it is vital to understand the problem to be solved. This may seem obvious, but business projects seldom come pre-packaged as clear and unambiguous data mining

1. See also the Wikipedia page on the CRISP-DM process model.

problems. Often recasting the problem and designing a solution is an iterative process of discovery. The diagram shown in Figure 2-2 represents this as cycles within a cycle, rather than as a simple linear process. The initial formulation may not be complete or optimal so multiple iterations may be necessary for an acceptable solution formulation to appear.

The Business Understanding stage represents a part of the craft where the analysts' creativity plays a large role. Data science has some things to say, as we will describe, but often the key to a great success is a creative problem formulation by some analyst regarding how to cast the business problem as one or more data science problems. High-level knowledge of the fundamentals helps creative business analysts see novel formulations.

We have a set of powerful tools to solve particular data mining problems: the basic data mining tasks discussed in "From Business Problems to Data Mining Tasks" on page 19. Typically, the early stages of the endeavor involve designing a solution that takes advantage of these tools. This can mean structuring (engineering) the problem such that one or more subproblems involve building models for classification, regression, probability estimation, and so on.

In this first stage, *the design team should think carefully about the use scenario*. This itself is one of the most important concepts of data science, to which we have devoted two entire chapters (Chapter 7 and Chapter 11). What exactly do we want to do? How exactly would we do it? What parts of this use scenario constitute possible data mining models? In discussing this in more detail, we will begin with a simplified view of the use scenario, but as we go forward we will loop back and realize that often the use scenario must be adjusted to better reflect the actual business need. We will present conceptual tools to help our thinking here, for example framing a business problem in terms of expected value can allow us to systematically decompose it into data mining tasks.

Data Understanding

If solving the business problem is the goal, the data comprise the available raw material from which the solution will be built. It is important to understand the strengths and limitations of the data because rarely is there an exact match with the problem. Historical data often are collected for purposes unrelated to the current business problem, or for no explicit purpose at all. A customer database, a transaction database, and a marketing response database contain different information, may cover different intersecting populations, and may have varying degrees of reliability.

It is also common for the *costs* of data to vary. Some data will be available virtually for free while others will require effort to obtain. Some data may be purchased. Still other data simply won't exist and will require entire ancillary projects to arrange their collection. A critical part of the data understanding phase is estimating the costs and benefits of each data source and deciding whether further investment is merited. Even after all datasets are acquired, collating them may require additional effort. For example, customer records and product identifiers are notoriously variable and noisy. Cleaning and matching customer records to ensure only one record per customer is itself a complicated analytics problem (Hernández & Stolfo, 1995; Elmagarmid, Ipeirotis, & Verykios, 2007).

As data understanding progresses, solution paths may change direction in response, and team efforts may even fork. Fraud detection provides an illustration of this. Data mining has been used extensively for fraud detection, and many fraud detection problems involve classic supervised data mining tasks. Consider the task of catching credit card fraud. Charges show up on each customer's account, so fraudulent charges are usually caught—if not initially by the company, then later by the customer when account activity is reviewed. We can assume that nearly all fraud is identified and reliably labeled, since the legitimate customer and the person perpetrating the fraud are different people and have opposite goals. Thus credit card transactions have reliable labels (*fraud* and *legitimate*) that may serve as targets for a supervised technique.

Now consider the related problem of catching Medicare fraud. This is a huge problem in the United States costing billions of dollars annually. Though this may seem like a conventional fraud detection problem, as we consider the relationship of the business problem to the data, we realize that the problem is significantly different. The perpetrators of fraud—medical providers who submit false claims, and sometimes their patients—are also legitimate service providers and users of the billing system. Those who commit fraud are a subset of the legitimate users; there is no separate disinterested party who will declare exactly what the "correct" charges should be. Consequently the Medicare billing data have no reliable target variable indicating fraud, and a supervised learning approach that could work for credit card fraud is not applicable. Such a problem usually requires unsupervised approaches such as profiling, clustering, anomaly detection, and co-occurrence grouping.

The fact that both of these are fraud detection problems is a superficial similarity that is actually misleading. In data understanding we need to dig beneath the surface to uncover the structure of the business problem and the data that are available, and then match them to one or more data mining tasks for which we may have substantial science and technology to apply. It is not unusual for a business problem to contain several data mining tasks, often of different types, and combining their solutions will be necessary (see Chapter 11).

Data Preparation

The analytic technologies that we can bring to bear are powerful but they impose certain requirements on the data they use. They often require data to be in a form different from how the data are provided naturally, and some conversion will be necessary.

Therefore a data preparation phase often proceeds along with data understanding, in which the data are manipulated and converted into forms that yield better results.

Typical examples of data preparation are converting data to tabular format, removing or inferring missing values, and converting data to different types. Some data mining techniques are designed for symbolic and categorical data, while others handle only numeric values. In addition, numerical values must often be normalized or scaled so that they are comparable. Standard techniques and rules of thumb are available for doing such conversions. Chapter 3 discusses the most typical format for mining data in some detail.

In general, though, this book will not focus on data preparation techniques, which could be the topic of a book by themselves (Pyle, 1999). We will define basic data formats in following chapters, and will only be concerned with data preparation details when they shed light on some fundamental principle of data science or are necessary to present a concrete example.

More generally, data scientists may spend considerable time early in the process defining the variables used later in the process. This is one of the main points at which human creativity, common sense, and business knowledge come into play. Often the quality of the data mining solution rests on how well the analysts structure the problems and craft the variables (and sometimes it can be surprisingly hard for them to admit it).

One very general and important concern during data preparation is to beware of "leaks" (Kaufman et al. 2012). A leak is a situation where a variable collected in historical data gives information on the target variable—information that appears in historical data but is not actually available when the decision has to be made. As an example, when predicting whether at a particular point in time a website visitor would end her session or continue surfing to another page, the variable "total number of webpages visited in the session" is predictive. However, the total number of webpages visited in the session would not be known until after the session was over (Kohavi et al., 2000)—at which point one would know the value for the target variable! As another illustrative example, consider predicting whether a customer *will be* a "big spender"; knowing the categories of the items purchased (or worse, the amount of tax paid) are very predictive, but are not known at decision-making time (Kohavi & Parekh, 2003). Leakage must be considered carefully during data preparation, because data preparation typically is performed after the fact—from historical data. We present a more detailed example of a real leak that was challenging to find in Chapter 14.

Modeling

Modeling is the subject of the next several chapters and we will not dwell on it here, except to say that the output of modeling is some sort of model or pattern capturing regularities in the data.

The modeling stage is the primary place where data mining techniques are applied to the data. It is important to have some understanding of the fundamental ideas of data mining, including the sorts of techniques and algorithms that exist, because this is the part of the craft where the most science and technology can be brought to bear.

Evaluation

The purpose of the evaluation stage is to assess the data mining results rigorously and to gain confidence that they are valid and reliable before moving on. If we look hard enough at any dataset we will find patterns, but they may not survive careful scrutiny. We would like to have confidence that the models and patterns extracted from the data are true regularities and not just idiosyncrasies or sample anomalies. It is possible to deploy results immediately after data mining but this is inadvisable; it is usually far easier, cheaper, quicker, and safer to test a model first in a controlled laboratory setting.

Equally important, the evaluation stage also serves to help ensure that the model satisfies the original business goals. Recall that the primary goal of data science for business is to support decision making, and that we started the process by focusing on the business problem we would like to solve. Usually a data mining solution is only a piece of the larger solution, and it needs to be evaluated as such. Further, even if a model passes strict evaluation tests in "in the lab," there may be external considerations that make it impractical. For example, a common flaw with detection solutions (such as fraud detection, spam detection, and intrusion monitoring) is that they produce too many false alarms. A model may be extremely accurate (> 99%) by laboratory standards, but evaluation in the actual business context may reveal that it still produces too many false alarms to be economically feasible. (How much would it cost to provide the staff to deal with all those false alarms? What would be the cost in customer dissatisfaction?)

Evaluating the results of data mining includes both quantitative and qualitative assessments. Various stakeholders have interests in the business decision-making that will be accomplished or supported by the resultant models. In many cases, these stakeholders need to "sign off" on the deployment of the models, and in order to do so need to be satisfied by the quality of the model's decisions. What that means varies from application to application, but often stakeholders are looking to see whether the model is going to do more good than harm, and especially that the model is unlikely to make catastrophic mistakes.[2] To facilitate such qualitative assessment, the data scientist must think about the *comprehensibility* of the model to stakeholders (not just to the data scientists). And if the model itself is not comprehensible (e.g., maybe the model is a very complex mathematical formula), how can the data scientists work to make the behavior of the model be comprehensible.

Finally, a comprehensive evaluation framework is important because getting detailed information on the performance of a deployed model may be difficult or impossible. Often there is only limited access to the deployment environment so making a comprehensive evaluation "in production" is difficult. Deployed systems typically contain many "moving parts," and assessing the contribution of a single part is difficult. Firms with sophisticated data science teams wisely build testbed environments that mirror production data as closely as possible, in order to get the most realistic evaluations before taking the risk of deployment.

Nonetheless, in some cases we may want to extend evaluation into the development environment, for example by instrumenting a live system to be able to conduct randomized experiments. In our churn example, if we have decided from laboratory tests that a data mined model will give us better churn reduction, we may want to move on to an "in vivo" evaluation, in which a live system randomly applies the model to some customers while keeping other customers as a control group (recall our discussion of causal modeling from Chapter 1). Such experiments must be designed carefully, and the technical details are beyond the scope of this book. The interested reader could start with the lessons-learned articles by Ron Kohavi and his coauthors (Kohavi et al., 2007, 2009, 2012). We may also want to instrument deployed systems for evaluations to make sure that the world is not changing to the detriment of the model's decision-making. For example, behavior can change—in some cases, like fraud or spam, in direct response to the deployment of models. Additionally, the output of the model is critically dependent on the input data; input data can change in format and in substance, often without any alerting of the data science team. Raeder et al. (2012) present a detailed discussion of system design to help deal with these and other related evaluation-in-deployment issues.

Deployment

In deployment the results of data mining—and increasingly the data mining techniques themselves—are put into real use in order to realize some return on investment. The clearest cases of deployment involve implementing a predictive model in some information system or business process. In our churn example, a model for predicting the likelihood of churn could be integrated with the business process for churn management

2. For example, in one data mining project a model was created to diagnose problems in local phone networks, and to dispatch technicians to the likely site of the problem. Before deployment, a team of phone company stakeholders requested that the model be tweaked so that exceptions were made for hospitals.

—for example, by sending special offers to customers who are predicted to be particularly at risk. (We will discuss this in increasing detail as the book proceeds.) A new fraud detection model may be built into a workforce management information system, to monitor accounts and create "cases" for fraud analysts to examine.

Increasingly, the data mining techniques themselves are deployed. For example, for targeting online advertisements, systems are deployed that automatically build (and test) models in production when a new advertising campaign is presented. Two main reasons for deploying the data mining system itself rather than the models produced by a data mining system are (i) the world may change faster than the data science team can adapt, as with fraud and intrusion detection, and (ii) a business has too many modeling tasks for their data science team to manually curate each model individually. In these cases, it may be best to deploy the data mining phase into production. In doing so, it is critical to instrument the process to alert the data science team of any seeming anomalies and to provide fail-safe operation (Raeder et al., 2012).

Deployment can also be much less "technical." In a celebrated case, data mining discovered a set of rules that could help to quickly diagnose and fix a common error in industrial printing. The deployment succeeded simply by taping a sheet of paper containing the rules to the side of the printers (Evans & Fisher, 2002). Deployment can also be much more subtle, such as a change to data acquisition procedures, or a change to strategy, marketing, or operations resulting from insight gained from mining the data.

Deploying a model into a production system typically requires that the model be recoded for the production environment, usually for greater speed or compatibility with an existing system. This may incur substantial expense and investment. In many cases, the data science team is responsible for producing a working prototype, along with its evaluation. These are passed to a development team.

Practically speaking, there are risks with "over the wall" transfers from data science to development. It may be helpful to remember the maxim: "Your model is not what the data scientists design, it's what the engineers build." From a management perspective, it is advisable to have members of the development team involved early on in the data science project. They can begin as advisors, providing critical insight to the data science team. Increasingly in practice, these particular developers are "data science engineers"—software engineers who have particular expertise both in the production systems and in data science. These developers gradually assume more responsibility as the project matures. At some point the developers will take the lead and assume ownership of the product. Generally, the data scientists should still remain involved in the project into final deployment, as advisors or as developers depending on their skills.

Regardless of whether deployment is successful, the process often returns to the Business Understanding phase. The process of mining data produces a great deal of insight into the business problem and the difficulties of its solution. A second iteration can yield an improved solution. Just the experience of thinking about the business, the data, and the performance goals often leads to new ideas for improving business performance, and even new lines of business or new ventures.

Note that it is not necessary to fail in deployment to start the cycle again. The Evaluation stage may reveal that results are not good enough to deploy, and we need to adjust the problem definition or get different data. This is represented by the "shortcut" link from Evaluation back to Business Understanding in the process diagram. In practice, there should be shortcuts back from each stage to each prior one because the process always retains some exploratory aspects, and a project should be flexible enough to revisit prior steps based on discoveries made.[3]

Implications for Managing the Data Science Team

It is tempting—but usually a mistake—to view the data mining process as a software development cycle. Indeed, data mining projects are often treated and managed as engineering projects, which is understandable when they are initiated by software departments, with data generated by a large software system and analytics results fed back into it. Managers are usually familiar with software technologies and are comfortable managing software projects. Milestones can be agreed upon and success is usually unambiguous. Software managers might look at the CRISP data mining cycle (Figure 2-2) and think it looks comfortably similar to a software development cycle, so they should be right at home managing an analytics project the same way.

This can be a mistake because data mining is an exploratory undertaking closer to research and development than it is to engineering. The CRISP cycle is based around exploration; it iterates on *approaches* and *strategy* rather than on software designs. Outcomes are far less certain, and the results of a given step may change the fundamental understanding of the problem. Engineering a data mining solution directly for deployment can be an expensive premature commitment. Instead, analytics projects should prepare to invest in information to reduce uncertainty in various ways. Small invest-

3. Software professionals may recognize the similarity to the philosophy of "Fail faster to succeed sooner" (Muoio, 1997).

ments can be made via pilot studies and throwaway prototypes. Data scientists should review the literature to see what else has been done and how it has worked. On a larger scale, a team can invest substantially in building experimental testbeds to allow extensive agile experimentation. If you're a software manager, this will look more like research and exploration than you're used to, and maybe more than you're comfortable with.

Software skills versus analytics skills

Although data mining involves software, it also requires skills that may not be common among programmers. In software engineering, the ability to write efficient, high-quality code from requirements may be paramount. Team members may be evaluated using software metrics such as the amount of code written or number of bug tickets closed. In analytics, it's more important for individuals to be able to formulate problems well, to prototype solutions quickly, to make reasonable assumptions in the face of ill-structured problems, to design experiments that represent good investments, and to analyze results. In building a data science team, these qualities, rather than traditional software engineering expertise, are skills that should be sought.

Other Analytics Techniques and Technologies

Business analytics involves the application of various technologies to the analysis of data. Many of these go beyond this book's focus on data-analytic thinking and the principles of extracting useful patterns from data. Nonetheless, it is important to be acquainted with these related techniques, to understand what their goals are, what role they play, and when it may be beneficial to consult experts in them.

To this end, we present six groups of related analytic techniques. Where appropriate we draw comparisons and contrasts with data mining. The main difference is that data mining focuses on the *automated* search for *knowledge*, *patterns*, or *regularities* from data.[4] An important skill for a business analyst is to be able to recognize what sort of analytic technique is appropriate for addressing a particular problem.

Statistics

The term "statistics" has two different uses in business analytics. First, it is used as a catchall term for the computation of particular numeric values of interest from data (e.g., "We need to gather some statistics on our customers' usage to determine what's going wrong here.") These values often include sums, averages, rates, and so on. Let's

4. It is important to keep in mind that it is rare for the discovery to be completely automated. The important factor is that data mining automates at least partially the search and discovery process, rather than providing technical support for manual search and discovery.

call these “summary statistics.” Often we want to dig deeper, and calculate summary statistics *conditionally* on one or more subsets of the population (e.g., “Does the churn rate differ between male and female customers?” and “What about high-income customers in the Northeast (denotes a region of the USA)?”) Summary statistics are the basic building blocks of much data science theory and practice.

Summary statistics should be chosen with close attention to the business problem to be solved (one of the fundamental principles we will present later), and also with attention to the *distribution* of the data they are summarizing. For example, the average (mean) income in the United States according to the 2004 Census Bureau Economic Survey was over $60,000. If we were to use that as a measure of the average income in order to make policy decisions, we would be misleading ourselves. The distribution of incomes in the U.S. is highly skewed, with many people making relatively little and some people making fantastically much. In such cases, the arithmetic mean tells us relatively little about how much people are making. Instead, we should use a different measure of “average” income, such as the median. The median income—that amount where half the population makes more and half makes less—in the U.S. in the 2004 Census study was only $44,389 —considerably less than the mean. This example may seem obvious because we are so accustomed to hearing about the “median income,” but the same reasoning applies to any computation of summary statistics: have you thought about the problem you would like to solve or the question you would like to answer? Have you considered the distribution of the data, and whether the chosen statistic is appropriate?

The other use of the term “statistics” is to denote the field of study that goes by that name, for which we might differentiate by using the proper name, Statistics. The field of Statistics provides us with a huge amount of knowledge that underlies analytics, and can be thought of as a component of the larger field of Data Science. For example, Statistics helps us to understand different data distributions and what statistics are appropriate to summarize each. Statistics helps us understand how to use data to test hypotheses and to estimate the uncertainty of conclusions. In relation to data mining, hypothesis testing can help determine whether an observed pattern is likely to be a valid, general regularity as opposed to a chance occurrence in some particular dataset. Most relevant to this book, many of the techniques for extracting models or patterns from data have their roots in Statistics.

For example, a preliminary study may suggest that customers in the Northeast have a churn rate of 22.5%, whereas the nationwide average churn rate is only 15%. This may be just a chance fluctuation since the churn rate is not constant; it varies over regions and over time, so differences are to be expected. But the Northeast rate is one and a half times the U.S. average, which seems unusually high. What is the chance that this is due to random variation? Statistical hypothesis testing is used to answer such questions.

Closely related is the quantification of uncertainty into confidence intervals. The overall churn rate is 15%, but there is some variation; traditional statistical analysis may reveal that 95% of the time the churn rate is expected to fall between 13% and 17%.

This contrasts with the (complementary) process of data mining, which may be seen as hypothesis *generation*. Can we find patterns in data in the first place? Hypothesis generation should then be followed by careful hypothesis testing (generally on different data; see Chapter 5). In addition, data mining procedures may produce numerical estimates, and we often also want to provide confidence intervals on these estimates. We will return to this when we discuss the evaluation of the results of data mining.

In this book we are not going to spend more time discussing these basic statistical concepts. There are plenty of introductory books on statistics and statistics for business, and any treatment we would try to squeeze in would be either very narrow or superficial.

That said, one statistical term that is often heard in the context of business analytics is "correlation." For example, "Are there any indicators that correlate with a customer's later defection?" As with the term statistics, "correlation" has both a general-purpose meaning (variations in one quantity tell us something about variations in the other), and a specific technical meaning (e.g., linear correlation based on a particular mathematical formula). The notion of correlation will be the jumping off point for the rest of our discussion of data science for business, starting in the next chapter.

Database Querying

A *query* is a specific request for a subset of data or for statistics about data, formulated in a technical language and posed to a database system. Many tools are available to answer one-off or repeating queries about data posed by an analyst. These tools are usually frontends to database systems, based on Structured Query Language (SQL) or a tool with a graphical user interface (GUI) to help formulate queries (e.g., query-by-example, or QBE). For example, if the analyst can define "profitable" in operational terms computable from items in the database, then a query tool could answer: "Who are the most profitable customers in the Northeast?" The analyst may then run the query to retrieve a list of the most profitable customers, possibly ranked by profitability. This activity differs fundamentally from data mining in that there is no discovery of patterns or models.

Database queries are appropriate when an analyst already has an idea of what might be an interesting subpopulation of the data, and wants to investigate this population or confirm a hypothesis about it. For example, if an analyst suspects that middle-aged men living in the Northeast have some particularly interesting churning behavior, she could compose a SQL query:

```
SELECT * FROM CUSTOMERS WHERE AGE > 45 and SEX='M' and DOMICILE = 'NE'
```

If those are the people to be targeted with an offer, a query tool can be used to retrieve all of the information about them ("*") from the `CUSTOMERS` table in the database.

In contrast, data mining could be used to come up with this query in the first place—as a pattern or regularity in the data. A data mining procedure might examine prior customers who did and did not defect, and determine that this segment (characterized as "AGE is greater than 45 and SEX is male and DOMICILE is Northeast-USA") is predictive with respect to churn rate. After translating this into a SQL query, a query tool could then be used to find the matching records in the database.

Query tools generally have the ability to execute sophisticated logic, including computing summary statistics over subpopulations, sorting, joining together multiple tables with related data, and more. Data scientists often become quite adept at writing queries to extract the data they need.

On-line Analytical Processing (OLAP) provides an easy-to-use GUI to query large data collections, for the purpose of facilitating data exploration. The idea of "on-line" processing is that it is done in realtime, so analysts and decision makers can find answers to their queries quickly and efficiently. Unlike the "ad hoc" querying enabled by tools like SQL, for OLAP the dimensions of analysis must be pre-programmed into the OLAP system. If we've foreseen that we would want to explore sales volume by region and time, we could have these three dimensions programmed into the system, and drill down into populations, often simply by clicking and dragging and manipulating dynamic charts.

OLAP systems are designed to facilitate manual or visual exploration of the data by analysts. OLAP performs no modeling or automatic pattern finding. As an additional contrast, unlike with OLAP, data mining tools generally can incorporate new dimensions of analysis easily as part of the exploration. OLAP tools can be a useful complement to data mining tools for discovery from business data.

Data Warehousing

Data warehouses collect and coalesce data from across an enterprise, often from multiple transaction-processing systems, each with its own database. Analytical systems can access data warehouses. Data warehousing may be seen as a facilitating technology of data mining. It is not always necessary, as most data mining does not access a data warehouse, but firms that decide to invest in data warehouses often can apply data mining more broadly and more deeply in the organization. For example, if a data warehouse integrates records from sales and billing as well as from human resources, it can be used to find characteristic patterns of effective salespeople.

Regression Analysis

Some of the same methods we discuss in this book are at the core of a different set of analytic methods, which often are collected under the rubric *regression analysis*, and are widely applied in the field of statistics and also in other fields founded on econometric analysis. This book will focus on different issues than usually encountered in a regression analysis book or class. Here we are less interested in explaining a particular dataset as we are in extracting patterns that will generalize to other data, and for the purpose of improving some business process. Typically, this will involve estimating or predicting values for cases that are not in the analyzed data set. So, as an example, in this book we are less interested in digging into the reasons for churn (important as they may be) in a particular historical set of data, and more interested in predicting which customers who have not yet left would be the best to target to reduce future churn. Therefore, we will spend some time talking about testing patterns on new data to evaluate their generality, and about techniques for reducing the tendency to find patterns specific to a particular set of data, but that do not generalize to the population from which the data come.

The topic of explanatory modeling versus predictive modeling can elicit deep-felt debate,[5] which goes well beyond our focus. What is important is to realize that there is considerable overlap in the *techniques* used, but that the lessons learned from explanatory modeling do not all apply to predictive modeling. So a reader with some background in regression analysis may encounter new and even seemingly contradictory lessons.[6]

Machine Learning and Data Mining

The collection of methods for extracting (predictive) models from data, now known as machine learning methods, were developed in several fields contemporaneously, most notably Machine Learning, Applied Statistics, and Pattern Recognition. Machine Learning as a field of study arose as a subfield of Artificial Intelligence, which was concerned with methods for improving the knowledge or performance of an intelligent agent over time, in response to the agent's experience in the world. Such improvement often involves analyzing data from the environment and making predictions about unknown quantities, and over the years this data analysis aspect of machine learning has come to play a very large role in the field. As machine learning methods were deployed broadly, the scientific disciplines of Machine Learning, Applied Statistics, and Pattern Recognition developed close ties, and the separation between the fields has blurred.

5. The interested reader is urged to read the discussion by Shmueli (2010).

6. Those who pursue the study in depth will have the seeming contradictions worked out. Such deep study is not necessary to understand the fundamental principles.

The field of Data Mining (or KDD: Knowledge Discovery and Data Mining) started as an offshoot of Machine Learning, and they remain closely linked. Both fields are concerned with the analysis of data to find useful or informative patterns. Techniques and algorithms are shared between the two; indeed, the areas are so closely related that researchers commonly participate in both communities and transition between them seamlessly. Nevertheless, it is worth pointing out some of the differences to give perspective.

Speaking generally, because Machine Learning is concerned with many types of performance improvement, it includes subfields such as robotics and computer vision that are not part of KDD. It also is concerned with issues of *agency* and *cognition*—how will an intelligent agent use learned knowledge to reason and act in its environment—which are not concerns of Data Mining.

Historically, KDD spun off from Machine Learning as a research field focused on concerns raised by examining real-world applications, and a decade and a half later the KDD community remains more concerned with applications than Machine Learning is. As such, research focused on commercial applications and business issues of data analysis tends to gravitate toward the KDD community rather than to Machine Learning. KDD also tends to be more concerned with the entire process of data analytics: data preparation, model learning, evaluation, and so on.

Answering Business Questions with These Techniques

To illustrate how these techniques apply to business analytics, consider a set of questions that may arise and the technologies that would be appropriate for answering them. These questions are all related but each is subtly different. It is important to understand these differences in order to understand what technologies one needs to employ and what people may be necessary to consult.

1. *Who are the most profitable customers?*

   If "profitable" can be defined clearly based on existing data, this is a straightforward database query. A standard query tool could be used to retrieve a set of customer records from a database. The results could be sorted by cumulative transaction amount, or some other operational indicator of profitability.

2. *Is there really a difference between the profitable customers and the average customer?*

   This is a question about a conjecture or hypothesis (in this case, "There is a difference in value to the company between the profitable customers and the average customer"), and statistical hypothesis testing would be used to confirm or disconfirm it. Statistical analysis could also derive a probability or confidence bound that the difference was real. Typically, the result would be like: "The value of these profitable customers is significantly different from that of the average customer, with probability < 5% that this is due to random chance."

3. *But who really are these customers? Can I characterize them?*

   We often would like to do more than just list out the profitable customers. We would like to describe common characteristics of profitable customers. The characteristics of individual customers can be extracted from a database using techniques such as database querying, which also can be used to generate summary statistics. A deeper analysis should involve determining what characteristics *differentiate* profitable customers from unprofitable ones. This is the realm of data science, using data mining techniques for automated pattern finding—which we discuss in depth in the subsequent chapters.

4. *Will some particular new customer be profitable? How much revenue should I expect this customer to generate?*

   These questions could be addressed by data mining techniques that examine historical customer records and produce predictive models of profitability. Such techniques would generate models from historical data that could then be applied to new customers to generate predictions. Again, this is the subject of the following chapters.

Note that this last pair of questions are subtly different data mining questions. The first, a classification question, may be phrased as a prediction of whether a given new customer will be profitable (yes/no or the probability thereof). The second may be phrased as a prediction of the value (numerical) that the customer will bring to the company. More on that as we proceed.

Summary

Data mining is a craft. As with many crafts, there is a well-defined process that can help to increase the likelihood of a successful result. This process is a crucial conceptual tool for thinking about data science projects. We will refer back to the data mining process repeatedly throughout the book, showing how each fundamental concept fits in. In turn, understanding the fundamentals of data science substantially improves the chances of success as an enterprise invokes the data mining process.

The various fields of study related to data science have developed a set of canonical task types, such as classification, regression, and clustering. Each task type serves a different purpose and has an associated set of solution techniques. A data scientist typically attacks a new project by decomposing it such that one or more of these canonical tasks is revealed, choosing a solution technique for each, then composing the solutions. Doing this expertly may take considerable experience and skill. A successful data mining project involves an intelligent compromise between what the data can do (i.e., what they can predict, and how well) and the project goals. For this reason it is important to keep in mind how data mining results will be used, and use this to inform the data mining process itself.

Data mining differs from, and is complementary to, important supporting technologies such as statistical hypothesis testing and database querying (which have their own books and classes). Though the boundaries between data mining and related techniques are not always sharp, it is important to know about other techniques' capabilities and strengths to know when they should be used.

To a business manager, the data mining process is useful as a framework for analyzing a data mining project or proposal. The process provides a systematic organization, including a set of questions that can be asked about a project or a proposed project to help understand whether the project is well conceived or is fundamentally flawed. We will return to this after we have discussed in detail some more of the fundamental principles themselves—to which we turn now.

CHAPTER 3

Introduction to Predictive Modeling: From Correlation to Supervised Segmentation

Fundamental concepts: *Identifying informative attributes; Segmenting data by progressive attribute selection.*

Exemplary techniques: *Finding correlations; Attribute/variable selection; Tree induction.*

The previous chapters discussed models and modeling at a high level. This chapter delves into one of the main topics of data mining: predictive modeling. Following our example of data mining for churn prediction from the first section, we will begin by thinking of predictive modeling as *supervised* segmentation—how can we segment the population into groups that differ from each other with respect to some quantity of interest. In particular, how can we segment the population with respect to something that we would like to predict or estimate. The target of this prediction can be something we would like to avoid, such as which customers are likely to leave the company when their contracts expire, which accounts have been defrauded, which potential customers are likely not to pay off their account balances (*write-offs*, such as defaulting on one's phone bill or credit card balance), or which web pages contain objectionable content. The target might instead be cast in a positive light, such as which consumers are most likely to respond to an advertisement or special offer, or which web pages are most appropriate for a search query.

In the process of discussing supervised segmentation, we introduce one of the fundamental ideas of data mining: finding or selecting important, informative variables or "attributes" of the entities described by the data. What exactly it means to be "informative" varies among applications, but generally, *information is a quantity that reduces uncertainty about something.* So, if an old pirate gives me information about where his treasure is hidden that does not mean that I know for certain where it is, it only means that my uncertainty about where the treasure is hidden is reduced. The better the information, the more my uncertainty is reduced.

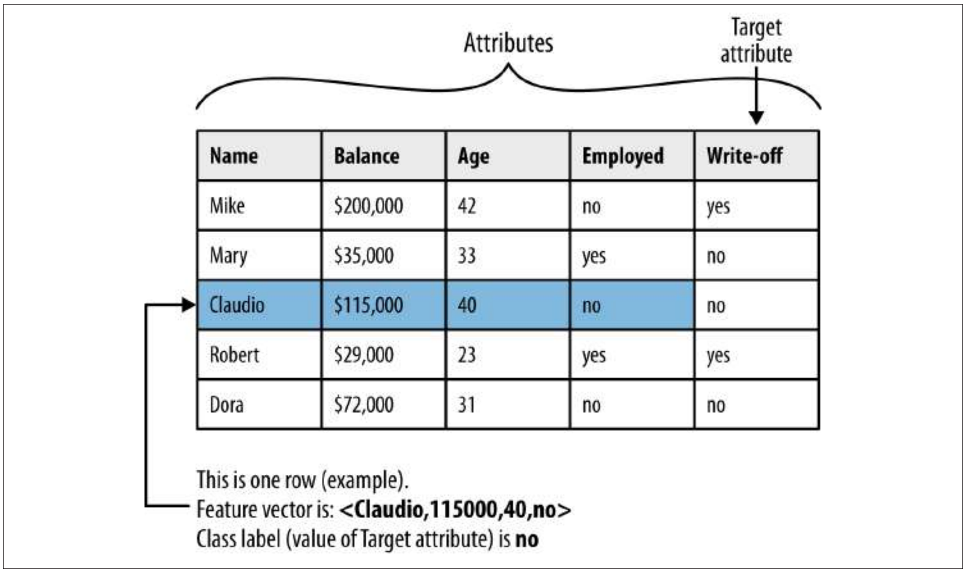
Now, recall the notion of "supervised" data mining from the previous chapter. A key to supervised data mining is that we have some target quantity we would like to predict or to otherwise understand better. Often this quantity is unknown or unknowable at the time we would like to make a business decision, such as whether a customer will churn soon after her contract expires, or which accounts have been defrauded. Having a target variable crystalizes our notion of finding informative attributes: is there one or more other variables that reduces our uncertainty about the value of the target? This also gives a common analytics application of the general notion of correlation discussed above: we would like to find knowable attributes that correlate with the target of interest —that reduce our uncertainty in it. Just finding these correlated variables may provide important insight into the business problem.

Finding informative attributes also is useful to help us deal with increasingly larger databases and data streams. Datasets that are too large pose computational problems for analytic techniques, especially when the analyst does not have access to high-performance computers. One tried-and-true method for analyzing very large datasets is first to select a subset of the data to analyze. Selecting informative attributes provides an "intelligent" method for selecting an informative subset of the data. In addition, attribute selection prior to data-driven modeling can increase the accuracy of the modeling, for reasons we will discuss in Chapter 5.

Finding informative attributes also is the basis for a widely used predictive modeling technique called *tree induction*, which we will introduce toward the end of this chapter as an application of this fundamental concept. Tree induction incorporates the idea of supervised segmentation in an elegant manner, repeatedly selecting informative attributes. By the end of this chapter we will have achieved an understanding of: the basic concepts of predictive modeling; the fundamental notion of finding informative attributes, along with one particular, illustrative technique for doing so; the notion of tree-structured models; and a basic understanding of the process for extracting tree-structured models from a dataset—performing supervised segmentation.

Models, Induction, and Prediction

Generally speaking, a model is a simplified representation of reality created to serve a purpose. It is simplified based on some assumptions about what is and is not important for the specific purpose, or sometimes based on constraints on information or tractability. For example, a map is a model of the physical world. It abstracts away a tremendous amount of information that the mapmaker deemed irrelevant for its purpose. It preserves, and sometimes further simplifies, the relevant information. For example, a road map keeps and highlights the roads, their basic topology, their relationships to places one would want to travel, and other relevant information. Various professions have well-known model types: an architectural blueprint, an engineering prototype, the


Attributes
Target attribute

Name	Balance	Age	Employed	Write-off
Mike	$200,000	42	no	yes
Mary	$35,000	33	yes	no
Claudio	$115,000	40	no	no
Robert	$29,000	23	yes	yes
Dora	$72,000	31	no	no

This is one row (example).
Feature vector is: **<Claudio,115000,40,no>**
Class label (value of Target attribute) is **no**

Figure 3-1. Data mining terminology for a supervised classification problem. The problem is supervised because it has a target attribute and some "training" data where we know the value for the target attribute. It is a classification (rather than regression) problem because the target is a category (yes or no) rather than a number.

Black-Scholes model of option pricing, and so on. Each of these abstracts away details that are not relevant to their main purpose and keeps those that are.

In data science, a predictive model is a formula for estimating the unknown value of interest: the target. The formula could be mathematical, or it could be a logical statement such as a rule. Often it is a hybrid of the two. Given our division of supervised data mining into classification and regression, we will consider classification models (and class-probability estimation models) and regression models.

Terminology: Prediction

In common usage, prediction means to forecast a future event. In data science, prediction more generally means *to estimate an unknown value*. This value could be something in the future (in common usage, true prediction), but it could also be something in the present or in the past. Indeed, since data mining usually deals with historical data, models very often are built and tested using events from the past. Predictive models for credit scoring estimate the likelihood that a potential customer will default (become a write-off). Predictive models for spam filtering estimate whether a given piece of email is spam. Predictive models for fraud detection judge whether an account has been defrauded. The key is that the model is intended to be used to estimate an unknown value.

This is in contrast to *descriptive* modeling, where the primary purpose of the model is not to estimate a value but instead to gain insight into the underlying phenomenon or process. A descriptive model of churn behavior would tell us what customers who churn typically look like.[1] A descriptive model must be judged in part on its intelligibility, and a less accurate model may be preferred if it is easier to understand. A predictive model may be judged solely on its predictive performance, although we will discuss why intelligibility is nonetheless important. The difference between these model types is not as strict as this may imply; some of the same techniques can be used for both, and usually one model can serve both purposes (though sometimes poorly). Sometimes much of the value of a predictive model is in the understanding gained from looking at it rather than in the predictions it makes.

Before we discuss predictive modeling further, we must introduce some terminology. Supervised learning is model creation where the model describes a relationship between a set of selected variables (*attributes* or *features*) and a predefined variable called the *target* variable. The model estimates the value of the target variable as a function (possibly a probabilistic function) of the features. So, for our churn-prediction problem we would like to build a model of the propensity to churn as a function of customer account attributes, such as age, income, length with the company, number of calls to customer service, overage charges, customer demographics, data usage, and others.

Figure 3-1 illustrates some of the terminology we introduce here, in an oversimplified example problem of credit write-off prediction. An *instance* or *example* represents a fact or a data point—in this case a historical customer who had been given credit. This is also called a *row* in database or spreadsheet terminology. An instance is described by a set of *attributes* (fields, columns, variables, or features). An instance is also sometimes called a *feature vector*, because it can be represented as a fixed-length ordered collection (vector) of feature values. Unless stated otherwise, we will assume that the values of all the attributes (but not the target) are present in the data.

1. Descriptive modeling often is used to work toward a causal understanding of the data generating process (*why do people churn?*).

Many Names for the Same Things

The principles and techniques of data science historically have been studied in several different fields, including machine learning, pattern recognition, statistics, databases, and others. As a result there often are several different names for the same things. We typically will refer to a *dataset*, whose form usually is the same as a *table* of a database or a *worksheet* of a spreadsheet. A dataset contains a set of *examples* or *instances*. An instance also is referred to as a *row* of a database table or sometimes a *case* in statistics.

The features (table columns) have many different names as well. Statisticians speak of *independent variables* or *predictors* as the attributes supplied as input. In operations research you may also hear *explanatory variable*. The target variable, whose values are to be predicted, is commonly called the *dependent variable* in statistics. This terminology may be somewhat confusing; the independent variables may not be independent of each other (or anything else), and the dependent variable doesn't always depend on all the independent variables. For this reason we have avoided the dependent/independent terminology in this book. Some experts consider the target variable to be included in the set of features, some do not. The important thing is rather obvious: the target variable is not used to predict itself. However, it may be that prior values for the target variable are quite helpful to predict future values—so such prior values may be included as features.

The creation of models from data is known as model induction. Induction is a term from philosophy that refers to generalizing from specific cases to general rules (or laws, or truths). Our models are general rules in a statistical sense (they usually do not hold 100% of the time; often not nearly), and the procedure that creates the model from the data is called the induction algorithm or learner. Most inductive procedures have variants that induce models both for classification and for regression. We will discuss mainly classification models because they tend to receive less attention in other treatments of statistics, and because they are relevant to many business problems (and thus much work in data science focuses on classification).

Terminology: Induction and deduction

Induction can be contrasted with *deduction*. Deduction starts with general rules and specific facts, and creates other specific facts from them. The *use* of our models can be considered a procedure of (probabilistic) deduction. We will get to this shortly.

The input data for the induction algorithm, used for inducing the model, are called the *training* data. As mentioned in Chapter 2, they are called *labeled* data because the value for the target variable (the label) is known.

Let's return to our example churn problem. Based on what we learned in Chapter 1 and Chapter 2, we might decide that in the modeling stage we should build a "supervised segmentation" model, which divides the sample into segments having (on average) higher or lower tendency to leave the company after contract expiration. To think about how this might be done, let's now turn to one of our fundamental concepts: How can we select one or more attributes/features/variables that will best divide the sample *with respect to our target variable of interest*?

Supervised Segmentation

Recall that a predictive model focuses on estimating the value of some particular target variable of interest. An intuitive way of thinking about extracting patterns from data in a supervised manner is to try to segment the population into subgroups that have different values for the target variable (and within the subgroup the instances have similar values for the target variable). If the segmentation is done using values of variables that will be known when the target is not, then these segments can be used to predict the value of the target variable. Moreover, the segmentation may at the same time provide a human-understandable set of segmentation patterns. One such segment expressed in English might be: "Middle-aged professionals who reside in New York City on average have a churn rate of 5%." Specifically, the term "middle-aged professionals who reside in New York City" is the definition of the segment (which references some particular attributes) and "a churn rate of 5%" describes the predicted value of the target variable for the segment.[2]

Often we are interested in applying data mining when we have many attributes, and are not sure exactly what the segments should be. In our churn-prediction problem, who is to say what are the best segments for predicting the propensity to churn? If there exist in the data segments with significantly different (average) values for the target variable, we would like to be able to extract them automatically.

This brings us to our fundamental concept: how can we judge whether a variable contains important information about the target variable? How much? We would like automatically to get a selection of the more informative variables with respect to the particular task at hand (namely, predicting the value of the target variable). Even better, we might like to rank the variables by how good they are at predicting the value of the target.

2. The predicted value can be estimated from the data in different ways, which we will get to. At this point we can think of it roughly as an average of some sort from the training data that fall into the segment.

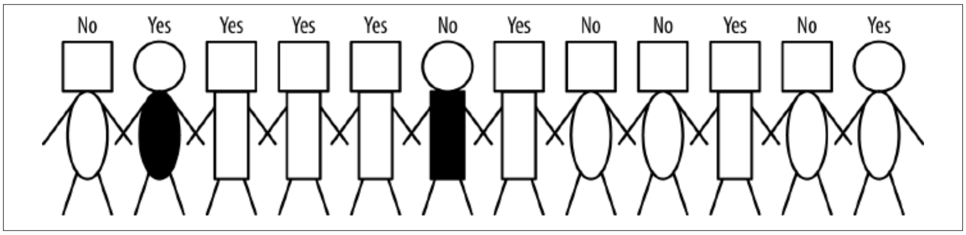


Figure 3-2. A set of people to be classified. The label over each head represents the value of the target variable (write-off or not). Colors and shapes represent different predictor attributes.

Consider just the selection of the single most informative attribute. Solving this problem will introduce our first concrete data mining technique—simple, but easily extendable to be very useful. In our example, what variable gives us the most information about the future churn rate of the population? Being a professional? Age? Place of residence? Income? Number of complaints to customer service? Amount of overage charges?

We now will look carefully into one useful way to select informative variables, and then later will show how this technique can be used repeatedly to build a supervised segmentation. While very useful and illustrative, please keep in mind that direct, multivariate supervised segmentation is just one application of this fundamental idea of selecting informative variables. This notion should become one of your conceptual tools when thinking about data science problems more generally. For example, as we go forward we will delve into other modeling approaches, ones that do not incorporate variable selection directly. When the world presents you with very large sets of attributes, it may be (extremely) useful to harken back to this early idea and to select a subset of informative attributes. Doing so can substantially reduce the size of an unwieldy dataset, and as we will see, often will improve the accuracy of the resultant model.

Selecting Informative Attributes

Given a large set of examples, how do we select an attribute to partition them in an informative way? Let's consider a binary (two class) classification problem, and think about what we would like to get out of it. To be concrete, Figure 3-2 shows a simple segmentation problem: twelve people represented as stick figures. There are two types of heads: square and circular; and two types of bodies: rectangular and oval; and two of the people have gray bodies while the rest are white.

These are the attributes we will use to describe the people. Above each person is the binary target label, *Yes* or *No*, indicating (for example) whether the person becomes a loan write-off. We could describe the data on these people as:

- Attributes:
  - — head-shape: square, circular
  - — body-shape: rectangular, oval
  - — body-color: gray, white
- Target variable:
  - — write-off: Yes, No

So let's ask ourselves: which of the attributes would be best to segment these people into groups, in a way that will distinguish write-offs from non-write-offs? Technically, we would like the resulting groups to be as *pure* as possible. By pure we mean *homogeneous with respect to the target variable*. If every member of a group has the same value for the target, then the group is pure. If there is at least one member of the group that has a different value for the target variable than the rest of the group, then the group is impure.

Unfortunately, in real data we seldom expect to find a variable that will make the segments pure. However, if we can reduce the impurity substantially, then we can both learn something about the data (and the corresponding population), and importantly for this chapter, we can use the attribute in a predictive model—in our example, predicting that members of one segment will have higher or lower write-off rates than those in another segment. If we can do that, then we can for example offer credit to those with the lower predicted write-off rates, or can offer different credit terms based on the different predicted write-off rates.

Technically, there are several complications:

1. Attributes rarely split a group perfectly. Even if one subgroup happens to be pure, the other may not. For example, in Figure 3-2, consider if the second person were not there. Then *body-color=gray* would create a pure segment (*write-off=no*). However, the other associated segment, *body-color=white*, still is not pure.
2. In the prior example, the condition *body-color=gray* only splits off one single data point into the pure subset. Is this better than another split that does not produce any pure subset, but reduces the impurity more broadly?
3. Not all attributes are binary; many attributes have three or more distinct values. We must take into account that one attribute can split into two groups while another might split into three groups, or seven. How do we compare these?
4. Some attributes take on numeric values (continuous or integer). Does it make sense to make a segment for every numeric value? (No.) How should we think about creating supervised segmentations using numeric attributes?

Fortunately, for classification problems we can address all the issues by creating a formula that evaluates how well each attribute splits a set of examples into segments, with respect to a chosen target variable. Such a formula is based on a *purity measure*.

The most common splitting criterion is called *information gain*, and it is based on a purity measure called *entropy*. Both concepts were invented by one of the pioneers of information theory, Claude Shannon, in his seminal work in the field (Shannon, 1948).

Entropy is a measure of disorder that can be applied to a set, such as one of our individual segments. Consider that we have a set of *properties* of members of the set, and each member has one and only one of the properties. In supervised segmentation, the member properties will correspond to the values of the target variable. Disorder corresponds to how mixed (impure) the segment is with respect to these properties of interest. So, for example, a mixed up segment with lots of write-offs and lots of non-write-offs would have high entropy.

More technically, entropy is defined as:

Equation 3-1. Entropy

$$\textit{entropy} = -p_1 \log(p_1) - p_2 \log(p_2) - \cdots$$

Each p_i is the probability (the relative percentage) of property i within the set, ranging from $p_i = 1$ when all members of the set have property i, and $p_i = 0$ when no members of the set have property i. The … simply indicates that there may be more than just two properties (and for the technically minded, the logarithm is generally taken as base 2).

Since the entropy equation might not lend itself to intuitive understanding, Figure 3-3 shows a plot of the entropy of a set containing 10 instances of two classes, + and –. We can see then that entropy measures the general disorder of the set, ranging from zero at minimum disorder (the set has members all with the same, single property) to one at maximal disorder (the properties are equally mixed). Since there are only two classes, $p_+ = 1 - p_-$. Starting with all negative instances at the lower left, $p_+ = 0$, the set has minimal disorder (it is pure) and the entropy is zero. If we start to switch class labels of elements of the set from – to +, the entropy increases. Entropy is maximized at 1 when the instance classes are balanced (five of each), and $p_+ = p_- = 0.5$. As more class labels are switched, the + class starts to predominate and the entropy lowers again. When all instances are positive, $p_+ = 1$ and entropy is minimal again at zero.

As a concrete example, consider a set *S* of 10 people with seven of the *non-write-off* class and three of the *write-off* class. So:

$$p(\text{non-write-off}) = 7 / 10 = 0.7$$
$$p(\text{write-off}) = 3 / 10 = 0.3$$

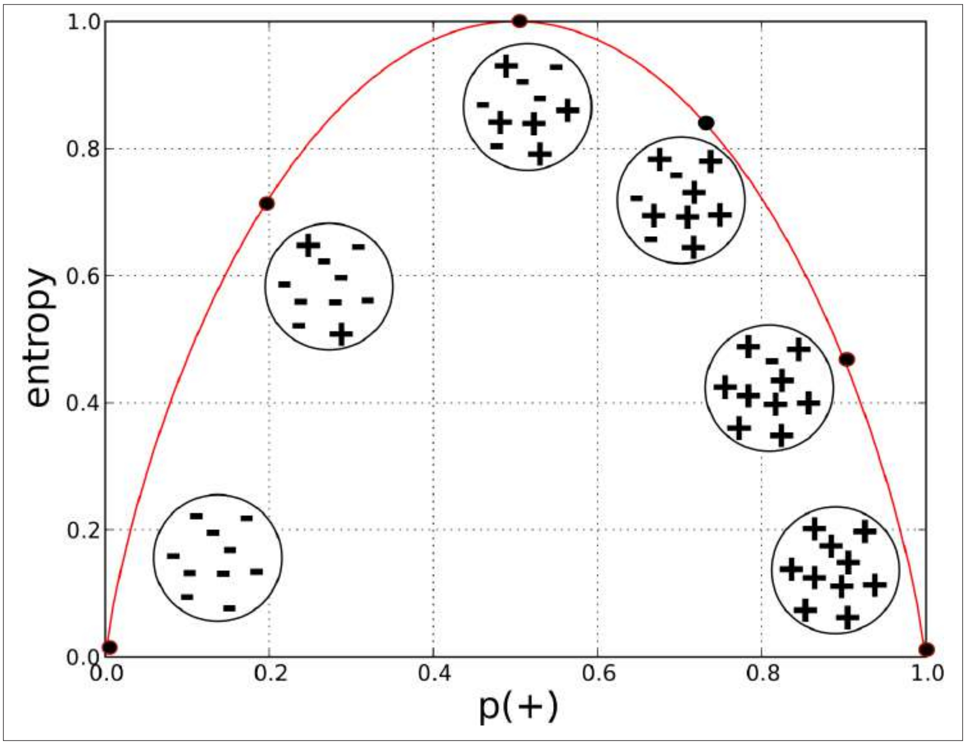


Figure 3-3. Entropy of a two-class set as a function of p(+).

$$\begin{aligned} entropy(S) &= -[0.7 \times \log_2 (0.7) + 0.3 \times \log_2 (0.3)] \\ &\approx -[0.7 \times -0.51 + 0.3 \times -1.74] \\ &\approx 0.88 \end{aligned}$$

Entropy is only part of the story. We would like to measure how *informative* an attribute is with respect to our target: how much gain in information it gives us about the value of the target variable. An attribute segments a set of instances into several subsets. Entropy only tells us how impure one individual subset is. Fortunately, with entropy to measure how disordered any set is, we can define *information gain* (IG) to measure how much an attribute improves (decreases) entropy over the whole segmentation it creates. Strictly speaking, information gain measures the *change* in entropy due to any amount of new information being added; here, in the context of supervised segmentation, we consider the information gained by splitting the set on all values of a single attribute. Let's say the attribute we split on has *k* different values. Let's call the original set of examples the *parent* set, and the result of splitting on the attribute values the *k children* sets. Thus, information gain is a function of both a parent set and of the children resulting from some partitioning of the parent set—how much information has this attribute provided? That depends on how much purer the children are than the parent. Stated in the context of predictive modeling, if we were to know the value of this attribute, how much would it increase our knowledge of the value of the target variable?

Specifically, the definition of information gain (IG) is:

Equation 3-2. Information gain

$$\begin{aligned} IG(parent, children) &= entropy(parent) - \\ & [p(c_1) \times entropy(c_1) + p(c_2) \times entropy(c_2) + \cdots] \end{aligned}$$

Notably, the entropy for each child (c_i) is weighted by the proportion of instances belonging to that child, $p(c_i)$. This addresses directly our concern from above that splitting off a single example, and noticing that that set is pure, may not be as good as splitting the parent set into two nice large, relatively pure subsets, even if neither is pure.

As an example, consider the split in Figure 3-4. This is a two-class problem (• and ★). Examining the figure, the children sets certainly seem "purer" than the parent set. The parent set has 30 instances consisting of 16 dots and 14 stars, so:

$$\begin{aligned} entropy(parent) &= -[p(\bullet) \times \log_2 p(\bullet) + p(☆) \times \log_2 p(☆)] \\ &\approx -[0.53 \times -0.9 + 0.47 \times -1.1] \\ &\approx 0.99 \quad \text{(very impure)} \end{aligned}$$

The entropy of the *left* child is:

$$\begin{aligned} entropy(Balance < 50K) &= -[p(\bullet) \times \log_2 p(\bullet) + p(☆) \times \log_2 p(☆)] \\ &\approx -[0.92 \times (-0.12) + 0.08 \times (-3.7)] \\ &\approx 0.39 \end{aligned}$$

The entropy of the *right* child is:

$$\begin{aligned} entropy(Balance \geq 50K) &= -[p(\bullet) \times \log_2 p(\bullet) + p(☆) \times \log_2 p(☆)] \\ &\approx -[0.24 \times (-2.1) + 0.76 \times (-0.39)] \\ &\approx 0.79 \end{aligned}$$

Using Equation 3-2, the information gain of this split is:

$$\begin{aligned} IG &= entropy(parent) - [\,p(\text{Balance} < 50\text{K}) \times entropy(\text{Balance} < 50\text{K}) \\ &\qquad + p(\text{Balance} \geq 50\text{K}) \times entropy(\text{Balance} \geq 50\text{K})] \\ &\approx 0.99 - [0.43 \times 0.39 + 0.57 \times 0.79] \\ &\approx 0.37 \end{aligned}$$

So this split reduces entropy substantially. In predictive modeling terms, the attribute provides a lot of information on the value of the target.

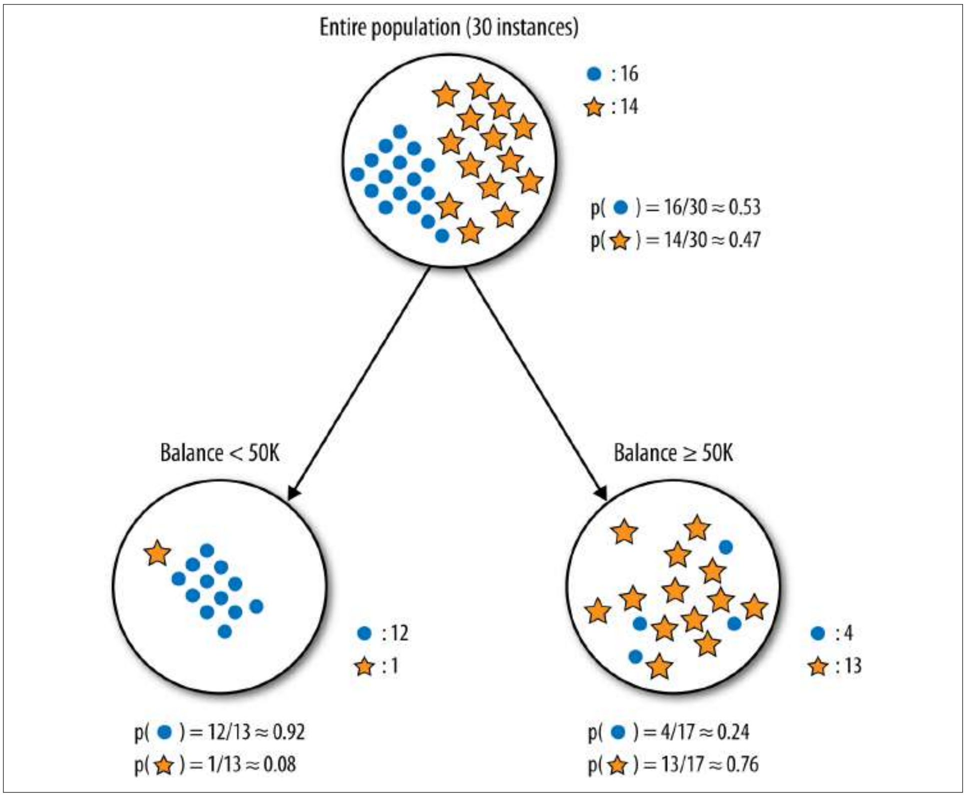


Figure 3-4. Splitting the "write-off" sample into two segments, based on splitting the Balance attribute (account balance) at 50K.

As a second example, consider another candidate split shown in Figure 3-5. This is the same parent set as in Figure 3-4, but instead we consider splitting on the attribute *Residence* with three values: OWN, RENT, and OTHER. Without showing the detailed calculations:

$$\begin{aligned}
entropy(parent) &\approx 0.99\\
entropy(\text{Residence=OWN}) &\approx 0.54\\
entropy(\text{Residence=RENT}) &\approx 0.97\\
entropy(\text{Residence=OTHER}) &\approx 0.98\\
IG &\approx 0.13
\end{aligned}$$

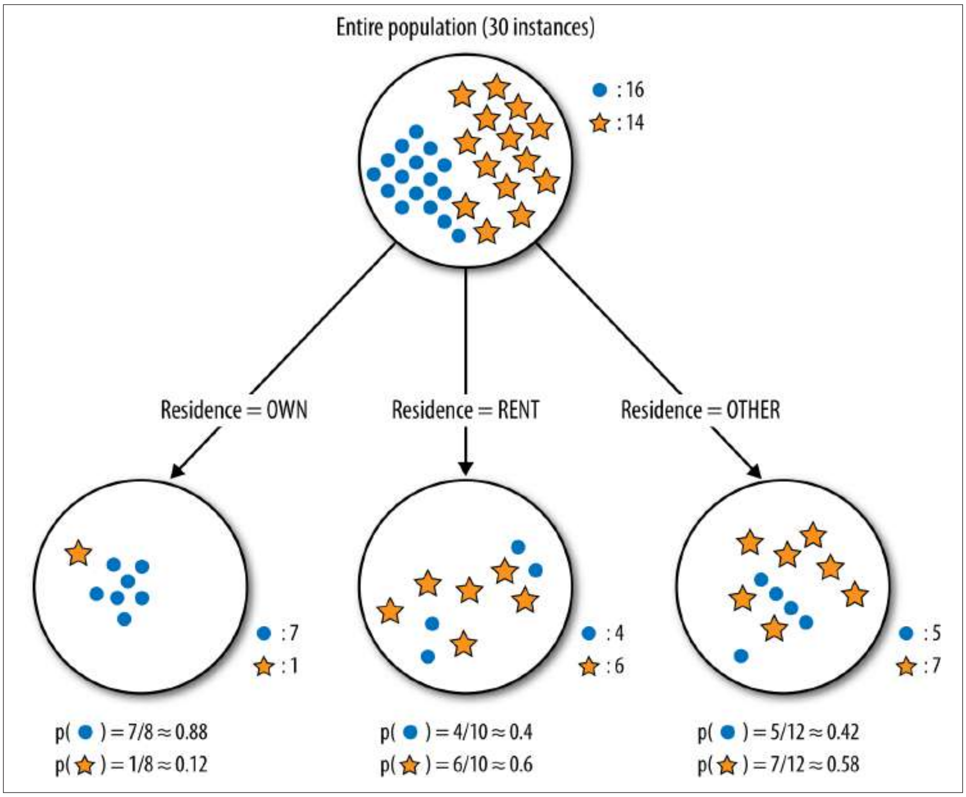


Figure 3-5. A classification tree split on the three-valued Residence attribute.

The Residence variable does have a positive information gain, but it is lower than that of Balance. Intuitively, this is because, while the one child Residence=OWN has considerably reduced entropy, the other values RENT and OTHER produce children that are no more pure than the parent. Thus, based on these data, the Residence variable is less informative than Balance.

Looking back at our concerns from above about creating supervised segmentation for classification problems, information gain addresses them all. It does not require absolute purity. It can be applied to any number of child subsets. It takes into account the relative sizes of the children, giving more weight to larger subsets.[3]

Numeric variables

We have not discussed what exactly to do if the attribute is numeric. Numeric variables can be "discretized" by choosing a split point (or many split points) and then treating the result as a categorical attribute. For example, Income could be divided into two or more ranges. Information gain can be applied to evaluate the segmentation created by this discretization of the numeric attribute. We still are left with the question of how to choose the split point(s) for the numeric attribute. Conceptually, we can try all reasonable split points, and choose the one that gives the highest information gain.

Finally, what about supervised segmentations for regression problems—problems with a numeric target variable? Looking at reducing the impurity of the child subsets still makes intuitive sense, but information gain is not the right measure, because entropy-based information gain is based on the distribution of the *properties* in the segmentation. Instead, we would want a measure of the purity of the numeric (target) values in the subsets.

We will not go through a derivation here, but the fundamental idea is important: a natural measure of impurity for numeric values is *variance*. If the set has all the same values for the numeric target variable, then the set is pure and the variance is zero. If the numeric target values in the set are very different, then the set will have high variance. We can create a similar notion to information gain by looking at reductions in variance between parent and children. The process proceeds in direct analogy to the derivation for information gain above. To create the best segmentation given a numeric target, we might choose the one that produces the best weighted average variance reduction. In essence, we again would be finding variables that have the best correlation with the target, or alternatively, are most predictive of the target.

Example: Attribute Selection with Information Gain

Now we are ready to apply our first concrete data mining technique. For a dataset with instances described by attributes and a target variable, we can determine which attribute is the most informative with respect to estimating the value of the target variable. (We will delve into this more deeply below.) We also can rank a set of attributes by their informativeness, in particular by their information gain. This can be used simply to

3. Technically, there remains a concern with attributes with very many values, as splitting on them may result in large information gain, but not be predictive. This problem ("overfitting") is the subject of Chapter 5.

understand the data better. It can be used to help predict the target. Or it can be used to reduce the size of the data to be analyzed, by selecting a subset of attributes in cases where we can not or do not want to process the entire dataset.

To illustrate the use of information gain, we introduce a simple but realistic dataset taken from the machine learning dataset repository at the University of California at Irvine.[4] It is a dataset describing edible and poisonous mushrooms taken from The Audubon Society Field Guide to North American Mushrooms. From the description:

> This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500–525). Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like “leaflets three, let it be” for Poisonous Oak and Ivy.

Each data example (instance) is one mushroom sample, described in terms of its observable attributes (the features). The twenty-odd attributes and the values for each are listed in Table 3-1. For a given example, each attribute takes on a single discrete value (e.g., *gill-color=black*). We use 5,644 examples from the dataset, comprising 2,156 poisonous and 3,488 edible mushrooms.

This is a classification problem because we have a target variable, called *edible?*, with two values *yes* (edible) and *no* (poisonous), specifying our two classes. Each of the rows in the training set has a value for this target variable. We will use information gain to answer the question: “Which single attribute is the most useful for distinguishing edible (edible?=Yes) mushrooms from poisonous (edible?=No) ones?” This is a basic attribute selection problem. In much larger problems we could imagine selecting the best ten or fifty attributes out of several hundred or thousand, and often you want do this if you suspect there are far too many attributes for your mining problem, or that many are not useful. Here, for simplicity, we will find the single best attribute instead of the top ten.

Table 3-1. The attributes of the Mushroom dataset

Attribute name	Possible values
CAP-SHAPE	`bell, conical, convex, flat, knobbed, sunken`
CAP-SURFACE	`fibrous, grooves, scaly, smooth`
CAP-COLOR	`brown, buff, cinnamon, gray, green, pink, purple, red, white, yellow`
BRUISES?	`yes, no`
ODOR	`almond, anise, creosote, fishy, foul, musty, none, pungent, spicy`
GILL-ATTACHMENT	`attached, descending, free, notched`

4. See this UC Irvine Machine Learning Repository page.

Attribute name	Possible values
GILL-SPACING	`close, crowded, distant`
GILL-SIZE	`broad, narrow`
GILL-COLOR	`black, brown, buff, chocolate, gray, green, orange, pink, purple, red, white, yellow`
STALK-SHAPE	`enlarging, tapering`
STALK-ROOT	`bulbous, club, cup, equal, rhizomorphs, rooted, missing`
STALK-SURFACE-ABOVE-RING	`fibrous, scaly, silky, smooth`
STALK-SURFACE-BELOW-RING	`fibrous, scaly, silky, smooth`
STALK-COLOR-ABOVE-RING	`brown, buff, cinnamon, gray, orange, pink, red, white, yellow`
STALK-COLOR-BELOW-RING	`brown, buff, cinnamon, gray, orange, pink, red, white, yellow`
VEIL-TYPE	`partial, universal`
VEIL-COLOR	`brown, orange, white, yellow`
RING-NUMBER	`none, one, two`
RING-TYPE	`cobwebby, evanescent, flaring, large, none, pendant, sheathing, zone`
SPORE-PRINT-COLOR	`black, brown, buff, chocolate, green, orange, purple, white, yellow`
POPULATION	`abundant, clustered, numerous, scattered, several, solitary`
HABITAT	`grasses, leaves, meadows, paths, urban, waste, woods`
EDIBLE? *(Target variable)*	`yes, no`

Since we now have a way to measure information gain this is straightforward: we are asking for the single attribute that gives the highest information gain.

To do this, we calculate the information gain achieved by splitting on each attribute. The information gain from Equation 3-2 is defined on a parent and a set of children. The parent in each case is the whole dataset. First we need *entropy*(*parent*), the entropy of the whole dataset. If the two classes were perfectly balanced in the dataset it would have an entropy of 1. This dataset is slightly unbalanced (more edible than poisonous mushrooms are represented) and its entropy is 0.96.

To illustrate entropy reduction graphically, we'll show a number of *entropy graphs* for the mushroom domain (Figure 3-6 through Figure 3-8). Each graph is a two-dimensional description of the entire dataset's entropy as it is divided in various ways by different attributes. On the *x* axis is the proportion of the dataset (0 to 1), and on the *y* axis is the entropy (also 0 to 1) of a given piece of the data. The amount of shaded area in each graph represents the amount of entropy in the dataset when it is divided by some chosen attribute (or not divided, in the case of Figure 3-6). Our goal of having the lowest entropy corresponds to having as *little* shaded area as possible.

The first chart, Figure 3-6, shows the entropy of the entire dataset. In such a chart, the highest possible entropy corresponds to the entire area being shaded; the lowest possible entropy corresponds to the entire area being white. Such a chart is useful for visualizing information gain from different partitions of a dataset, because any partition can be shown simply as slices of the graph (with widths corresponding to the proportion of the dataset), each with its own entropy. The weighted sum of entropies in the information gain calculation will be depicted simply by the total amount of shaded area.

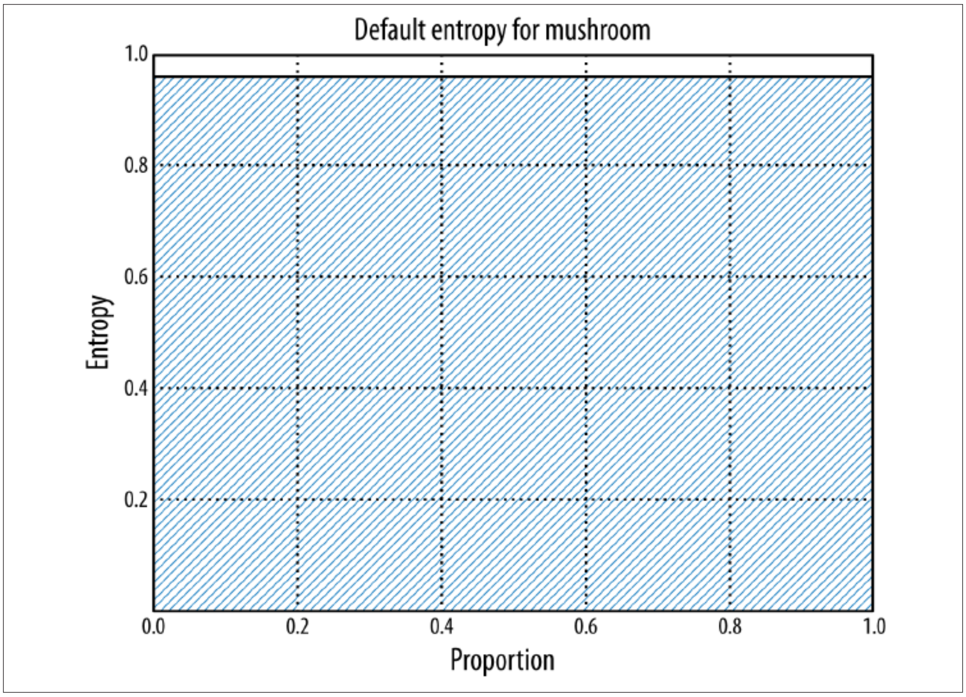


Figure 3-6. Entropy chart for the entire Mushroom dataset. The entropy for the entire dataset is 0.96, so 96% of the area is shaded.

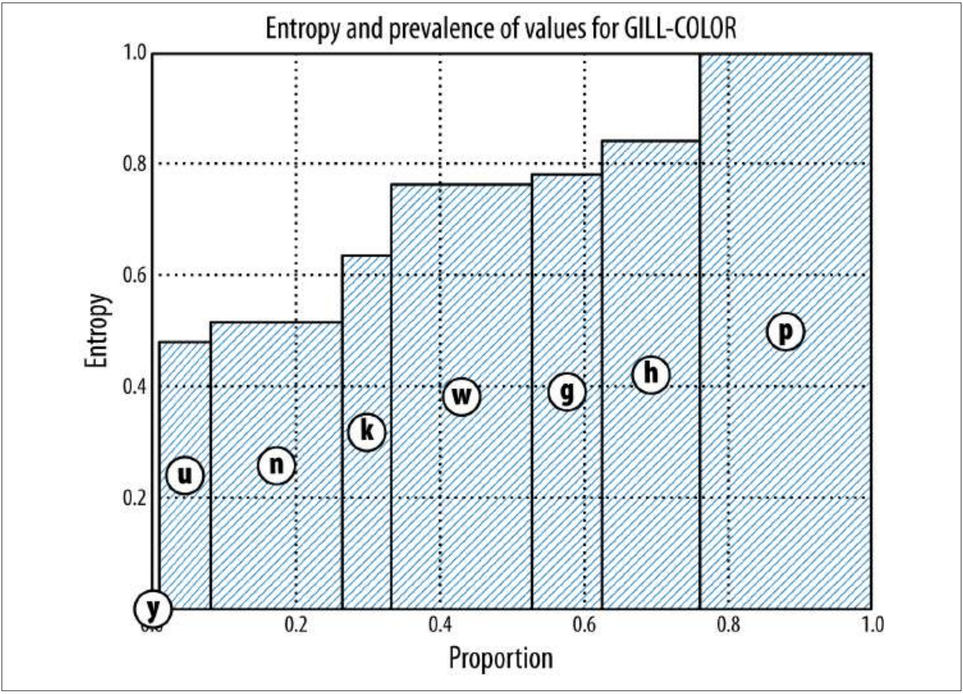


Figure 3-7. Entropy chart for the Mushroom dataset as split by GILL-COLOR. The amount of shading corresponds to the total (weighted sum) entropy, with each bar corresponding to the entropy of one of the attribute's values, and the width of the bar corresponding to the prevalence of that value in the data.

For our entire dataset, the global entropy is 0.96, so Figure 3-6 shows a large shaded area below the line *y* = 0.96. We can think of this as our starting entropy—any informative attribute should produce a new graph with less shaded area. Now we show the entropy charts of three sample attributes. Each value of an attribute occurs in the dataset with a different frequency, so each attribute splits the set in a different way.

Figure 3-7 shows the dataset split apart by the attribute GILL-COLOR, whose values are coded as y (`yellow`), u (`purple`), n (`brown`), and so on. The width of each attribute represents what proportion of the dataset has that value, and the height is its entropy. We can see that GILL-COLOR reduces the entropy somewhat; the shaded area in Figure 3-7 is considerably less than the area in Figure 3-6.

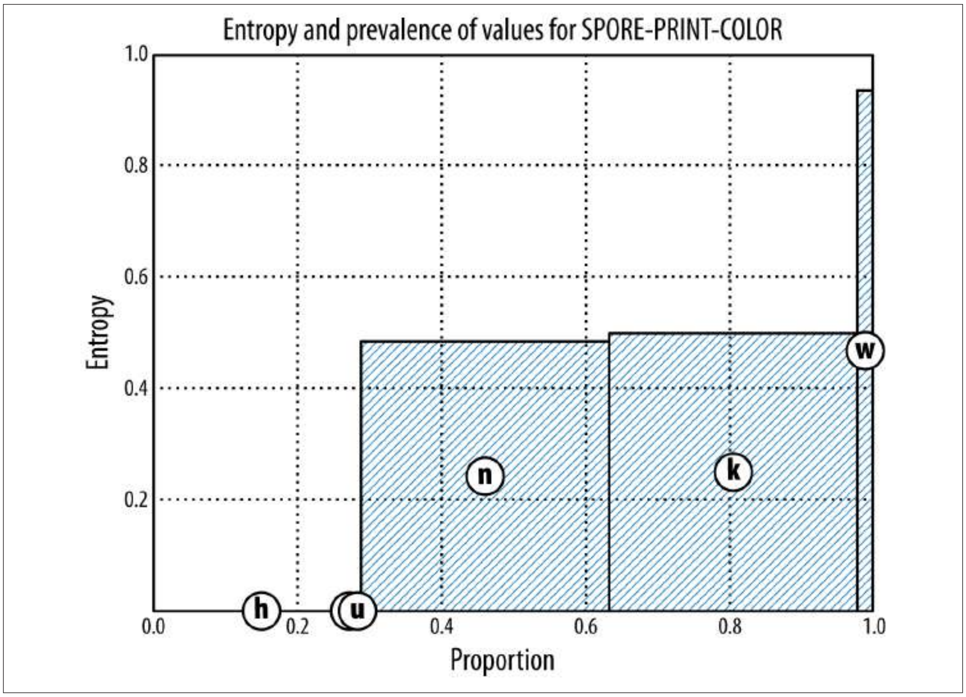


Figure 3-8. Entropy chart for the Mushroom dataset as split by SPORE-PRINT-COLOR. The amount of shading corresponds to the total (weighted sum) entropy, with each bar corresponding to the entropy of one of the attribute's values, and the width of the bar corresponding to the prevalence of that value in the data.

Similarly, Figure 3-8 shows how SPORE-PRINT-COLOR decreases uncertainty (entropy). A few of the values, such as h (`chocolate`), specify the target value perfectly and thus produce zero-entropy bars. But notice that they don't account for very much of the population, only about 30%.

Figure 3-9 shows the graph produced by ODOR. Many of the values, such as a (`almond`), c (`creosote`), and m (`musty`) produce zero-entropy partitions; only n (`no odor`) has a considerable entropy (about 20%). In fact, ODOR has the highest information gain of any attribute in the Mushroom dataset. It can reduce the dataset's total entropy to about 0.1, which gives it an information gain of 0.96 – 0.1 = 0.86. What is this saying? Many odors are completely characteristic of poisonous or edible mushrooms, so odor is a very informative attribute to check when considering mushroom edibility.[5] If you're

5. This assumes odor can be measured accurately, of course. If your sense of smell is poor you may not want to bet your life on it. Frankly, you probably wouldn't want to bet your life on the results of mining data from a field guide. Nevertheless, it makes a nice example.

going to build a model to determine the mushroom edibility using only a *single* feature, you should choose its odor. If you were going to build a more complex model you might start with the attribute ODOR before considering adding others. In fact, this is exactly the topic of the next section.

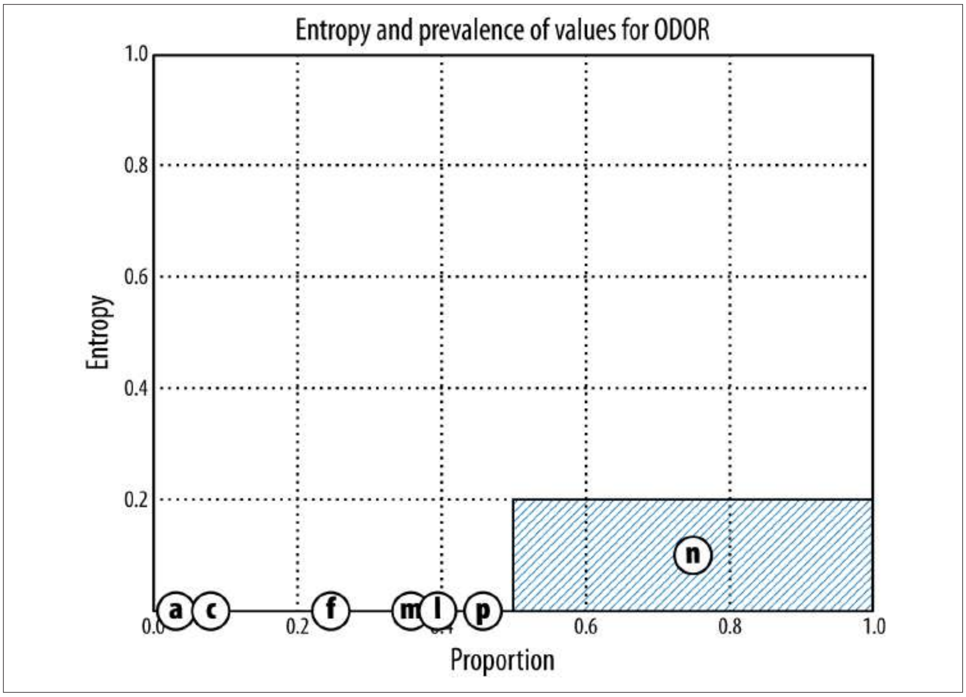


Figure 3-9. Entropy chart for the Mushroom dataset as split by ODOR. The amount of shading corresponds to the total (weighted sum) entropy, with each bar corresponding to the entropy of one of the attribute's values, and the width of the bar corresponding to the prevalence of that value in the data.

Supervised Segmentation with Tree-Structured Models

We have now introduced one of the fundamental ideas of data mining: finding informative attributes from the data. Let's continue on the topic of creating a supervised segmentation, because as important as it is, attribute selection alone does not seem to be sufficient. If we select the single variable that gives the most information gain, we create a very simple segmentation. If we select multiple attributes each giving some information gain, it's not clear how to put them together. Recall from earlier that we would like to create segments that use multiple attributes, such as "Middle-aged professionals who reside in New York City on average have a churn rate of 5%." We now introduce an elegant application of the ideas we've developed for selecting important attributes, to produce a multivariate (multiple attribute) supervised segmentation.

Consider a segmentation of the data to take the form of a "tree," such as that shown in Figure 3-10. In the figure, the tree is upside down with the root at the top. The tree is made up of *nodes*, interior nodes and terminal nodes, and branches emanating from the interior nodes. Each interior node in the tree contains a test of an attribute, with each branch from the node representing a distinct value of the attribute. Following the branches from the root node down (in the direction of the arrows), each path eventually terminates at a terminal node, or *leaf*. The tree creates a segmentation of the data: every data point will correspond to one and only one path in the tree, and thereby to one and only one leaf. In other words, each leaf corresponds to a segment, and the attributes and values along the path give the characteristics of the segment. So the rightmost path in the tree in Figure 3-10 corresponds to the segment "Older, unemployed people with high balances." The tree is a *supervised* segmentation, because each leaf contains a value for the target variable. Since we are talking about classification, here each leaf contains a classification for its segment. Such a tree is called a *classification tree* or more loosely a *decision tree*.

Classification trees often are used as predictive models—"tree structured models." In use, when presented with an example for which we do not know its classification, we can predict its classification by finding the corresponding segment and using the class value at the leaf. Mechanically, one would start at the root node and descend through the interior nodes, choosing branches based on the specific attribute values in the example. The nonleaf nodes are often referred to as "decision nodes," because when descending through the tree, at each node one uses the values of the attribute to make a decision about which branch to follow. Following these branches ultimately leads to a final decision about what class to predict: eventually a terminal node is reached, which gives a class prediction. In a tree, no two parents share descendants and there are no cycles; the branches always "point downwards" so that every example always ends up at a leaf node with some specific class determination.

Consider how we would use the classification tree in Figure 3-10 to classify an example of the person named Claudio from Figure 3-1. The values of Claudio's attributes are *Balance=115K*, *Employed=No*, and *Age=40*. We begin at the root node that tests *Employed*. Since the value is *No* we take the right branch. The next test is *Balance*. The value of *Balance* is 115K, which is greater than 50K so we take a right branch again to a node that tests *Age*. The value is 40 so we take the left branch. This brings us to a leaf node specifying *class=Not Write-off*, representing a prediction that Claudio will not default. Another way of saying this is that we have classified Claudio into a segment defined by *(Employed=No, Balance=115K, Age<45)* whose classification is *Not Write -off*.

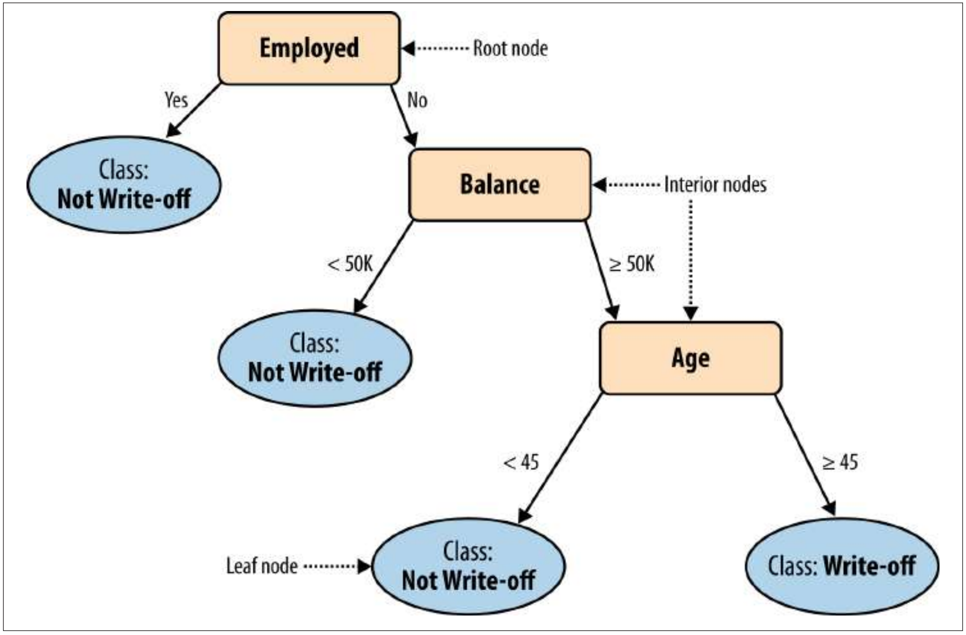


Figure 3-10. A simple classification tree.

Classification trees are one sort of tree-structured model. As we will see later, in business applications often we want to predict the probability of membership in the class (e.g., the probability of churn or the probability of write-off), rather than the class itself. In this case, the leaves of the *probability estimation tree* would contain these probabilities rather than a simple value. If the target variable is numeric, the leaves of the *regression tree* contain numeric values. However, the basic idea is the same for all.

Trees provide a model that can represent exactly the sort of supervised segmentation we often want, and we know how to use such a model to predict values for new cases (in "use"). However, we still have not addressed how to create such a model from the data. We turn to that now.

There are many techniques to induce a supervised segmentation from a dataset. One of the most popular is to create a tree-structured model (*tree induction*). These techniques are popular because tree models are easy to understand, and because the induction procedures are elegant (simple to describe) and easy to use. They are robust to many common data problems and are relatively efficient. Most data mining packages include some type of tree induction technique.

How do we create a classification tree from data? Combining the ideas introduced above, the goal of the tree is to provide a supervised segmentation—more specifically, to partition the instances, based on their attributes, into subgroups that have similar values

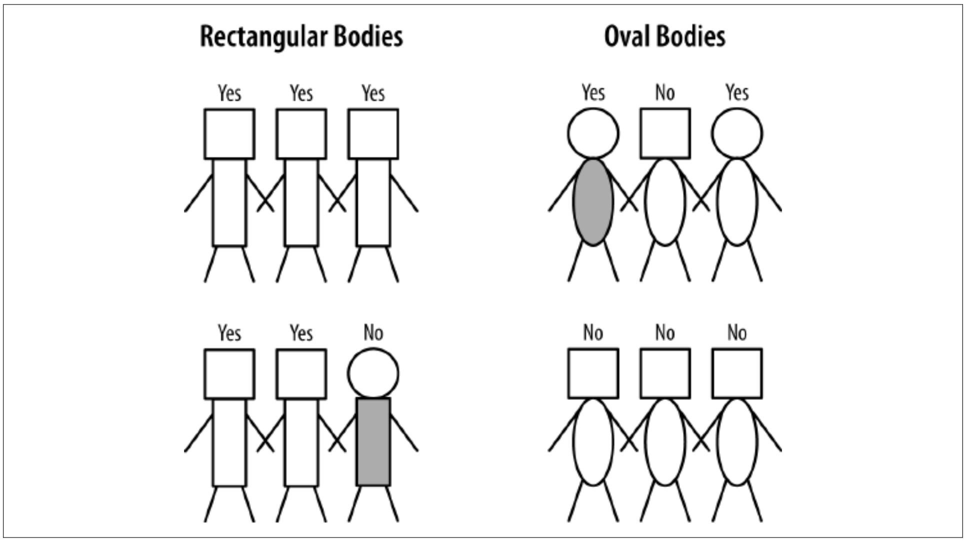


Figure 3-11. First partitioning: splitting on body shape (rectangular versus oval).

for their target variables. We would like for each "leaf" segment to contain instances that tend to belong to the same class.

To illustrate the process of classification tree induction, consider the very simple example set shown previously in Figure 3-2.

Tree induction takes a divide-and-conquer approach, starting with the whole dataset and applying variable selection to try to create the "purest" subgroups possible using the attributes. In the example, one way is to separate people based on their body type: rectangular versus oval. This creates the two groups shown in Figure 3-11. How good is this partitioning? The rectangular-body people on the left are mostly *Yes*, with a single *No* person, so it is mostly pure. The oval-body group on the right has mostly *No* people, but two *Yes* people. This step is simply a direct application of the attribute selection ideas presented above. Let's consider this "split" to be the one that yields the largest information gain.

Looking at Figure 3-11, we can now see the elegance of tree induction, and why it resonates well with so many people. The left and right subgroups are simply smaller versions of the problem with which we initially were faced! We can simply take each data subset and *recursively* apply attribute selection to find the best attribute to partition it. So in our example, we recursively consider the oval-body group (Figure 3-12). To split this group again we now consider another attribute: head shape. This splits the group in two on the right side of the figure. How good is this partitioning? Each new group has a single target label: four (square heads) of *No*, and two (round heads) of

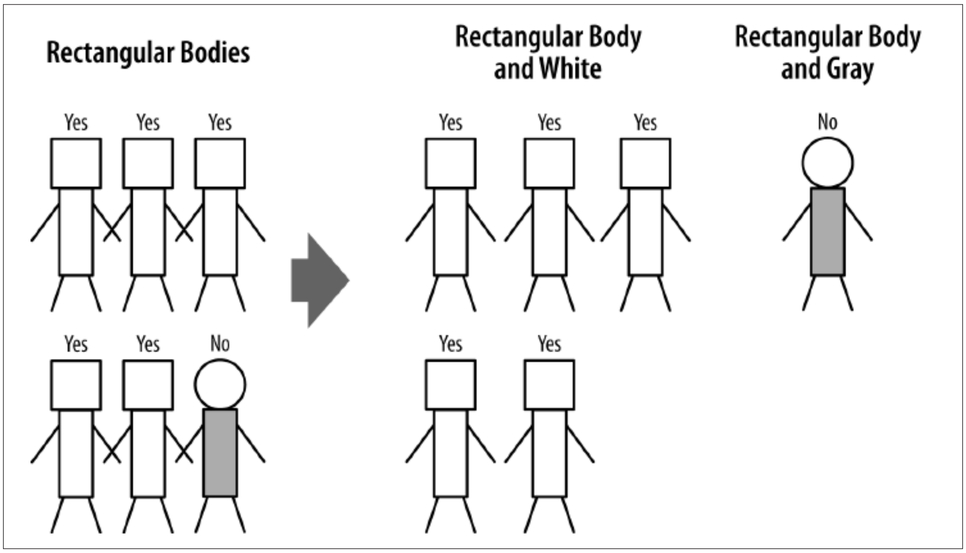


Figure 3-13. Third partitioning: the rectangular body people subgrouped by body color.

Yes. These groups are "maximally pure" with respect to class labels and there is no need to split them further.

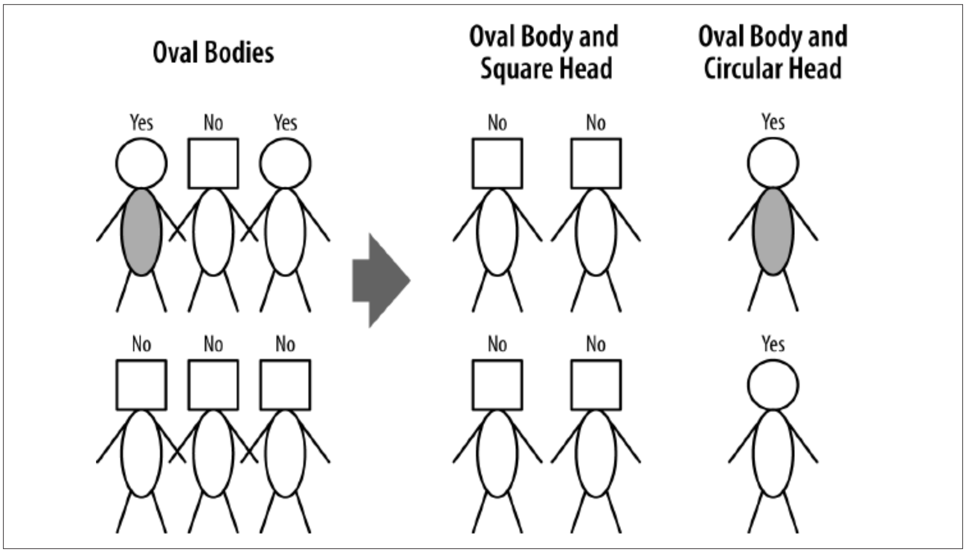


Figure 3-12. Second partitioning: the oval body people sub-grouped by head type.

We still have not done anything with the rectangular body group on the left side of Figure 3-11, so let's consider how to split them. There are five *Yes* people and one *No* person. There are two attributes we could split upon: head shape (square or round), and body color (white or gray). Either of these would work, so we arbitrarily choose body color. This produces the groupings in Figure 3-13. These are pure groups (all of one type) so we are finished. The classification tree corresponding to these groupings is shown in Figure 3-14.

In summary, the procedure of classification tree induction is a recursive process of divide and conquer, where the goal at each step is to select an attribute to partition the current group into subgroups that are as pure as possible with respect to the target variable. We perform this partitioning recursively, splitting further and further until we are done. We choose the attributes to split upon by testing all of them and selecting whichever yields the purest subgroups. When are we done? (In other words, when do we stop recursing?) It should be clear that we would stop when the nodes are pure, or when we run out of variables to split on. But we may want to stop earlier; we will return to this question in Chapter 5.

Visualizing Segmentations

Continuing with the metaphor of predictive model building as supervised segmentation, it is instructive to visualize exactly how a classification tree partitions the instance space. The instance space is simply the space described by the data features. A common form of instance space visualization is a scatterplot on some pair of features, used to compare one variable against another to detect correlations and relationships.

Though data may contain dozens or hundreds of variables, it is only really possible to visualize segmentations in two or three dimensions at once. Still, visualizing models in instance space in a few dimensions is useful for understanding the different *types* of models because it provides insights that apply to higher dimensional spaces as well. It may be difficult to compare very different families of models just by examining their form (e.g., a mathematical formula versus a set of rules) or the algorithms that generate them. Often it is easier to compare them based on how they partition the instance space.

For example, Figure 3-15 shows a simple classification tree next to a two-dimensional graph of the instance space: Balance on the *x* axis and Age on the *y* axis. The root node of the classification tree tests Balance against a threshold of 50K. In the graph, this corresponds to a vertical line at 50K on the *x* axis splitting the plane into Balance<50K and Balance≥50K. At the left of this line lie the instances whose Balance values are less than 50K; there are 13 examples of class Write-off (black dot) and 2 examples of class non-Write-off (plus sign) in this region.

On the right branch out of the root node are instances with Balance≥50K. The next node in the classification tree tests the Age attribute against the threshold 45. In the

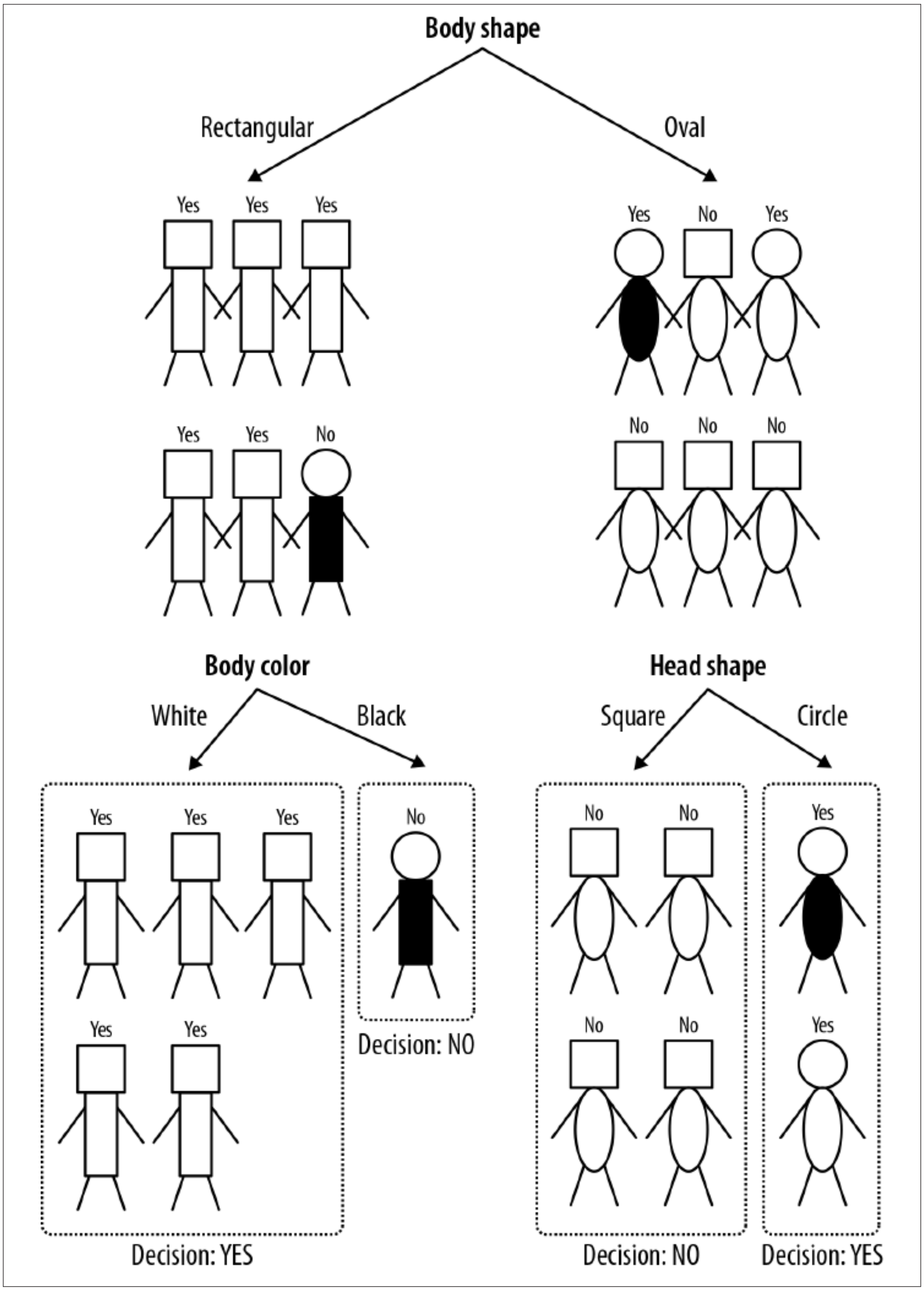


Figure 3-14. The classification tree resulting from the splits done in Figure 3-11 to Figure 3-13.

graph this corresponds to the horizontal dotted line at Age=45. It appears only on the right side of the graph because this partition only applies to examples with Balance≥50. The Age decision node assigns to its left branch instances with Age<45, corresponding to the lower right segment of the graph, representing: (Balance≥50K AND Age<45).

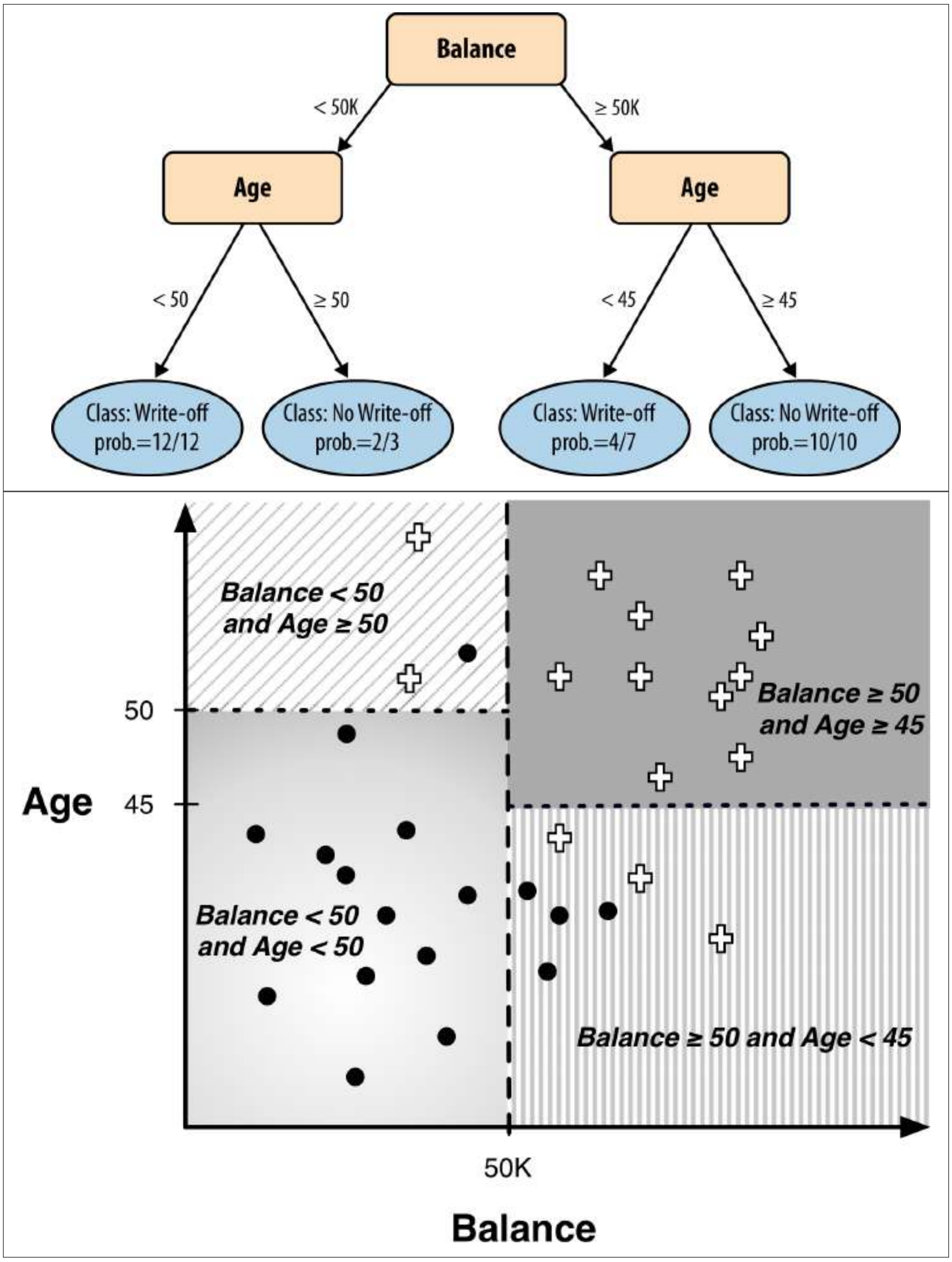
Notice that each internal (decision) node corresponds to a split of the instance space. Each leaf node corresponds to an unsplit region of the space (a segment of the population). Whenever we follow a path in the tree out of a decision node we are restricting attention to one of the two (or more) subregions defined by the split. As we descend through a classification tree we consider progressively more focused subregions of the instance space.

Decision lines and hyperplanes

The lines separating the regions are known as *decision lines* (in two dimensions) or more generally *decision surfaces* or *decision boundaries*. Each node of a classification tree tests a single variable against a fixed value so the decision boundary corresponding to it will always be perpendicular to the axis representing this variable. In two dimensions, the line will be either horizontal or vertical. If the data had three variables the instance space would be three-dimensional and each boundary surface imposed by a classification tree would be a two-dimensional plane. In higher dimensions, since each node of a classification tree tests one variable it may be thought of as "fixing" that one dimension of a decision boundary; therefore, for a problem of n variables, each node of a classification tree imposes an $(n-1)$-dimensional "hyperplane" decision boundary on the instance space.

You will often see the term *hyperplane* used in data mining literature to refer to the general separating surface, whatever it may be. Don't be intimidated by this terminology. You can always just think of it as a generalization of a line or a plane.

Other decision surfaces are possible, as we shall see later.



Balance
< 50K
≥ 50K
Age
Age
< 50
≥ 50
< 45
≥ 45
Class: Write-off prob.=12/12
Class: No Write-off prob.=2/3
Class: Write-off prob.=4/7
Class: No Write-off prob.=10/10
Balance < 50 and Age ≥ 50
Balance ≥ 50 and Age ≥ 45
Balance < 50 and Age < 50
Balance ≥ 50 and Age < 45
50
45
Age
50K
Balance

Figure 3-15. A classification tree and the partitions it imposes in instance space. The black dots correspond to instances of the class Write-off, the plus signs correspond to instances of class non-Write-off. The shading shows how the tree leaves correspond to segments of the population in instance space.

Trees as Sets of Rules

Before moving on from the interpretation of classification trees, we should mention their interpretation as logical statements. Consider again the tree shown at the top of Figure 3-15. You classify a new unseen instance by starting at the root node and following the attribute tests downward until you reach a leaf node, which specifies the instance's predicted class. If we trace down a single path from the root node to a leaf, collecting the conditions as we go, we generate a rule. Each rule consists of the attribute tests along the path connected with AND. Starting at the root node and choosing the left branches of the tree, we get the rule:

```
IF (Balance < 50K) AND (Age < 50) THEN Class=Write-off
```

We can do this for every possible path to a leaf node. From this tree we get three more rules:

```
IF (Balance < 50K) AND (Age ≥ 50) THEN Class=No Write-off
IF (Balance ≥ 50K) AND (Age < 45) THEN Class=Write-off
IF (Balance ≥ 50K) AND (Age < 45) THEN Class=No Write-off
```

The classification tree is equivalent to this rule set. If these rules look repetitive, that's because they are: the tree gathers common rule prefixes together toward the top of the tree. Every classification tree can be expressed as a set of rules this way. Whether the tree or the rule set is more intelligible is a matter of opinion; in this simple example, both are fairly easy to understand. As the model becomes larger, some people will prefer the tree or the rule set.

Probability Estimation

In many decision-making problems, we would like a more informative prediction than just a classification. For example, in our churn-prediction problem, rather than simply predicting whether a person will leave the company within 90 days of contract expiration, we would much rather have an estimate of the probability that he will leave the company within that time. Such estimates can be used for many purposes. We will discuss some of these in detail in later chapters, but briefly: you might then rank prospects by their probability of leaving, and then allocate a limited incentive budget to the highest probability instances. Alternatively, you may want to allocate your incentive budget to the instances with the highest expected loss, for which you'll need (an estimate of) the probability of churn. Once you have such probability estimates you can use them in a more sophisticated decision-making process than these simple examples, as we'll describe in later chapters.

There is another, even more insidious problem with models that give simple classifications, rather than estimates of class membership probability. Consider the problem of estimating credit default. Under normal circumstances, for just about any segment of the population to which we would be considering giving credit, the probability of write-off will be very small—far less than 0.5. In this case, when we build a model to estimate the classification (write-off or not), we'd have to say that for each segment, the members are likely not to default—and they will all get the same classification (not write-off). For example, in a naively built tree model every leaf will be labeled "not write-off." This turns out to be a frustrating experience for new data miners: after all that work, the model really just says that no one is likely to default? This does *not* mean that the model is useless. It may be that the different segments indeed have very different probabilities of write-off, they just all are less than 0.5. If instead we use these probabilities for assigning credit, we may be able reduce our risk substantially.

So, in the context of supervised segmentation, we would like each segment (leaf of a tree model) to be assigned an estimate of the probability of membership in the different classes. Figure 3-15 more generally shows a "probability estimation tree" model for our simple write-off prediction example, giving not only a prediction of the class but also the estimate of the probability of membership in the class.[6]

Fortunately, the tree induction ideas we have discussed so far can easily produce probability estimation trees instead of simple classification trees.[7] Recall that the tree induction procedure subdivides the instance space into regions of class purity (low entropy). If we are satisfied to assign the same class probability to every member of the segment corresponding to a tree leaf, we can use instance counts at each leaf to compute a class probability estimate. For example, if a leaf contains n positive instances and m negative instances, the probability of any new instance being positive may be estimated as $n/(n+m)$. This is called a *frequency-based* estimate of class membership probability.

At this point you may spot a problem with estimating class membership probabilities this way: we may be overly optimistic about the probability of class membership for segments with very small numbers of instances. At the extreme, if a leaf happens to have only a single instance, should we be willing to say that there is a 100% probability that members of that segment will have the class that this one instance happens to have?

6. We often deal with binary classification problems, such as write-off or not, or churn or not. In these cases it is typical just to report the probability of membership in one chosen class $p(c)$, because the other is just $1 - p(c)$.

7. Often these are still called classification trees, even if the decision maker intends to use the probability estimates rather than the simple classifications.

This phenomenon is one example of a fundamental issue in data science ("overfitting"), to which we devote a chapter later in the book. For completeness, let's quickly discuss one easy way to address this problem of small samples for tree-based class probability estimation. Instead of simply computing the frequency, we would often use a "smoothed" version of the frequency-based estimate, known as the Laplace correction, the purpose of which is to moderate the influence of leaves with only a few instances. The equation for binary class probability estimation becomes:

$$p(c) = \frac{n + 1}{n + m + 2}$$

where n is the number of examples in the leaf belonging to class c, and m is the number of examples not belonging to class c.

Let's walk through an example with and without the Laplace correction. A leaf node with two positive instances and no negative instances would produce the same frequency-based estimate ($p = 1$) as a leaf node with 20 positive instances and no negatives. However, the first leaf node has much less evidence and may be extreme only due to there being so few instances. Its estimate should be tempered by this consideration. The Laplace equation smooths its estimate down to $p = 0.75$ to reflect this uncertainty; the Laplace correction has much less effect on the leaf with 20 instances ($p \approx 0.95$). As the number of instances increases, the Laplace equation converges to the frequency-based estimate. Figure 3-16 shows the effect of Laplace correction on several class ratios as the number of instances increases (2/3, 4/5, and 1/1). For each ratio the solid horizontal line shows the uncorrected (constant) estimate, while the corresponding dashed line shows the estimate with the Laplace correction applied. The uncorrected line is the asymptote of the Laplace correction as the number of instances goes to infinity.

Example: Addressing the Churn Problem with Tree Induction

Now that we have a basic data mining technique for predictive modeling, let's consider the churn problem again. How could we use tree induction to help solve it?

For this example, we have a historical data set of 20,000 customers. At the point of collecting the data, each customer either had stayed with the company or had left (churned). Each customer is described by the variables listed in Table 3-2.

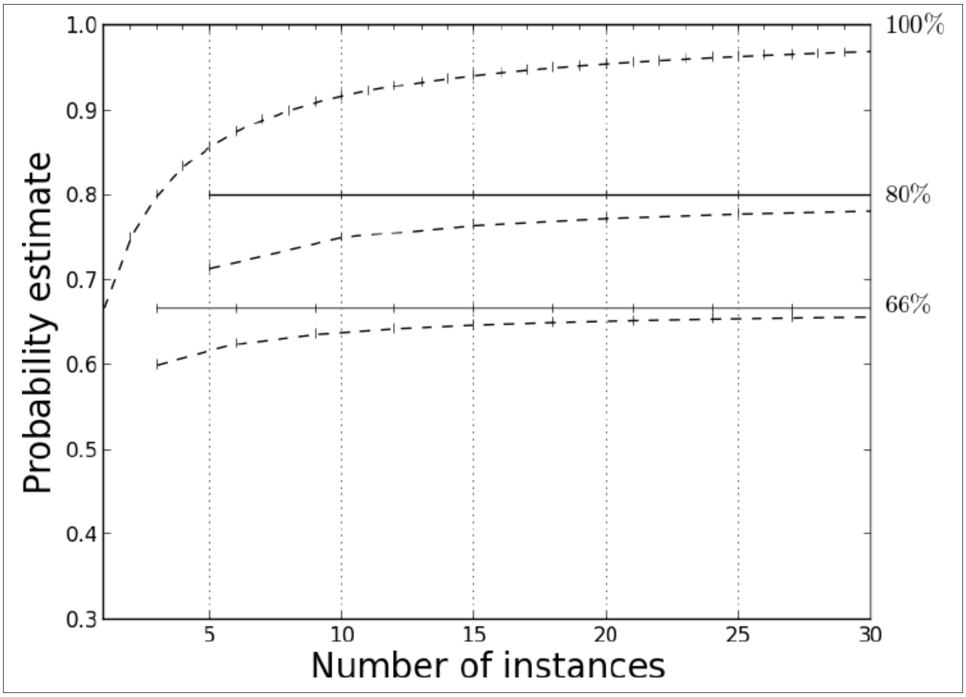


Figure 3-16. The effect of Laplace smoothing on probability estimation for several instance ratios.

Table 3-2. Attributes for the cellular phone churn-prediction problem

Variable	Explanation
COLLEGE	Is the customer college educated?
INCOME	Annual income
OVERAGE	Average overcharges per month
LEFTOVER	Average number of leftover minutes per month
HOUSE	Estimated value of dwelling (from census tract)
HANDSET_PRICE	Cost of phone
LONG_CALLS_PER_MONTH	Average number of long calls (15 mins or over) per month
AVERAGE_CALL_DURATION	Average duration of a call
REPORTED_SATISFACTION	Reported level of satisfaction
REPORTED_USAGE_LEVEL	Self-reported usage level
LEAVE *(Target variable)*	Did the customer stay or leave (churn)?

These variables comprise basic demographic and usage information available from the customer's application and account. We want to use these data with our tree induction technique to predict which new customers are going to churn.

Before starting to build a classification tree with these variables, it is worth asking, *How good are each of these variables individually?* For this we measure the information gain of each attribute, as discussed earlier. Specifically, we apply Equation 3-2 to each variable independently over the entire set of instances, to see what each gains us.

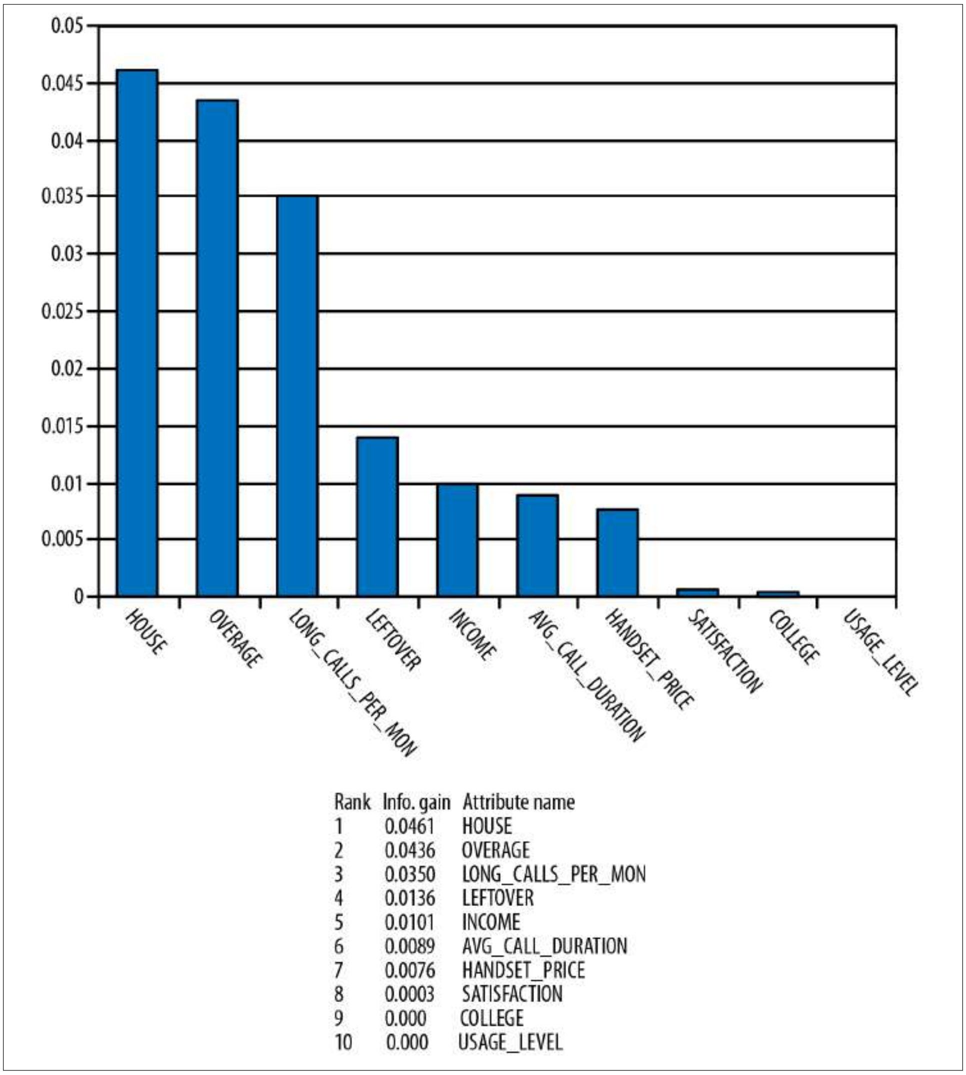
The results are in Figure 3-17, with a table listing the exact values. As you can see, the first three variables—the house value, the number of leftover minutes, and the number of long calls per month—have a higher information gain than the rest.[8] Perhaps surprisingly, neither the amount the phone is used nor the reported degree of satisfaction seems, in and of itself, to be very predictive of churning.

Applying a classification tree algorithm to the data, we get the tree shown in Figure 3-18. The highest information gain feature (HOUSE) according to Figure 3-17 is at the root of the tree. This is to be expected since it will always be chosen first. The second best feature, OVERAGE, also appears high in the tree. However, the order in which features are chosen for the tree doesn't exactly correspond to their ranking in Figure 3-17. Why is this?

The answer is that the table ranks each feature by how good it is *independently*, evaluated separately on the entire population of instances. Nodes in a classification tree depend on the instances above them in the tree. Therefore, except for the root node, features in a classification tree are not evaluated on the entire set of instances. The information gain of a feature depends on the set of instances against which it is evaluated, so the ranking of features for some internal node may not be the same as the global ranking.

We have not yet discussed how we decide to stop building the tree. The dataset has 20,000 examples yet the tree clearly doesn't have 20,000 leaf nodes. Can't we just keep selecting more attributes to split upon, building the tree downwards until we've exhausted the data? The answer is yes, we can, but we should stop long before the model becomes that complex. This issue ties in closely with model generality and overfitting, whose discussion we defer to Chapter 5.

8. Note that the information gains for the attributes in this churn data set are much smaller than those shown previously for the mushroom data set.



Rank	Info. gain	Attribute name
1	0.0461	HOUSE
2	0.0436	OVERAGE
3	0.0350	LONG_CALLS_PER_MON
4	0.0136	LEFTOVER
5	0.0101	INCOME
6	0.0089	AVG_CALL_DURATION
7	0.0076	HANDSET_PRICE
8	0.0003	SATISFACTION
9	0.000	COLLEGE
10	0.000	USAGE_LEVEL

Figure 3-17. Churn attributes from Table 3-2 ranked by information gain.

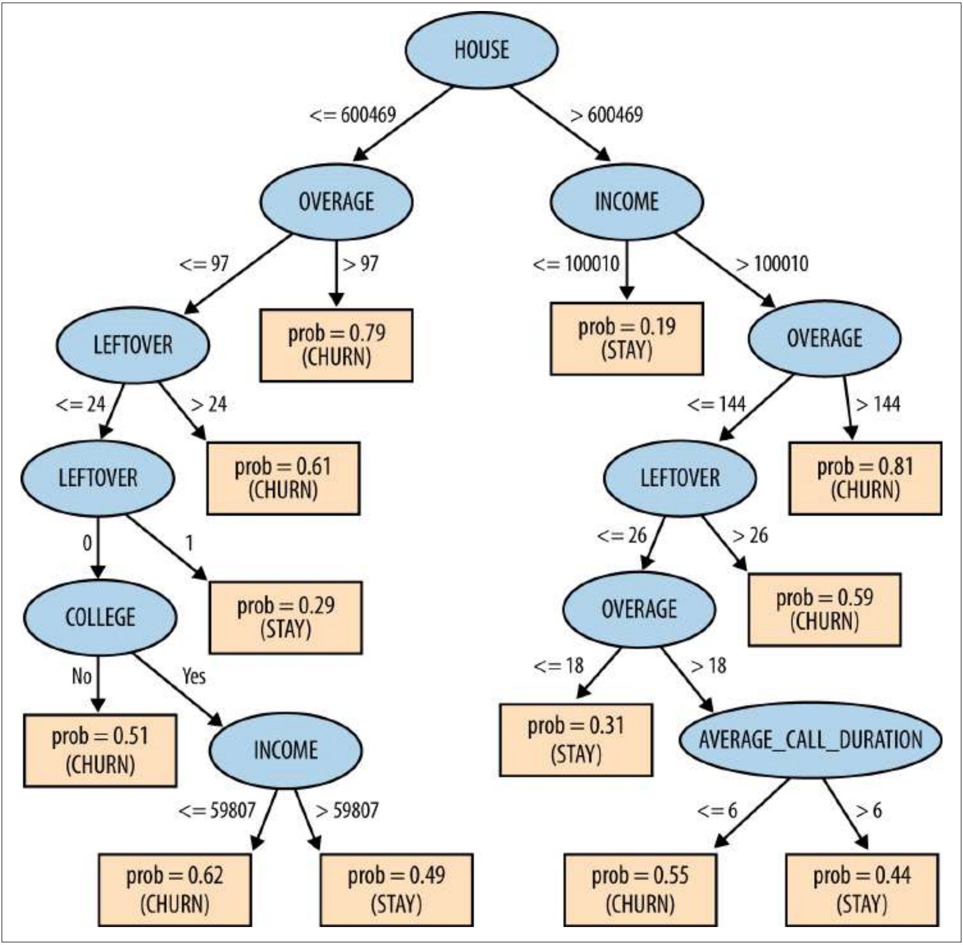


Figure 3-18. Classification tree learned from the cellular phone churn data.

Consider a final issue with this dataset. After building a tree model from the data, we measured its accuracy against the data to see how good of a model it is. Specifically, we used a training set consisting half of people who churned and the other half who did not; after learning a classification tree from this, we applied the tree to the dataset to see how many of the examples it could classify correctly. The tree achieved 73% accuracy on its decisions. This raises two questions:

1. First, do you trust this number? If we applied the tree to another sample of 20,000 people from the same dataset, do you think we'd still get about 73% accuracy?
2. If you *do* trust the number, does it mean this model is good? In other words, is a model with 73% accuracy worth using?

We will revisit these questions in Chapter 7 and Chapter 8, which delve into issues of model evaluation.

Summary

In this chapter, we introduced basic concepts of predictive modeling, one of the main tasks of data science, in which a model is built that can estimate the value of a target variable for a new unseen example. In the process, we introduced one of data science's fundamental notions: finding and selecting informative attributes. Selecting informative attributes can be a useful data mining procedure in and of itself. Given a large collection of data, we now can find those variables that correlate with or give us information about another variable of interest. For example, if we gather historical data on which customers have or have not left the company (churned) shortly after their contracts expire, attribute selection can find demographic or account-oriented variables that provide information about the likelihood of customers churning. One basic measure of attribute information is called *information gain*, which is based on a purity measure called *entropy*; another is variance reduction.

Selecting informative attributes forms the basis of a common modeling technique called tree induction. Tree induction recursively finds informative attributes for subsets of the data. In so doing it segments the space of instances into similar regions. The partitioning is "supervised" in that it tries to find segments that give increasingly precise information about the quantity to be predicted, the target. The resulting tree-structured model partitions the space of all possible instances into a set of segments with different predicted values for the target. For example, when the target is a binary "class" variable such as churn versus not churn, or write-off versus not write-off, each leaf of the tree corresponds to a population segment with a different estimated probability of class membership.

As an exercise, think about what would be different in building a tree-structured model for regression rather than for classification. What would need to be changed from what you've learned about classification tree induction?

Historically, tree induction has been a very popular data mining procedure because it is easy to understand, easy to implement, and computationally inexpensive. Research on tree induction goes back at least to the 1950s and 1960s. Some of the earliest popular tree induction systems include CHAID (Chi-squared Automatic Interaction Detection) (Kass, 1980) and CART (Classification and Regression Trees) (Breiman, Friedman, Olshen, & Stone, 1984), which are still widely used. C4.5 and C5.0 are also very popular tree induction algorithms, which have a notable lineage (Quinlan, 1986, 1993). J48 is a reimplementation of C4.5 in the Weka package (Witten & Frank, 2000; Hall et al., 2001).

In practice, tree-structured models work remarkably well, though they may not be the most accurate model one can produce from a particular data set. In many cases, especially early in the application of data mining, it is important that models be understood and explained easily. This can be useful not just for the data science team but for communicating results to stakeholders not knowledgeable about data mining.

CHAPTER 4
Fitting a Model to Data

Fundamental concepts: *Finding "optimal" model parameters based on data; Choosing the goal for data mining; Objective functions; Loss functions.*

Exemplary techniques: *Linear regression; Logistic regression; Support-vector machines.*

As we have seen, predictive modeling involves finding a model of the target variable in terms of other descriptive attributes. In Chapter 3, we constructed a supervised segmentation model by recursively finding informative attributes on ever-more-precise subsets of the set of all instances, or from the geometric perspective, ever-more-precise subregions of the instance space. From the data we produced both the structure of the model (the particular tree model that resulted from the tree induction) and the numeric "parameters" of the model (the probability estimates at the leaf nodes).

An alternative method for learning a predictive model from a dataset is to start by specifying the structure of the model with certain numeric parameters left unspecified. Then the data mining calculates the best parameter values given a particular set of training data. A very common case is where the structure of the model is a parameterized mathematical function or equation of a set of numeric attributes. The attributes used in the model could be chosen based on domain knowledge regarding which attributes ought to be informative in predicting the target variable, or they could be chosen based on other data mining techniques, such as the attribute selection procedures introduced in Chapter 3. The data miner specifies the form of the model and the attributes; the goal of the data mining is to tune the parameters so that the model fits the data as well as possible. This general approach is called *parameter learning* or *parametric modeling*.

In certain fields of statistics and econometrics, the bare model with unspecified parameters is called "the model." We will clarify that this is the structure of the model, which still needs to have its parameters specified to be useful.

Many data mining procedures fall within this general framework. We will illustrate with some of the most common, all of which are based on *linear* models. If you've taken a statistics course, you're probably already familiar with one linear modeling technique: linear regression. We will see the same differences in models that we've seen already, such as the differences in task between classification, class probability estimation, and regression. As examples we will present some common techniques used for predicting (estimating) unknown numeric values, unknown binary values (such as whether a document or web page is relevant to a query), as well as likelihoods of events, such as default on credit, response to an offer, fraud on an account, and so on.

We also will explicitly discuss something that we skirted in Chapter 3: what exactly do we mean when we say a model fits the data well? This is the crux of the fundamental concept of this chapter—fitting a model to data by finding "optimal" model parameters —and is a notion that will resurface in later chapters. Because of its fundamental concepts, this chapter is more mathematically focused than the rest. We will keep the math to a minimum, and encourage the less mathematical reader to proceed boldly.

Sidebar: Simplifying Assumptions in This Chapter

The point of this chapter is to introduce and explain parametric modeling. To keep the discussion focused, and to avoid excessive footnotes, we've made some simplifying assumptions:

- First, for classification and class probability estimation we will consider only binary classes: the models predict events that either take place or do not, such as responding to an offer, leaving the company, being defrauded, etc. The methods here can all be generalized to work with multiple (nonbinary) classes, but the generalization complicates the description unnecessarily.
- Second, because we're dealing with equations, this chapter assumes all attributes are numeric. There are techniques for converting categorical (symbolic) attributes into numerical values for use with these equations.
- Finally, we ignore the need to normalize numeric measurements to a common scale. Attributes such as Age and Income have vastly different ranges and they are usually normalized to a common scale to help with model interpretability, as well as other things (to be discussed later).

We ignore these complications in this chapter. However, dealing with them is ultimately important and often necessary regardless of the data mining technique.

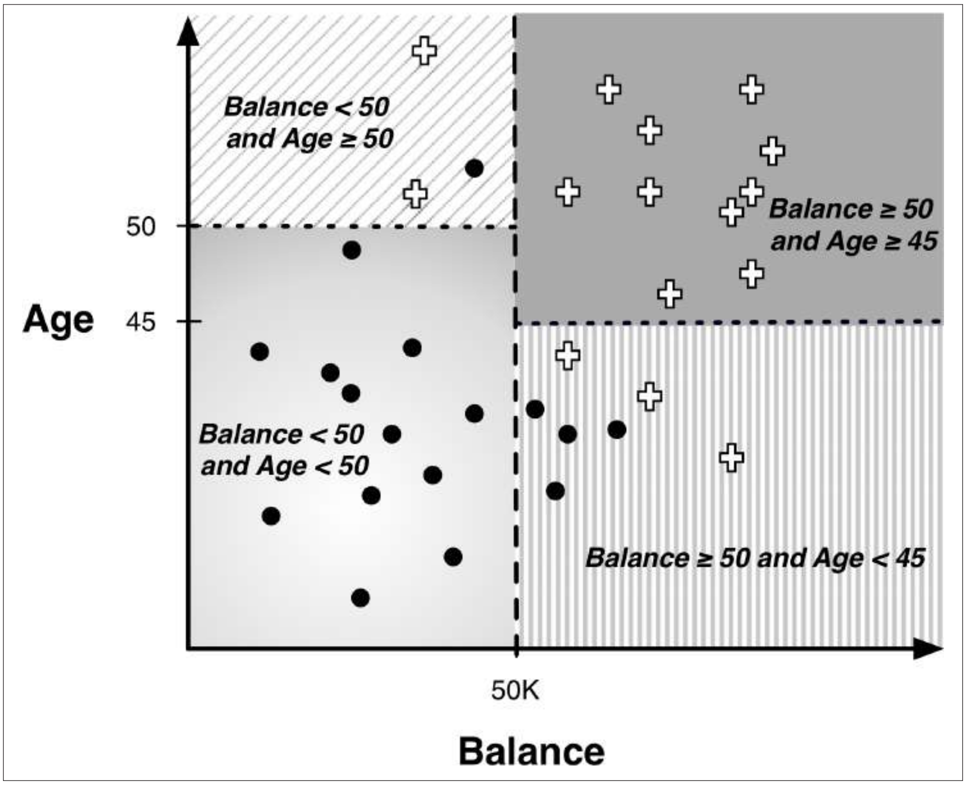


Figure 4-1. A dataset split by a classification tree with four leaf nodes.

Classification via Mathematical Functions

Recall the instance-space view of tree models from Chapter 3. One such diagram is replicated in Figure 4-1. It shows the space broken up into regions by horizontal and vertical *decision boundaries* that partition the instance space into similar regions. Examples in each region should have similar values for the target variable. In the last chapter we saw how the entropy measure gives us a way of measuring homogeneity so we can choose such boundaries.

A main purpose of creating homogeneous regions is so that we can predict the target variable of a new, unseen instance by determining which segment it falls into. For example, in Figure 4-1, if a new customer falls into the lower-left segment, we can conclude that the target value is very likely to be "•". Similarly, if it falls into the upper-right segment, we can predict its value as "+".

The instance-space view is helpful because if we take away the axis-parallel boundaries (see Figure 4-2) we can see that there clearly are other, possibly better, ways to partition

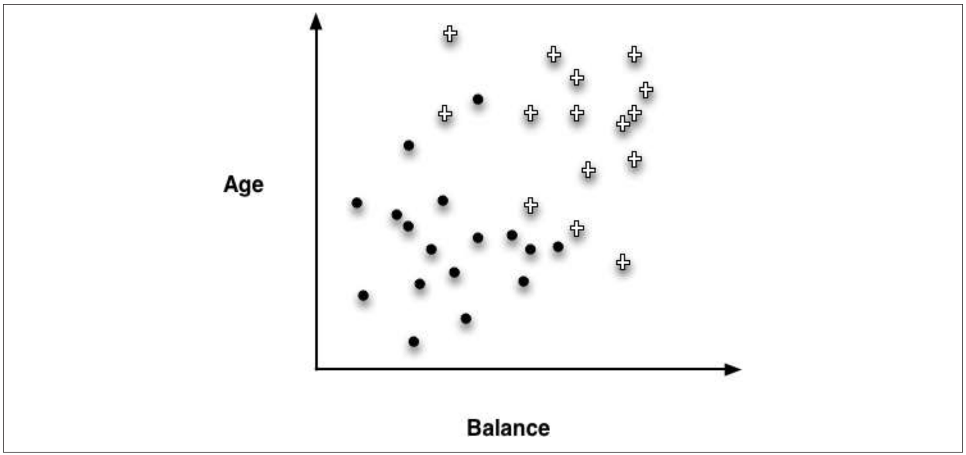


Figure 4-2. The raw data points of Figure 4-1, without decision lines.

the space. For example, we can separate the instances almost perfectly (by class) if we are allowed to introduce a boundary that is still a straight line, but is not perpendicular to the axes (Figure 4-3).

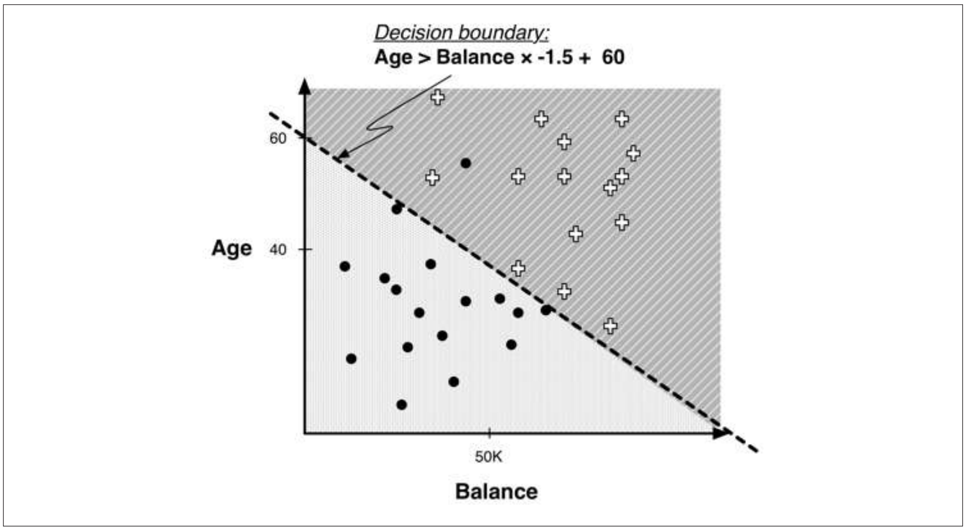


Figure 4-3. The dataset of Figure 4-2 with a single linear split.

This is called a *linear* classifier and is essentially a weighted sum of the values for the various attributes, as we will describe next.

Linear Discriminant Functions

Our goal is going to be to fit our model to the data, and to do so it is quite helpful to represent the model mathematically. You may recall that the equation of a line in two dimensions is $y = mx + b$, where m is the slope of the line and b is the y intercept (the y value when $x = 0$). The line in Figure 4-3 can be expressed in this form (with Balance in thousands) as:

$$Age = (-1.5) \times Balance + 60$$

We would classify an instance **x** as a + if it is above the line, and as a • if it is below the line. Rearranging this mathematically leads to the function that is the basis of all the techniques discussed in this chapter. First, for this example form the classification solution is shown in Equation 4-1.

Equation 4-1. Classification function

$$class(\mathbf{x}) = \begin{cases} + \text{ if } 1.0 \times Age - 1.5 \times Balance + 60 > 0 \\ \bullet \text{ if } 1.0 \times Age - 1.5 \times Balance + 60 \leq 0 \end{cases}$$

This is called a *linear discriminant* because it discriminates between the classes, and the function of the decision boundary is a linear combination—a weighted sum—of the attributes. In the two dimensions of our example, the linear combination corresponds to a line. In three dimensions, the decision boundary is a plane, and in higher dimensions it is a *hyperplane* (see Decision lines and hyperplanes in "Visualizing Segmentations" on page 67). For our purposes, the important thing is that we can express the model as a weighted sum of the attribute values.

Thus, this linear model is a different sort of multivariate supervised segmentation. Our goal with supervised segmentation still is to separate the data into regions with different values of the target variable. The difference is that the method for taking multiple attributes into account is to create a mathematical function of them.

In "Trees as Sets of Rules" on page 71 we showed how a classification tree corresponds to a rule set—a logical classification model of the data. A linear discriminant function is a numeric classification model. For example, consider our feature vector **x**, with the individual component features being x_i. A linear model then can be written as follows in Equation 4-2.

Equation 4-2. A general linear model

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots$$

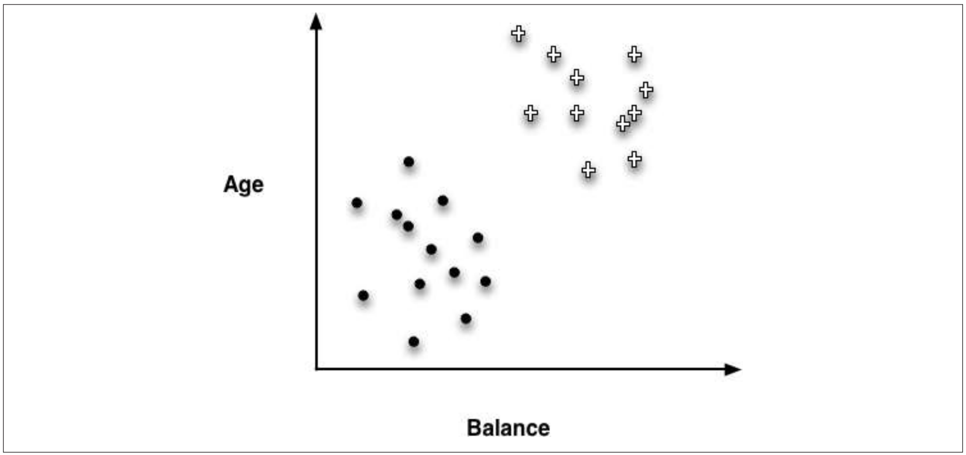


Figure 4-4. A basic instance space in two dimensions containing points of two classes.

The concrete example from Equation 4-1 can be written in this form:

$$f(\mathbf{x}) = 60 + 1.0 \times Age - 1.5 \times Balance$$

To use this model as a linear discriminant, for a given instance represented by a feature vector **x**, we check whether $f(\mathbf{x})$ is positive or negative. As discussed above, in the two-dimensional case, this corresponds to seeing whether the instance **x** falls above or below the line.

Linear functions are one of the workhorses of data science; now we finally come to the data mining. We now have a *parameterized* model: the weights of the linear function (w_i) are the parameters.[1] The data mining is going to "fit" this parameterized model to a particular dataset—meaning specifically, to find a good set of weights on the features.

After learning, these weights are often loosely interpreted as importance indicators of the features. Roughly, the larger the magnitude of a feature's weight, the more important that feature is for classifying the target (recalling the assumptions discussed earlier). By the same token, if a feature's weight is near zero the corresponding feature can usually be ignored or discarded. For now, we are interested in a set of weights that discriminate the training data well and predict as accurately as possible the value of the target variable for cases where we don't know it.

1. In order that the line need not go through the origin, it is typical to include the weight w_0, which is the intercept.

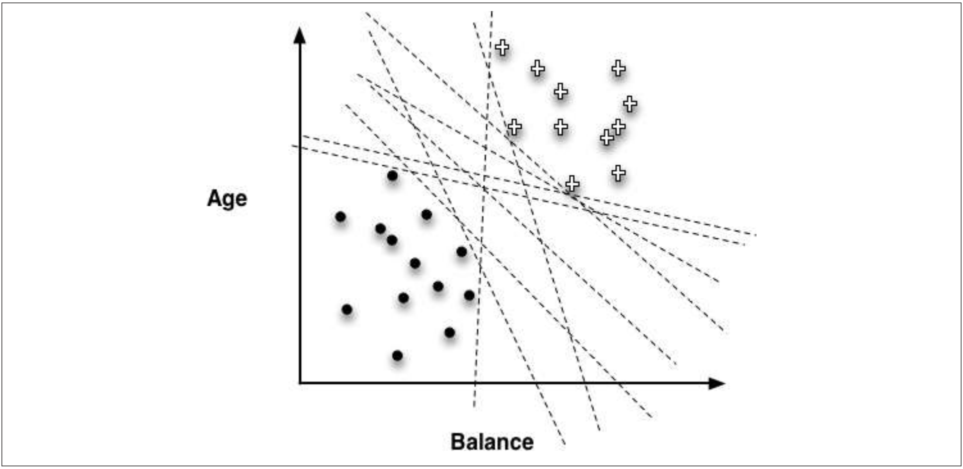


Figure 4-5. Many different possible linear boundaries can separate the two groups of points of Figure 4-4.

Unfortunately, it's not trivial to choose the "best" line to separate the classes. Let's consider a simple case, illustrated in Figure 4-4. Here the training data can indeed be separated by class using a linear discriminant. However, as shown in Figure 4-5, there actually are many different linear discriminants that can separate the classes perfectly. They have very different slopes and intercepts, and each represents a different model of the data. In fact, there are infinitely many lines (models) that classify this training set perfectly. Which should we pick?

Optimizing an Objective Function

This brings us to one of the most important fundamental ideas in data mining—one that surprisingly is often overlooked even by data scientists themselves: we need to ask, what should be our goal or *objective* in choosing the parameters? In our case, this would allow us to answer the question: what weights should we choose? Our general procedure will be to define an *objective function* that represents our goal, and can be calculated for a particular set of weights and a particular set of data. We will then find the optimal value for the weights by maximizing or minimizing the objective function. What can easily be overlooked is that these weights are "best" only if we believe that the objective function truly represents what we want to achieve, or practically speaking, is the best proxy we can come up with. We will return to this later in the book.

Unfortunately, creating an objective function that matches the true goal of the data mining is usually impossible, so data scientists often choose based on faith[2] and expe-

2. And sometimes it can be surprisingly hard for them to admit it.

rience. Several choices have been shown to be remarkably effective. One of these choices creates the so-called "support vector machine," about which we will say a few words after presenting a concrete example with a simpler objective function. After that, we will briefly discuss linear models for regression, rather than classification, and end with one of the most useful data mining techniques of all: *logistic regression*. Its name is something of a misnomer—logistic regression doesn't really do what we call regression, which is the estimation of a numeric target value. Logistic regression applies linear models to class probability estimation, which is particularly useful for many applications.

Linear regression, logistic regression, and support vector machines are all very similar instances of our basic fundamental technique: fitting a (linear) model to data. The key difference is that each uses a different objective function.

An Example of Mining a Linear Discriminant from Data

To illustrate linear discriminant functions, we use an adaptation of the Iris dataset taken from the UCI Dataset Repository (Bache & Lichman, 2013). This is an old and fairly simple dataset representing various types of iris, a genus of flowering plant. The original dataset includes three species of irises represented with four attributes, and the data mining problem is to classify each instance as belonging to one of the three species based on the attributes.

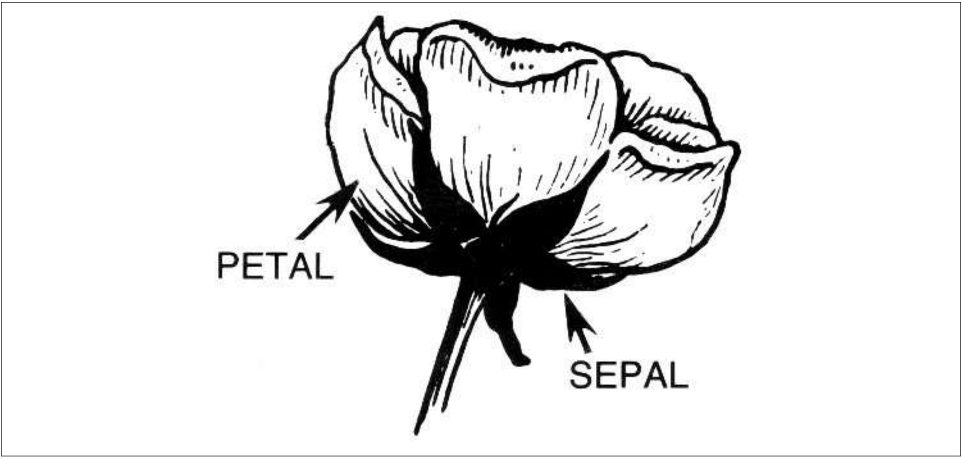


Figure 4-6. Two parts of a flower. Width measurements of these are used in the Iris dataset.

For this illustration we'll use just two species of irises, *Iris Setosa* and *Iris Versicolor*. The dataset describes a collection of flowers of these two species, each described with two measurements: the Petal width and the Sepal width (Figure 4-6). The flower dataset is plotted in Figure 4-7, with these two attributes on the *x* and *y* axis, respectively. Each instance is one flower and corresponds to one dot on the graph. The filled dots are of the species *Iris Setosa* and the circles are instances of the species *Iris Versicolor*.

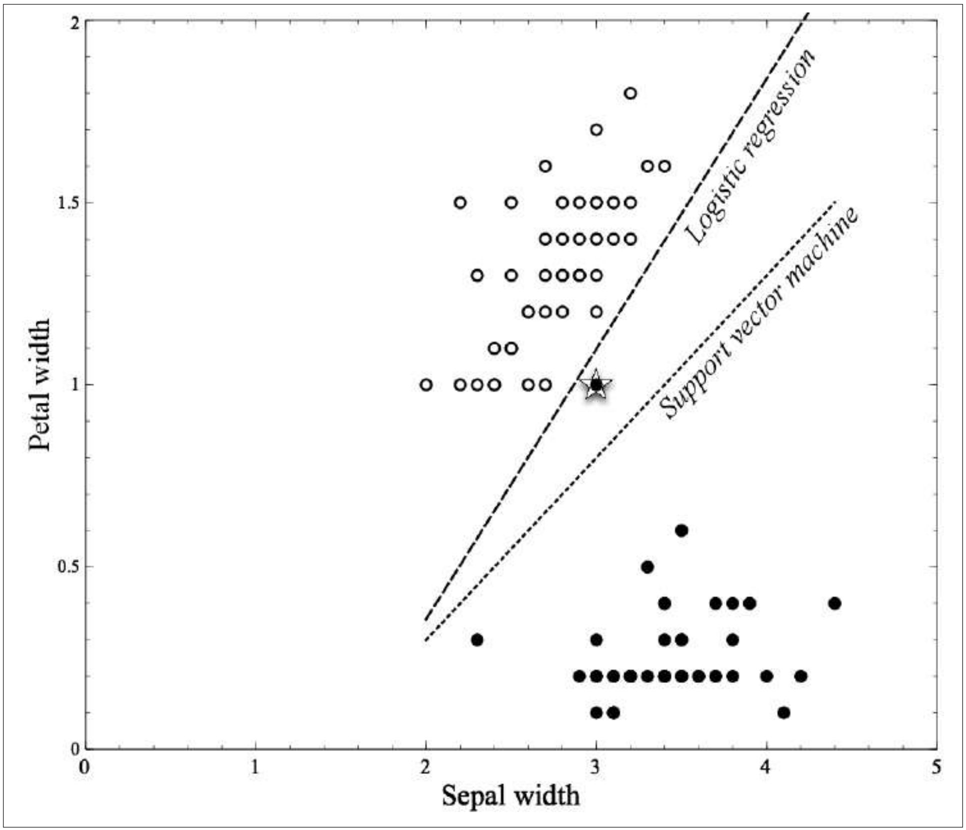


Figure 4-7. A dataset and two learned linear classifiers.

Two different separation lines are shown in the figure, one generated by logistic regression and the second by another linear method, a support vector machine (which will be described shortly). Note that the data comprise two fairly distinct clumps, with a few outliers. Logistic regression separates the two classes completely: all the *Iris Versicolor* examples are to the left of its line and all the *Iris Setosa* to the right. The Support vector machine line is almost midway between the clumps, though it misclassifies the starred point at (3, 1). Which separator do you think is better? In Chapter 5, we will get into details of why these separators are different and why one might be preferable to the other. For now it's enough just to notice that the methods produce different boundaries because they're optimizing different functions.

Linear Discriminant Functions for Scoring and Ranking Instances

In many applications, we don't simply want a *yes* or *no* prediction of whether an instance belongs to the class, but we want some notion of which examples are more or less likely to belong to the class. For example, which consumers are most likely to respond to this offer? Which customers are most likely to leave when their contracts expire? One option is to build a model that produces an estimate of class membership probability, as we did with tree induction for class probability estimation in Chapter 3. We can do this with linear models as well, and will treat this in detail below when we introduce logistic regression.

In other applications, we do not need a precise probability estimate. We simply need a score that will rank cases by the likelihood of belonging to one class or the other. For example, for targeted marketing we may have a limited budget for targeting prospective customers. We would like to have a list of consumers ranked by their predicted likelihood of responding positively to our offer. We don't necessarily need to be able to estimate the exact probability of response accurately, as long as the list is ranked reasonably well, and the consumers at the top of the list are the ones most likely to respond.

Linear discriminant functions can give us such a ranking for free. Look at Figure 4-4, and consider the + instances to be responders and • instances to be nonresponders. Assume we are presented with a new instance **x** for which we do not yet know the class (i.e., we have not yet made an offer to **x**). In which portion of the instance space would we like **x** to fall in order to expect the highest likelihood of response? Where would we be most certain that **x** would *not* respond? Where would we be most *un*certain?

Many people suspect that right near the decision boundary we would be most uncertain about a class (and see the discussion below on the "margin"). Far away from the decision boundary, on the + side would be where we would expect the highest likelihood of response. In the equation of the separating boundary, given above in Equation 4-2, $f(\mathbf{x})$ will be zero when **x** is sitting on the decision boundary (technically, **x** in that case is one of the points of the line or hyperplane). $f(\mathbf{x})$ will be relatively small when **x** is near the boundary. And $f(\mathbf{x})$ will be large (and positive) when **x** is far from the boundary in the + direction. Thus $f(\mathbf{x})$ itself—the output of the linear discriminant function—gives an intuitively satisfying ranking of the instances by their (estimated) likelihood of belonging to the class of interest.

Support Vector Machines, Briefly

If you're even on the periphery of the world of data science these days, you eventually will run into the *support vector machine* or "SVM." This is a notion that can strike fear into the hearts even of people quite knowledgeable in data science. Not only is the name itself opaque, but the method often is imbued with the sort of magic that derives from perceived effectiveness without understanding.

Fortunately, we now have the concepts necessary to understand support vector machines. In short, support vector machines are linear discriminants. For many business users interacting with data scientists, that will be sufficient. Nevertheless, let's look at SVMs a little more carefully; if we can get through some minor details, the procedure for fitting the linear discriminant is intuitively satisfying.

As with linear discriminants generally, SVMs classify instances based on a linear function of the features, described above in Equation 4-2.

You may also hear of *nonlinear* support vector machines. Oversimplifying slightly, a nonlinear SVM uses different features (that are functions of the original features), so that the linear discriminant with the new features is a nonlinear discriminant with the original features.

So, as we've discussed, the crucial question becomes: what is the objective function that is used to fit an SVM to data? For now we will skip the mathematical details in order to gain an intuitive understanding. There are two main ideas.

Recall Figure 4-5 showing the infinitude of different possible linear discriminants that would separate the classes, and recall that choosing an objective function for fitting the data amounts to choosing which of these lines is the best. SVMs choose based on a simple, elegant idea: instead of thinking about separating with a line, first fit the fattest bar between the classes. This is shown by the parallel dashed lines in Figure 4-8.

The SVM's objective function incorporates the idea that a wider bar is better. Then once the widest bar is found, the linear discriminant will be the center line through the bar (the solid middle line in Figure 4-8). The distance between the dashed parallel lines is called the *margin* around the linear discriminant, and thus the objective is to maximize the margin.

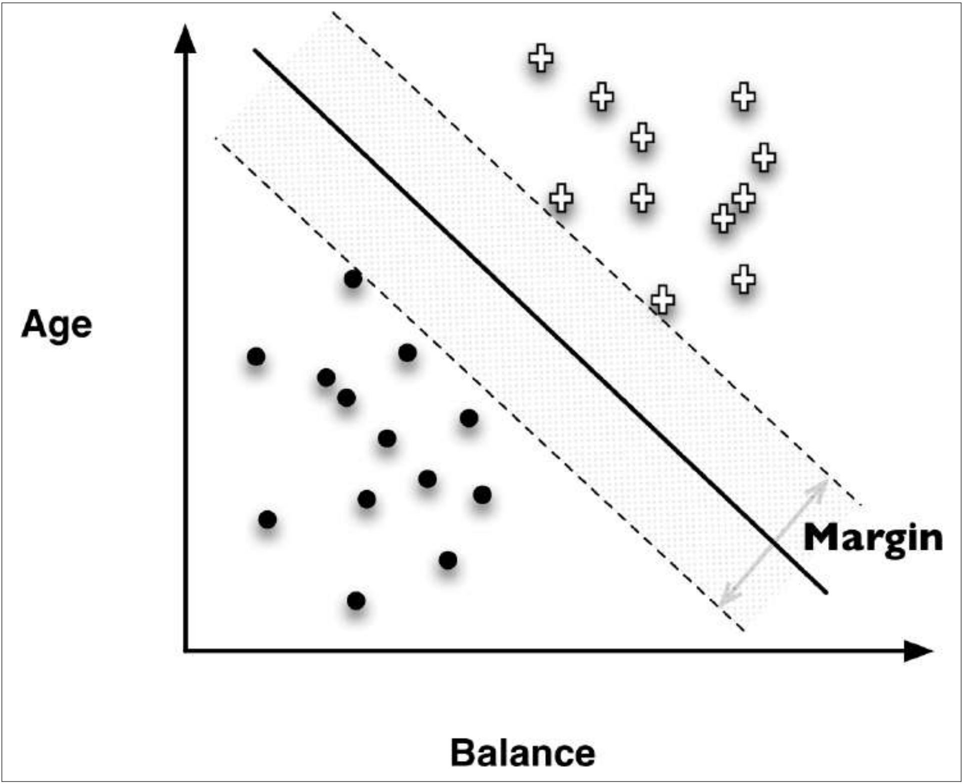


Figure 4-8. The points of Figure 4-2 and the maximal margin classifier.

The idea of maximizing the margin is intuitively satisfying for the following reason. The training dataset is just a sample from some population. In predictive modeling, we are interested in predicting the target for instances that we have not yet seen. These instances will be scattered about. Hopefully they will be distributed similarly to the training data, but they will in fact be different points. In particular, some of the positive examples will likely fall closer to the discriminant boundary than any positive example we have yet seen. All else being equal, the same applies to the negative examples. In other words, they may fall in the margin. The margin-maximizing boundary gives the maximal leeway for classifying such points. Specifically, by choosing the SVM decision boundary, in order for a new instance to be misclassified, one would have to place it further into the margin than with any other linear discriminant. (Or, of course, completely on the wrong side of the margin bar altogether.)

The second important idea of SVMs lies in how they handle points falling on the wrong side of the discrimination boundary. The original example of Figure 4-2 shows a situation in which a single line cannot perfectly separate the data into classes. This is true of most data from complex real-world applications—some data points will inevitably be misclassified by the model. This does not pose a problem for the general notion of linear discriminants, as their classifications don't necessarily have to be correct for all points. However, when fitting the linear function to the data we cannot simply ask which of all the lines that separate the data perfectly should we choose. There may be no such perfect separating line!

Once again, the support-vector machine's solution is intuitively satisfying. Skipping the math, the idea is as follows. In the objective function that measures how well a particular model fits the training points, we will simply penalize a training point for being on the wrong side of the decision boundary. In the case where the data indeed are linearly separable, we incur no penalty and simply maximize the margin. If the data are *not* linearly separable, the best fit is some balance between a fat margin and a low total error penalty. The penalty for a misclassified point is proportional to the distance from the decision boundary, so if possible the SVM will make only "small" errors. Technically, this error function is known as *hinge loss* (see "Sidebar: Loss functions" on page 94 and Figure 4-9).

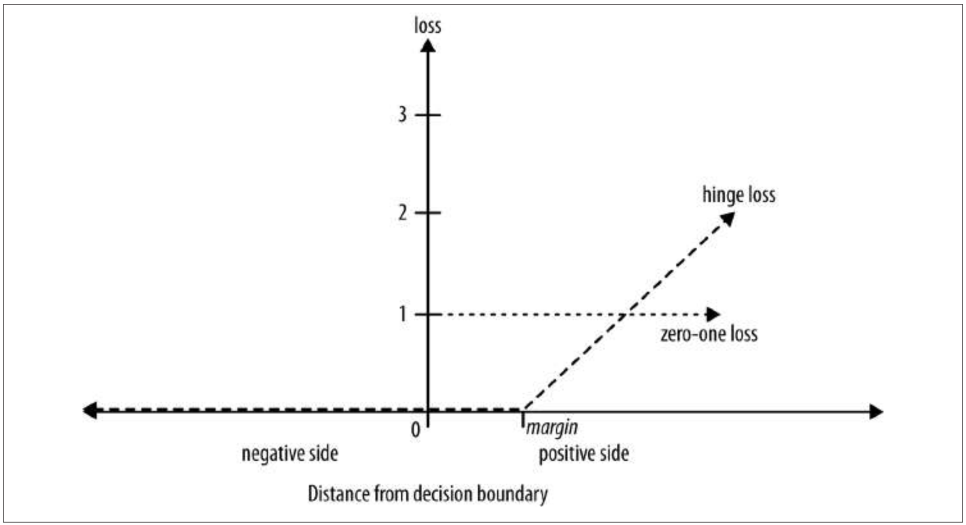


Figure 4-9. Two loss functions illustrated. The x axis shows the distance from the decision boundary. The y axis shows the loss incurred by a negative instance as a function of its distance from the decision boundary. (The case of a positive instance is symmetric.) If the negative instance is on the negative side of the boundary, there is no loss. If it is on the positive (wrong) side of the boundary, the different loss functions penalize it differently. (See "Sidebar: Loss functions" on page 94.)

Regression via Mathematical Functions

The previous chapter introduced the fundamental notion of selecting informative variables. We showed that this notion applies to classification, to regression, and to class probability estimation. Here too, this chapter's basic notion of fitting linear functions to data applies to classification, regression, and to class probability estimation. Let's now discuss regression briefly.[3]

Sidebar: Loss functions

The term "loss" is used across data science as a general term for error penalty. A loss function determines how much penalty should be assigned to an instance based on the error in the model's predicted value—in our present context, based on its distance from the separation boundary. Several loss functions are commonly used (two are shown in Figure 4-9). In the figure, the horizontal axis is the distance from the separating boundary. Errors have positive distances from the separator in Figure 4-9, while correct classifications have negative distances (the choice is arbitrary in this diagram).

Support vector machines use *hinge loss*, so called because the loss graph looks like a hinge. Hinge loss incurs no penalty for an example that is not on the wrong side of the margin. The hinge loss only becomes positive when an example is on the wrong side of the boundary and beyond the margin. Loss then increases linearly with the example's distance from the margin, thereby penalizing points more the farther they are from the separating boundary.

Zero-one loss, as its name implies, assigns a loss of zero for a correct decision and one for an incorrect decision.

For contrast, consider a different sort of loss function. *Squared error* specifies a loss proportional to the square of the distance from the boundary. Squared error loss usually is used for numeric value prediction (regression), rather than classification. The squaring of the error has the effect of greatly penalizing predictions that are grossly wrong. For classification, this would apply large penalties to points far over on the "wrong side" of the separating boundary. Unfortunately, using squared error for classification also penalizes points far on the *correct* side of the decision boundary. For most business problems, choosing squared-error loss for classification or class-probability estimation thus would violate our principle of thinking carefully about whether the loss function

3. There is an immense literature on linear regression for descriptive analysis of data, and we encourage the reader to delve into it. In this book, we treat linear regression simply as one of many modeling techniques. Our treatment does differ from what you are likely to have learned about regression analysis, because we focus on linear regression for making predictions. Other authors have discussed in detail the differences between descriptive modeling and predictive modeling (Shmueli, 2010).

is aligned with the business goal. (Hinge-like versions of squared error have been created because of this misalignment [Rosset & Zhu, 2007].)

We have already discussed most of what we need for linear regression. The linear regression model structure is exactly the same as for the linear discriminant function Equation 4-2:

$$f(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2 + \cdots$$

So, following our general framework for thinking about parametric modeling, we need to decide on the objective function we will use to optimize the model's fit to the data. There are many possibilities. Each different linear regression modeling procedure uses one particular choice (and the data scientist should think carefully about whether it is appropriate for the problem).

The most common ("standard") linear regression procedure makes a powerful and convenient choice. Recall that for regression problems the target variable is numeric. The linear function estimates this numeric target value using Equation 4-2, and of course the *training* data have the actual target value. Therefore, an intuitive notion of the fit of the model is: how far away are the estimated values from the true values on the training data? In other words, how big is the error of the fitted model? Presumably we'd like to minimize this error. For a particular training dataset, we could compute this error for each individual data point and sum up the results. Then the model that fits the data best would be the model with the minimum sum of errors on the training data. And that is exactly what regression procedures do.

You might notice that we really have not actually specified the objective function, because there are many ways to compute the error between an estimated value and an actual value. The method that is most natural is to simply subtract one from the other (and take the absolute value). So if I predict 10 and the actual value is 12 or 8, I make an error of 2. This is called *absolute error*, and we could then minimize the sum of absolute errors or equivalently the mean of the absolute errors across the training data. This makes a lot of sense, but it is not what standard linear regression procedures do.

Standard linear regression procedures instead minimize the sum or mean of the *squares* of these errors—which gives the procedure its common name "least squares" regression. So why do so many people use least squares regression without much thought to alternatives? The short answer is convenience. It is the technique we learn in basic statistics classes (and beyond). It is available to us to use in various software packages. Originally, the least squared error function was introduced by the famous 18th century mathematician Carl Friedrich Gauss, and there are certain theoretical arguments for its use (relating to the normal or "Gaussian" distribution). Often, more importantly, it turns out that squared error is particularly convenient mathematically.[4] This was helpful in the days before computers. From a data science perspective, the convenience extends to theoretical analyses, including a clean decomposition of model error into different sources. More pragmatically, analysts often claim to prefer squared error because it strongly penalizes very large errors. Whether the quadratic penalty is actually appropriate is specific to each application. (Why not take the fourth power of the errors, and penalize large errors even more strongly?)

Importantly, any choice for the objective function has both advantages and drawbacks. For least squares regression a serious drawback is that it is very sensitive to the data: erroneous or otherwise outlying data points can severely skew the resultant linear function. For some business applications, we may not have the resources to spend as much time on manual massaging of the data as we would in other applications. At the extreme, for systems that build and apply models totally automatically, the modeling needs to be much more robust than when doing a detailed regression analysis "by hand." Therefore, for the former application we may want to use a more robust modeling procedure (e.g., use as the objective function absolute error instead of squared error). An important thing to remember is that once we see linear regression simply as an instance of fitting a (linear) model to data, we see that we have to choose the objective function to optimize —and we should do so with the ultimate business application in mind.

Class Probability Estimation and Logistic "Regression"

As mentioned earlier, for many applications we would like to estimate the probability that a new instance belongs to the class of interest. In many cases, we would like to use the estimated probability in a decision-making context that includes other factors such as costs and benefits. For example, predictive modeling from large consumer data is used widely in fraud detection across many industries, especially banking, telecommunications, and online commerce. A linear discriminant could be used to identify accounts or transactions as likely to have been defrauded. The director of the fraud control operation may want the analysts to focus not simply on the cases most likely to be fraud, but on the cases where the most money is at stake—that is, accounts where the company's monetary loss is expected to be the highest. For this we need to estimate the actual probability of fraud. (Chapter 7 will discuss in detail the use of expected value to frame business problems.)

Fortunately, within this same framework for fitting linear models to data, by choosing a different objective function we can produce a model designed to give accurate estimates of class probability. The most common procedure by which we do this is called logistic regression.

4. Gauss agreed with objections to the arbitrariness of this choice.

What exactly is an accurate estimate of class membership probability is a subject of debate beyond the scope of this book. Roughly, we would like (i) the probability estimates to be well calibrated, meaning that if you take 100 cases whose class membership probability is estimated to be 0.2, then about 20 of them will actually belong to the class. We would also like (ii) the probability estimates to be discriminative, in that if possible they give meaningfully different probability estimates to different examples. The latter condition keeps us from simply giving the "base rate" (the overall prevalence in the population) as the prediction for every example. Say 0.5% of accounts overall are fraudulent. Without condition (ii) we could simply predict the same 0.5% probability for each account; those estimates would be well calibrated—but not discriminative at all.

To understand logistic regression, it is instructive to first consider: exactly what is the problem with simply using our basic linear model (Equation 4-2) to estimate the class probability? As we discussed, an instance being further from the separating boundary intuitively ought to lead to a higher probability of being in one class or the other, and the output of the linear function, $f(\mathbf{x})$, gives the distance from the separating boundary. However, this also shows the problem: $f(\boldsymbol{x})$ ranges from $-\infty$ to ∞, and a probability should range from zero to one.

So let's take a brief stroll down a garden path and ask how else we might cast our distance from the separator, $f(\boldsymbol{x})$, in terms of the likelihood of class membership. Is there another representation of the likelihood of an event that we use in everyday life? If we could come up with one that ranges from $-\infty$ to ∞, then we might model this other notion of likelihood with our linear equation.

One very useful notion of the likelihood of an event is the odds. The odds of an event is the ratio of the probability of the event occurring to the probability of the event not occurring. So, for example, if the event has an 80% probability of occurrence, the odds are 80:20 or 4:1. And if the linear function were to give us the odds, a little algebra would tell us the probability of occurrence. Let's look at a more detailed example. Table 4-1 shows the odds corresponding to various probabilities.

Table 4-1. Probabilities and the corresponding odds.

Probability	Corresponding odds
0.5	50:50 or 1
0.9	90:10 or 9
0.999	999:1 or 999
0.01	1:99 or 0.0101
0.001	1:999 or 0.001001

Looking at the range of the odds in Table 4-1, we can see that it still is not quite right as an interpretation of the distance from the separating boundary. Again, the distance from the boundary is between –∞ and ∞, but as we can see from the example, the odds range from 0 to ∞. Nonetheless, we can solve our garden-path problem simply by taking the logarithm of the odds (called the "log-odds"), since for any number in the range 0 to ∞ its log will be between –∞ to ∞. These are shown in Table 4-2.

Table 4-2. Probabilities, odds, and the corresponding log-odds.

Probability	Odds	Log-odds
0.5	50:50 or 1	0
0.9	90:10 or 9	2.19
0.999	999:1 or 999	6.9
0.01	1:99 or 0.0101	–4.6
0.001	1:999 or 0.001001	–6.9

So if we only cared about modeling *some* notion of likelihood, rather than the class membership probability specifically, we could model the *log-odds* with $f(\mathbf{x})$.

Lo and behold, our garden path has taken us directly back to our main topic. This is exactly a logistic regression model: the same linear function $f(\mathbf{x})$ that we've examined throughout the chapter is used as a measure of the log-odds of the "event" of interest. More specifically, $f(\mathbf{x})$ is the model's estimation of the log-odds that **x** belongs to the positive class. For example, the model might estimate the log-odds that a customer described by feature vector **x** will leave the company when her contract expires. Moreover, with a little algebra we can translate these log-odds into the probability of class membership. This is a little more technical than most of the book, so we've relegated it to a special "technical details" subsection (next), which also discusses what exactly is the objective function that is optimized to fit a logistic regression to the data. You can read that section in detail or just skim it. The most important points are:

- For probability estimation, logistic regression uses the same linear model as do our linear discriminants for classification and linear regression for estimating numeric target values.
- The output of the logistic regression model is interpreted as the log-odds of class membership.
- These log-odds can be translated directly into the probability of class membership. Therefore, logistic regression often is thought of simply as a model for the probability of class membership. You have undoubtedly dealt with logistic regression models many times without even knowing it. They are used widely to estimate quantities like the probability of default on credit, the probability of response to an

offer, the probability of fraud on an account, the probability that a document is relevant to a topic, and so on.

After the technical details section, we will compare the linear models we've developed in this chapter with the tree-structured models we developed in Chapter 3.

Note: Logistic regression is a misnomer

Above we mentioned that the name logistic *regression* is a misnomer under the modern use of data science terminology. Recall that the distinction between classification and regression is whether the value for the target variable is categorical or numeric. For logistic regression, the model produces a numeric estimate (the estimation of the log-odds). However, the values of the target variable in the data are categorical. Debating this point is rather academic. What is important to understand is what logistic regression is doing. It is estimating the log-odds or, more loosely, the probability of class membership (a numeric quantity) over a categorical class. So we consider it to be a class probability estimation model and *not* a regression model, despite its name.

* Logistic Regression: Some Technical Details

Since logistic regression is used so widely, and is not as intuitive as linear regression, let's examine a few of the technical details. You may skip this subsection without it affecting your understanding of the rest of the book.

So, technically, what is the bottom line for the logistic regression model? Let's use $p_+(\mathbf{x})$ to represent the model's estimate of the probability of class membership of a data item represented by feature vector $\mathbf{x}$.[5] Recall that the class + is whatever is the (binary) event that we are modeling: responding to an offer, leaving the company after contract expiration, being defrauded, etc. The estimated probability of the event not occurring is therefore $1 - p_+(\mathbf{x})$.

5. Often technical treatments use the "hat" notation, $\hat{p}$, to differentiate the model's *estimate* of the probability of class membership from the actual probability of class membership. We will not use the hat, but the technically savvy reader should keep that in mind.

Equation 4-3. Log-odds linear function

$$\log\left(\frac{p_+(\mathbf{x})}{1 - p_+(\mathbf{x})}\right) = f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots$$

Thus, Equation 4-3 specifies that for a particular data item, described by feature-vector **x**, the log-odds of the class is equal to our linear function, *f(**x**)*. Since often we actually want the estimated probability of class membership, not the log-odds, we can solve for $p_+(\mathbf{x})$ in Equation 4-3. This yields the not-so-pretty quantity in Equation 4-4.

Equation 4-4. The logistic function

$$p_+(\mathbf{x}) = \frac{1}{1 + e^{-f(\mathbf{x})}}$$

Although the quantity in Equation 4-4 is not very pretty, by plotting it in a particular way we can see that it matches exactly our intuitive notion that we would like there to be relative certainty in the estimations of class membership far from the decision boundary, and uncertainty near the decision boundary.

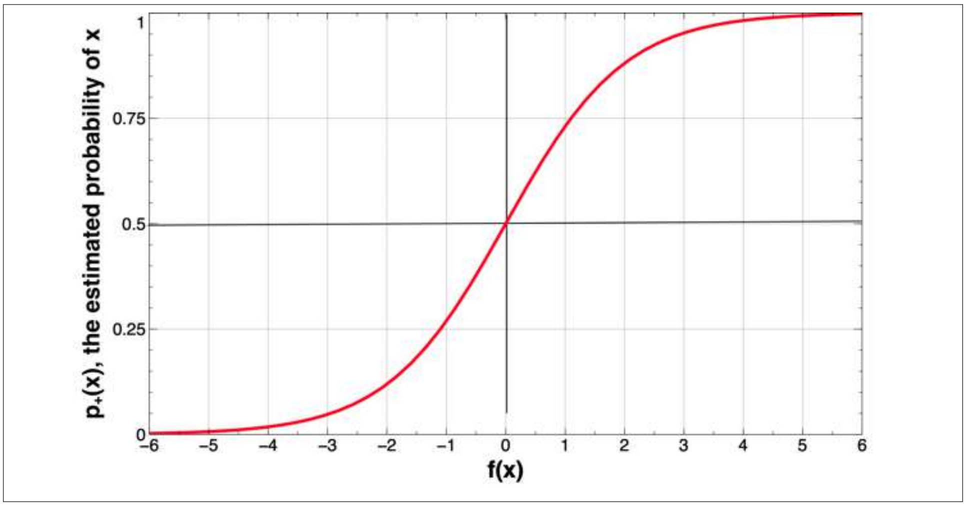


*Figure 4-10. Logistic regression's estimate of class probability as a function of f(**x**), (i.e., the distance from the separating boundary). This curve is called a "sigmoid" curve because of its "S" shape, which squeezes the probabilities into their correct range (between zero and one).*

Figure 4-10 plots the estimated probability $p_+(\mathbf{x})$ (vertical axis) as a function of the distance from the decision boundary (horizontal axis). The figure shows that at the decision boundary (at distance $x = 0$), the probability is 0.5 (a coin toss). The probability varies approximately linearly near to the decision boundary, but then approaches certainty farther away. Part of the "fitting" of the model to the data includes determining the slope of the almost-linear part, and thereby how quickly we are certain of the class as we move away from the boundary.

The other main technical point that we omitted in our main discussion above is: what then is the objective function we use to fit the logistic regression model to the data? Recall that the training data have binary values of the target variable. The model can be applied to the training data to produce estimates that each of the training data points belongs to the target class. What would we want? Ideally, any positive example $\mathbf{x}_+$ would have $p_+(\mathbf{x}_+) = 1$ and any negative example $\mathbf{x}_\bullet$ would have $p_+(\mathbf{x}_\bullet) = 0$. Unfortunately, with real-world data it is unlikely that we will be able to estimate these probabilities perfectly (consider the task of estimating that a consumer described by demographic variables would respond to a particular offer). Nevertheless, we still would like $p_+(\mathbf{x}_+)$ to be as close as possible to one and $p_+(\mathbf{x}_\bullet)$ to be as close as possible to zero.

This leads us to the standard objective function for fitting a logistic regression model to data. Consider the following function computing the "likelihood" that a particular labeled example belongs to the correct class, given a set of parameters $\mathbf{w}$ that produces class probability estimates $p_+(\mathbf{x})$:

$$g(\mathbf{x}, \mathbf{w}) = \begin{cases} p_+(\mathbf{x}) & \text{if } \mathbf{x} \text{ is a } + \\ 1 - p_+(\mathbf{x}) & \text{if } \mathbf{x} \text{ is a } \bullet \end{cases}$$

The g function gives the model's estimated probability of seeing $\mathbf{x}$'s actual class given $\mathbf{x}$'s features. Now consider summing the g values across all the examples in a labeled dataset. And do that for different parameterized models—in our case, different sets of weights ($\mathbf{w}$) for the logistic regression. The model (set of weights) that gives the highest sum is the model that gives the highest "likelihood" to the data—the "maximum likelihood" model. The maximum likelihood model "on average" gives the highest probabilities to the positive examples and the lowest probabilities to the negative examples.

Class Labels and Probabilities

One may be tempted to think that the target variable *is* a representation of the probability of class membership, and the observed values of the target variable in the training data simply report probabilities of $p(x) = 1$ for cases that are observed to be in the class and $p(x) = 0$ for instances that are observed not to be in the class. However, this is not generally consistent with how logistic regression models are used. Take an application to targeted marketing for example. For a consumer c, our model may estimate the probability of responding to the offer to be $p(c \text{ responds}) = 0.02$. In the data, we see that the person indeed does respond. That does not mean that this consumer's probability of responding actually was 1.0, nor that the model incurred a large error on this example. The consumer's probability may indeed have been around $p(c \text{ responds}) = 0.02$, which actually is a high probability of response for many campaigns, and the consumer just happened to respond this time.

A more satisfying way to think about it is that the training data comprise a set of statistical "draws" from the underlying probabilities, rather than representing the underlying probabilities themselves. The logistic regression procedure then tries to estimate the probabilities (the probability distribution over the instance space) with a linear-log-odds model, based on the observed data on the result of the draws from the distribution.

Example: Logistic Regression versus Tree Induction

Though classification trees and linear classifiers both use linear decision boundaries, there are two important differences between them:

1. A classification tree uses decision boundaries that are *perpendicular* to the instance-space axes (see Figure 4-1), whereas the linear classifier can use decision boundaries of any direction or orientation (see Figure 4-3). This is a direct consequence of the fact that classification trees select a single attribute at a time whereas linear classifiers use a weighted combination of all attributes.
2. A classification tree is a "piecewise" classifier that segments the instance space recursively when it has to, using a divide-and-conquer approach. In principle, a classification tree can cut up the instance space arbitrarily finely into very small regions (though we will see reasons to avoid that in Chapter 5). A linear classifier places a *single* decision surface through the entire space. It has great freedom in the orientation of the surface, but it is limited to a single division into two segments. This is a direct consequence of there being a single (linear) equation that uses all of the variables, and must fit the entire data space.

It is usually not easy to determine in advance which of these characteristics are a better match to a given dataset. You likely will not know what the best decision boundary will look like. So practically speaking, what are the consequences of these differences?

When applied to a business problem, there is a difference in the comprehensibility of the models to stakeholders with different backgrounds. For example, what exactly a logistic regression model is doing can be quite understandable to people with a strong background in statistics, and very difficult to understand for those who do not. A decision tree, if it is not too large, may be considerably more understandable to someone without a strong statistics or mathematics background.

Why is this important? For many business problems, the data science team does not have the ultimate say in which models are used or implemented. Often there is at least one manager who must "sign off" on the use of a model in practice, and in many cases a set of stakeholders need to be satisfied with the model. For example, to put in place a new model to dispatch technicians to repair problems after customer calls to the telephone company, managers from operations support, customer service, and technical development all need to be satisfied that the new model will do more good than harm —since for this problem no model is perfect.

Let's try logistic regression on a simple but realistic dataset, the Wisconsin Breast Cancer Dataset. As with the Mushroom dataset from the previous chapter, this is another popular dataset from the the machine learning dataset repository at the University of California at Irvine.

Each example describes characteristics of a cell nuclei image, which has been labeled as either *benign* or *malignant* (cancerous), based on an expert's diagnosis of the cells. A sample cell image is shown in Figure 4-11.

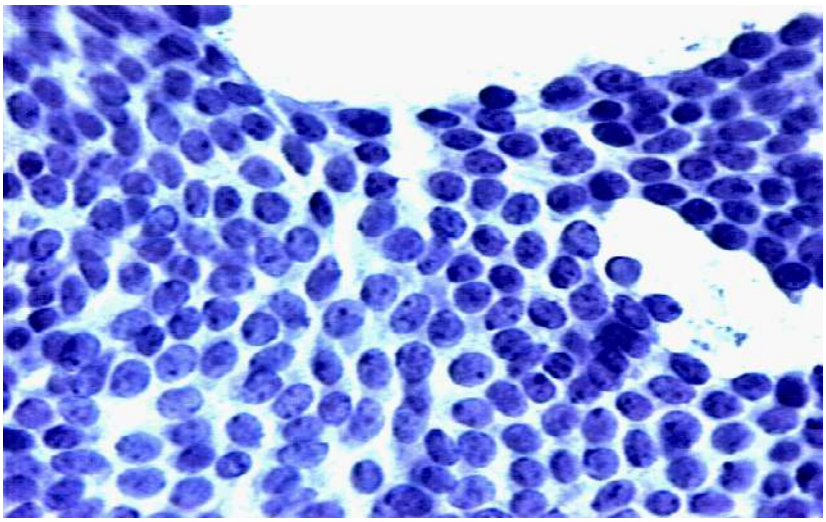


Figure 4-11. One of the cell images from which the Wisconsin Breast Cancer dataset was derived. (Image courtesy of Nick Street and Bill Wolberg.)

From each image 10 fundamental characteristics were extracted, listed in Table 4-3.

Table 4-3. The attributes of the Wisconsin Breast Cancer dataset.

Attribute name	Description
RADIUS	*Mean of distances from center to points on the perimeter*
TEXTURE	*Standard deviation of grayscale values*
PERIMETER	*Perimeter of the mass*
AREA	*Area of the mass*
SMOOTHNESS	*Local variation in radius lengths*
COMPACTNESS	*Computed as: perimeter^2/area – 1.0*
CONCAVITY	*Severity of concave portions of the contour*
CONCAVE POINTS	*Number of concave portions of the contour*
SYMMETRY	*A measure of the nucleii's symmetry*
FRACTAL DIMENSION	*'Coastline approximation'* – 1.0
DIAGNOSIS (Target)	*Diagnosis of cell sample: malignant or benign*

These were "computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image." From each of these basic characteristics, three values were computed: the mean (`_mean`), standard error (`_SE`), and "worst" or largest (mean of the three largest values, `_worst`). This resulted in 30 measured attributes in the dataset. There are 357 benign images and 212 malignant images.

Table 4-4. Linear equation learned by logistic regression on the Wisconsin Breast Cancer dataset (see text and Table 4-3 for a description of the attributes).

Attribute	Weight (learned parameter)
SMOOTHNESS_worst	22.3
CONCAVE_mean	19.47
CONCAVE_worst	11.68
SYMMETRY_worst	4.99
CONCAVITY_worst	2.86
CONCAVITY_mean	2.34
RADIUS_worst	0.25
TEXTURE_worst	0.13
AREA_SE	0.06
TEXTURE_mean	0.03
TEXTURE_SE	–0.29
COMPACTNESS_mean	–7.1
COMPACTNESS_SE	–27.87
w_0 *(intercept)*	–17.7

Table 4-4 shows the linear model learned by logistic regression to predict benign versus malignant for this dataset. Specifically, it shows the nonzero weights ordered from highest to lowest.

The performance of this model is quite good—it makes only six mistakes on the entire dataset, yielding an accuracy of about 98.9% (the percentage of the instances that the model classifies correctly). For comparison, a classification tree was learned from the same dataset (using Weka's J48 implementation). The resulting tree is shown in Figure 4-13. The tree has 25 nodes altogether, with 13 leaf nodes. Recall that this means that the tree model partitions the instances into 13 segments. The classification tree's accuracy is 99.1%, slightly higher than that of logistic regression.

The intent of this experiment is only to illustrate the results of two different methods on a dataset, but it is worth digressing briefly to think about these performance results. First, an accuracy figure like 98.9% sounds like a very good result. Is it? We see many such accuracy numbers thrown around in the data mining literature, but evaluating classifiers on real-world problems like cancer diagnosis is often difficult and complex. We discuss evaluation in detail in Chapter 7 and Chapter 8.

Second, consider the two performance results here: 98.9% versus 99.1%. Since the classification tree gives slightly higher accuracy, we might be tempted to conclude that it's the better model. Should we believe this? This difference is caused by only a *single* additional error out of the 569 examples. Furthermore, the accuracy numbers were derived by evaluating each model on the same set of examples it was built from. How confident should we be in this evaluation? Chapter 5, Chapter 7, and Chapter 8 discuss guidelines and pitfalls of model evaluation.

Nonlinear Functions, Support Vector Machines, and Neural Networks

So far this chapter has focused on the numeric functions most commonly used in data science: linear models. This set of models includes a wide variety of different techniques. In addition, in Figure 4-12 we show that such linear functions can actually represent nonlinear models, *if we include more complex features in the functions*. In this example, we used the Iris dataset from "An Example of Mining a Linear Discriminant from Data" on page 88 and added a squared term to the input data: **Sepal width^2**. The resulting model is a curved line (a parabola) in the original feature space. **Sepal width^2**. We also added a single data point to the original dataset, an Iris Versicolor example added at (4,0.7), shown starred.

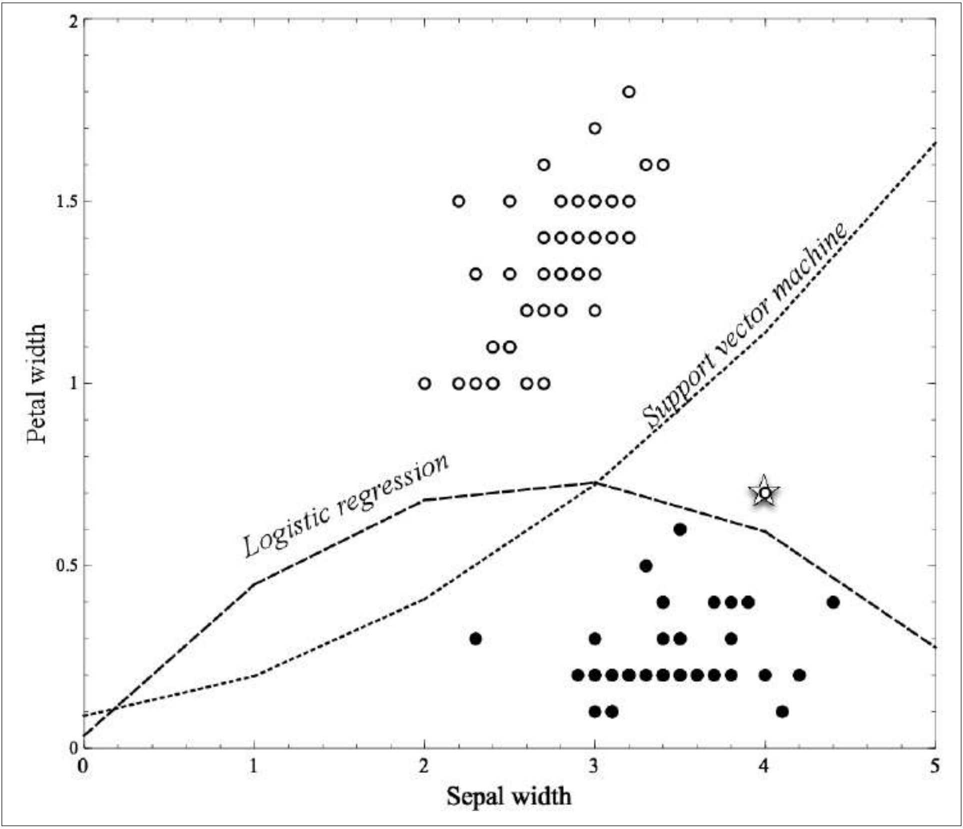


Figure 4-12. The Iris dataset with a nonlinear feature. In this figure, logistic regression and support vector machine—both linear models—are provided an additional feature, ***Sepal width^2****, which allows both the freedom to create more complex, nonlinear models (boundaries), as shown.*

Our fundamental concept is much more general than just the application of fitting linear functions. Of course, we could specify arbitrarily complex numeric functions and fit their parameters to the data. The two most common families of techniques that are based on fitting the parameters of complex, nonlinear functions are *nonlinear support-vector machines* and *neural networks*.

One can think of nonlinear support vector machines as essentially a systematic way of implementing the "trick" we just discussed of adding more complex terms and fitting a linear function to them. Support vector machines have a so-called "kernel function" that maps the original features to some other feature space. Then a linear model is fit to this new feature space, just as in our simple example in Figure 4-12. Generalizing this, one could implement a nonlinear support vector machine with a "polynomial kernel," which essentially means it would consider "higher-order" combinations of the original features (e.g., squared features, products of features). A data scientist would become familiar with the different alternatives for kernel functions (linear, polynomial, and others).

Neural networks also implement complex nonlinear numeric functions, based on the fundamental concepts of this chapter. Neural networks offer an intriguing twist. One can think of a neural network as a "stack" of models. On the bottom of the stack are the original features. From these features are learned a variety of relatively simple models. Let's say these are logistic regressions. Then, each subsequent layer in the stack applies a simple model (let's say, another logistic regression) to the outputs of the next layer down. So in a two-layer stack, we would learn a set of logistic regressions from the original features, and then learn a logistic regression using as features the outputs of the first set of logistic regressions. We could think of this very roughly as first creating a set of "experts" in different facets of the problem (the first-layer models), and then learning how to weight the opinions of these different experts (the second-layer model).[6]

The idea of neural networks gets even more intriguing. We might ask: if we are learning those lower-layer logistic regressions—the different experts—what would be the *target* variable for each? While some practitioners build stacked models where the lower-layer experts are built to represent specific things using specific target variables (e.g., Perlich et al., 2013), more generally with neural networks target labels for training are provided only for the final layer (the actual target variable). So how are the lower-layer logistic regressions trained? We can understand by returning to the fundamental concept of this chapter. The stack of models can be represented by one big parameterized numeric function. The paramenters now are the coefficients of all the models, taken together. So once we have decided on an objective function representing what we want to optimize (e.g., the fit to the training data, based on some fitting function), we can then apply an optimization procedure to find the best parameters to this very complex numeric function. When we're done, we have the parameters to all the models, and thereby have learned the "best" set of lower-level experts and also the best way to combine them, all simultaneously.

Note: Neural networks are useful for many tasks

This section describes neural networks for classification and regression. The field of neural networks is broad and deep, with a long history. Neural networks have found wide application throughout data mining. They are commonly used for many other tasks mentioned in Chapter 2, such as clustering, time series analysis, profiling, and so on.

6. Compare this with the notion of *ensemble methods* described in Chapter 12.

So, given how cool that sounds, why wouldn't we want to do that all the time? The tradeoff is that as we increase the amount of flexibility we have to fit the data, we increase the chance that we fit the data *too* well. The model can fit details of its particular training set rather than finding patterns or models that apply more generally. Specifically, we really want models that apply to other data drawn from the same population or application. This concern is not specific to neural networks, but is very general. It is one of the most important concepts in data science—and it is the subject of the next chapter.

Summary

This chapter introduced a second type of predictive modeling technique called function fitting or parametric modeling. In this case the model is a partially specified equation: a numeric function of the data attributes, with some unspecified numeric parameters. The task of the data mining procedure is to "fit" the model to the data by finding the best set of parameters, in some sense of "best."

There are many varieties of function fitting techniques, but most use the same linear model structure: a simple weighted sum of the attribute values. The parameters to be fit by the data mining are the weights on the attributes. Linear modeling techniques include linear discriminants such as support-vector machines, logistic regression, and traditional linear regression. Conceptually the key difference between these techniques is their answer to a key issue, *What exactly do we mean by best fitting the data?* The goodness of fit is described by an "objective function," and each technique uses a different function. The resulting techniques may be quite different.

We now have seen two very different sorts of data modeling, tree induction and function fitting, and have compared them (in "Example: Logistic Regression versus Tree Induction" on page 102). We have also introduced two criteria by which models can be evaluated: the predictive performance of a model and its intelligibility. It is often advantageous to build different sorts of models from a dataset to gain insight.

This chapter focused on the fundamental concept of optimizing a model's fit to data. However, doing this leads to the most important fundamental *problem* with data mining —if you look hard enough, you will find structure in a dataset, even if it's just there by chance. This tendency is known as *overfitting*. Recognizing and avoiding overfitting is an important general topic in data science; and we devote the entire next chapter to it.

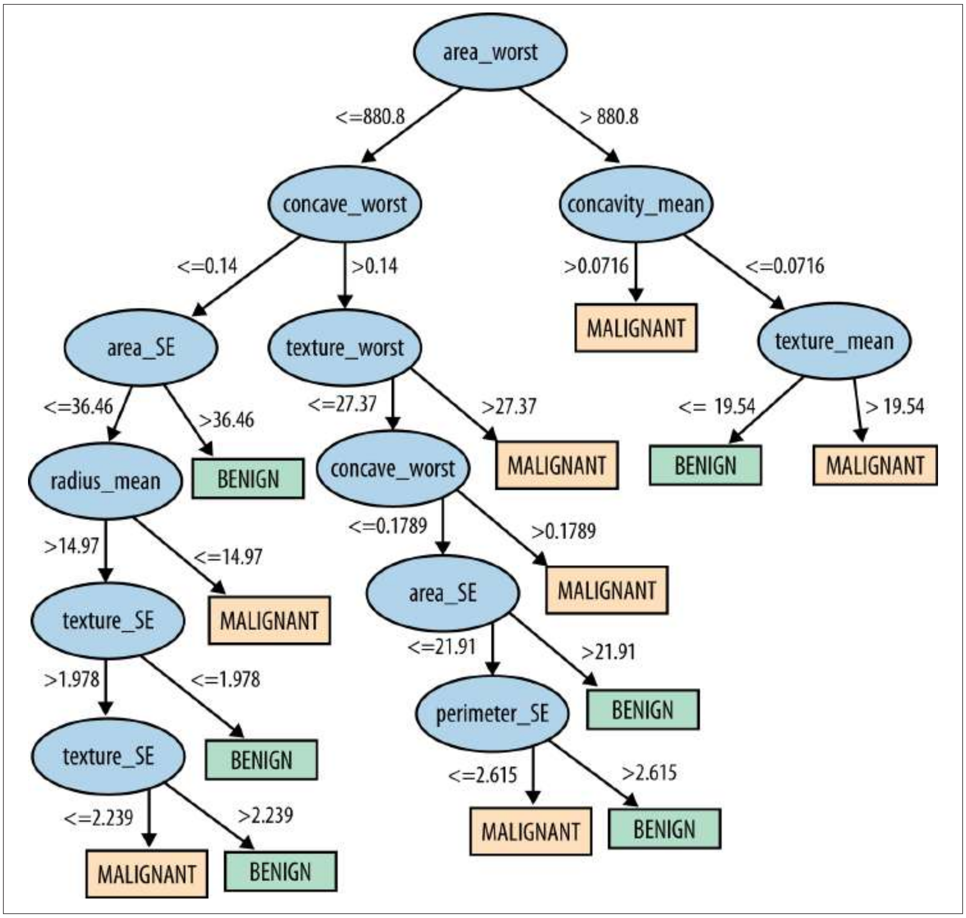


Figure 4-13. Decision tree learned from the Wisconsin Breast Cancer dataset.

CHAPTER 5

Overfitting and Its Avoidance

Fundamental concepts: *Generalization; Fitting and overfitting; Complexity control.*

Exemplary techniques: *Cross-validation; Attribute selection; Tree pruning; Regularization.*

One of the most important fundamental notions of data science is that of overfitting and generalization. If we allow ourselves enough flexibility in searching for patterns in a particular dataset, we will find patterns. Unfortunately, these "patterns" may be just chance occurrences in the data. As discussed previously, we are interested in patterns that generalize—that predict well for instances that we have not yet observed. Finding chance occurrences in data that look like interesting patterns, but which do not generalize, is called *overfitting* the data.

Generalization

Consider the following (extreme) example. You're a manager at MegaTelCo, responsible for reducing customer churn. I run a data mining consulting group. You give my data science team a set of historical data on customers who have stayed with the company and customers who have departed within six months of contract expiration. My job is to build a model to distinguish customers who are likely to churn based on some features, as we've discussed previously. I mine the data and build a model. I give you back the code for the model, to implement in your company's churn-reduction system.

Of course you are interested in whether my model is any good, so you ask your technical team to check the performance of the model on the historical data. You understand that historical performance is no guarantee of future success, but your experience tells you that churn patterns remain relatively stable, except for major changes to the industry (such as the introduction of the iPhone), and you know of no such major changes since these data were collected. So, the tech team runs the historical dataset through the model. Your technical lead reports back that this data science team is amazing. The model is

100% accurate. It does not make a single mistake, identifying correctly all the churners as well as the nonchurners.

You're experienced enough not to be comfortable with that answer. You've had experts looking at churn behavior for a long time, and if there really were 100% accurate indicators, you figure you would be doing better than you currently are. Maybe this is just a lucky fluke?

It was not a lucky fluke. Our data science team can do that every time. Here is how we built the model. We stored the feature vector for each customer who has churned in a database table. Let's call that T_c. Then, in use, when the model is presented with a customer to determine the likelihood of churning, it takes the customer's feature vector, looks her up in T_c, and reports "100% likelihood of churning" if she is in T_c and "0% likelihood of churning" if she is not in T_c. So, when the tech team applies our model to the historical dataset, the model predicts perfectly.[1]

Call this simple approach a *table model*. It memorizes the training data and performs no generalization. What is the problem with this? Consider how we'll use the model in practice. When a *previously unseen* customer's contract is about to expire, we'll want to apply the model. Of course, this customer was not part of the historical dataset, so the lookup will fail since there will be no exact match, and the model will predict "0% likelihood of churning" for this customer. In fact, the model will predict this for every customer (not in the training data). A model that looked perfect would be completely useless in practice!

This may seem like an absurd scenario. In reality, no one would throw raw customer data into a table and claim it was a "predictive model" of anything. But it is important to think about why this is a bad idea, because it fails for the same reason other, more realistic data mining efforts may fail. It is an extreme example of two related fundamental concepts of data science: *generalization* and *overfitting*. Generalization is the property of a model or modeling process, whereby the model applies to data that were not used to build the model. In this example, the model does not generalize at all beyond the data that were used to build it. It is tailored, or "fit," perfectly to the training data. In fact, it is "overfit."

This is the important point. Every dataset is a finite sample of a population—in this case, the population of phone customers. We want models to apply not just to the exact training set but to the general population from which the training data came. We may worry that the training data were not representative of the true population, but that is not the problem here. The data were representative, but the data mining did not create a model that generalized beyond the training data.

1. Technically, this is not necessarily true: there may be two customers with the same feature vector description, one of whom churns and the other does not. We can ignore that possibility for the sake of this example. For example, we can assume that the unique customer ID is one of the features.

Overfitting

Overfitting is the tendency of data mining procedures to tailor models to the training data, at the expense of generalization to previously unseen data points. The example from the previous section was contrived; the data mining built a model using pure memorization, the most extreme overfitting procedure possible. However, all data mining procedures have the tendency to overfit to some extent—some more than others. The idea is that if we look hard enough we will find patterns in a dataset. As the Nobel Laureate Ronald Coase said, "If you torture the data long enough, it will confess."

Unfortunately, the problem is insidious. The answer is not to use a data mining procedure that doesn't overfit because all of them do. Nor is the answer to simply use models that produce less overfitting, because there is a fundamental trade-off between model complexity and the possibility of overfitting. Sometimes we may simply want more complex models, because they will better capture the real complexities of the application and thereby be more accurate. There is no single choice or procedure that will eliminate overfitting. The best strategy is to recognize overfitting and to manage complexity in a principled way.

The rest of this chapter discusses overfitting in more detail, methods for assessing the degree of overfitting at modeling time, as well as methods for avoiding overfitting as much as possible.

Overfitting Examined

Before discussing what to do about overfitting, we need to know how to recognize it.

Holdout Data and Fitting Graphs

Let's now introduce a simple analytic tool: the *fitting graph*. A fitting graph shows the accuracy of a model as a function of complexity. To examine *over*fitting, we need to introduce a concept that is fundamental to evaluation in data science: *holdout* data.

The problem in the prior section was that the model was evaluated on the training data —exactly the same data that were used to build it. Evaluation on training data provides no assessment of how well the model generalizes to unseen cases. What we need to do is to "hold out" some data for which we know the value of the target variable, but which will not be used to build the model. These are not the actual *use* data, for which we ultimately would like to predict the value of the target variable. Instead, creating holdout data is like creating a "lab test" of generalization performance. We will simulate the use scenario on these holdout data: we will hide from the model (and possibly the modelers) the actual values for the target on the holdout data. The model will predict the values. Then we estimate the *generalization performance* by comparing the predicted values with the hidden true values. There is likely to be a difference between the model's ac-

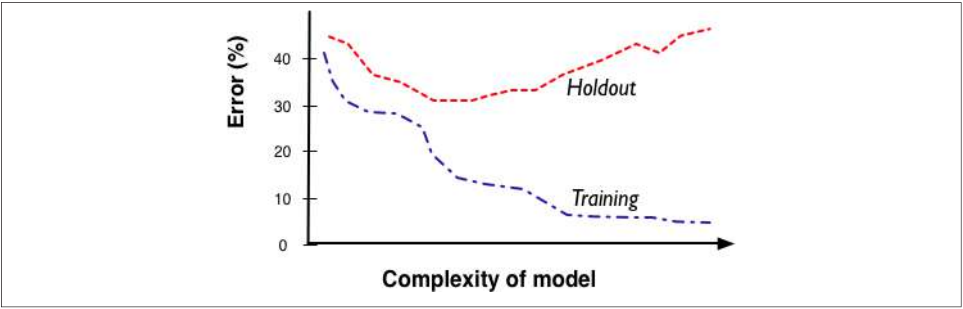


Figure 5-1. A typical fitting graph. Each point on a curve represents an accuracy estimation of a model with a specified complexity (as indicated on the horizontal axis). Accuracy estimates on training data and testing data vary differently based on how complex we allow a model to be. When the model is not allowed to be complex enough, it is not very accurate. As the models get too complex, they look very accurate on the training data, but in fact are overfitting—the training accuracy diverges from the holdout (generalization) accuracy.

curacy on the training set (sometimes called the "in-sample" accuracy) and the model's generalization accuracy, as estimated on the holdout data. Thus, when the holdout data are used in this manner, they often are called the "test set."

The accuracy of a model depends on how complex we allow it to be. A model can be complex in different ways, as we will discuss in this chapter. First let us use this distinction between training data and holdout data to define the fitting graph more precisely. The fitting graph (see Figure 5-1) shows the difference between a modeling procedure's accuracy on the training data and the accuracy on holdout data as model complexity changes. Generally, there will be more overfitting as one allows the model to be more complex. (Technically, the chance of overfitting increases as one allows the modeling procedure more flexibility in the models it can produce; we will ignore that distinction in this book).

Figure 5-2 shows a fitting graph for the customer churn "table model" described earlier. Since this was an extreme example the fitting graph will be peculiar. Again, the *x* axis measures the complexity of the model; in this case, the number of rows allowed in the table. The *y* axis measures the error. As we allow the table to increase in size, we can memorize more and more of the training set, and with each new row the training set error decreases. Eventually the table is large enough to contain the entire training set (marked N on the *x* axis) and the error goes to zero and remains there. However, the testing (holdout) set error starts at some value (let's call it *b*) and never decreases, because there is never an overlap between the training and holdout sets. The large gap between the two is a strong indication of memorization.

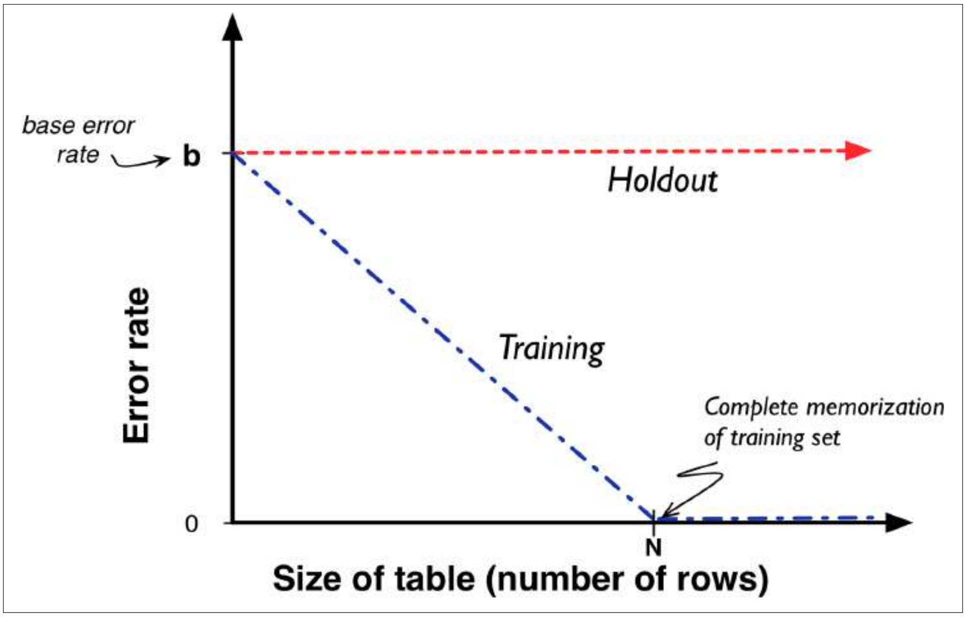


Figure 5-2. A fitting graph for the customer churn (table) model.

Note: Base rate

What would *b* be? Since the table model always predicts `no churn` for every new case with which it is presented, it will get every `no churn` case right and every `churn` case wrong. Thus the error rate will be the percentage of `churn` cases in the population. This is known as the *base rate*, and a classifier that always selects the majority class is called a base rate classifier.

A corresponding baseline for a regression model is a simple model that always predicts the mean or median value of the target variable.

You will occasionally hear reference to "base rate performance," and this is what it refers to. We will revisit the base rate again in the next chapter.

We've discussed in the previous chapters two very different sorts of modeling procedures: recursive partitioning of the data as done for tree induction, and fitting a numeric model by finding an optimal set of parameters, for example the weights in a linear model. We can now examine overfitting for each of these procedures.

Overfitting in Tree Induction

Recall how we built tree-structured models for classification. We applied a fundamental ability to find important, predictive individual attributes repeatedly (recursively) to smaller and smaller data subsets. Let's assume for illustration that the dataset does not have two instances with exactly the same feature vector but different target values. If we continue to split the data, eventually the subsets will be pure—all instances in any chosen subset will have the same value for the target variable. These will be the leaves of our tree. There might be multiple instances at a leaf, all with the same value for the target variable. If we have to, we can keep splitting on attributes, and subdividing our data until we're left with a single instance at each leaf node, which is pure by definition.

What have we just done? We've essentially built a version of the lookup table discussed in the prior section as an extreme example, of overfitting! Any training instance given to the tree for classification will make its way down, eventually landing at the appropriate leaf—the leaf corresponding to the subset of the data that includes this particular training instance. What will be the accuracy of this tree on the training set? It will be perfectly accurate, predicting correctly the class for every training instance.

Will it generalize? Possibly. This tree should be slightly better than the lookup table because every previously unseen instance will arrive at *some* classification, rather than just failing to match; the tree will give a nontrivial classification even for instances it has not seen before. Therefore, it is useful to examine empirically how well the accuracy on the training data tends to correspond to the accuracy on test data.

A procedure that grows trees until the leaves are pure tends to overfit. Tree-structured models are very flexible in what they can represent. Indeed, they can represent any function of the features, and if allowed to grow without bound they can fit it to arbitrary precision. But the trees may need to be huge in order to do so. The complexity of the tree lies in the number of nodes.

Figure 5-3 shows a typical fitting graph for tree induction. Here we artificially limit the maximum size of each tree, as measured by the number of nodes it's allowed to have, indicated on the *x* axis (which is log scale for convenience). For each tree size we create a new tree from scratch, using the training data. We measure two values: its accuracy on the training set and its accuracy on the holdout (test) set. If the data subsets at the leaves are not pure, we will predict the target variable based on some average over the target values in the subset, as we discussed in Chapter 3.

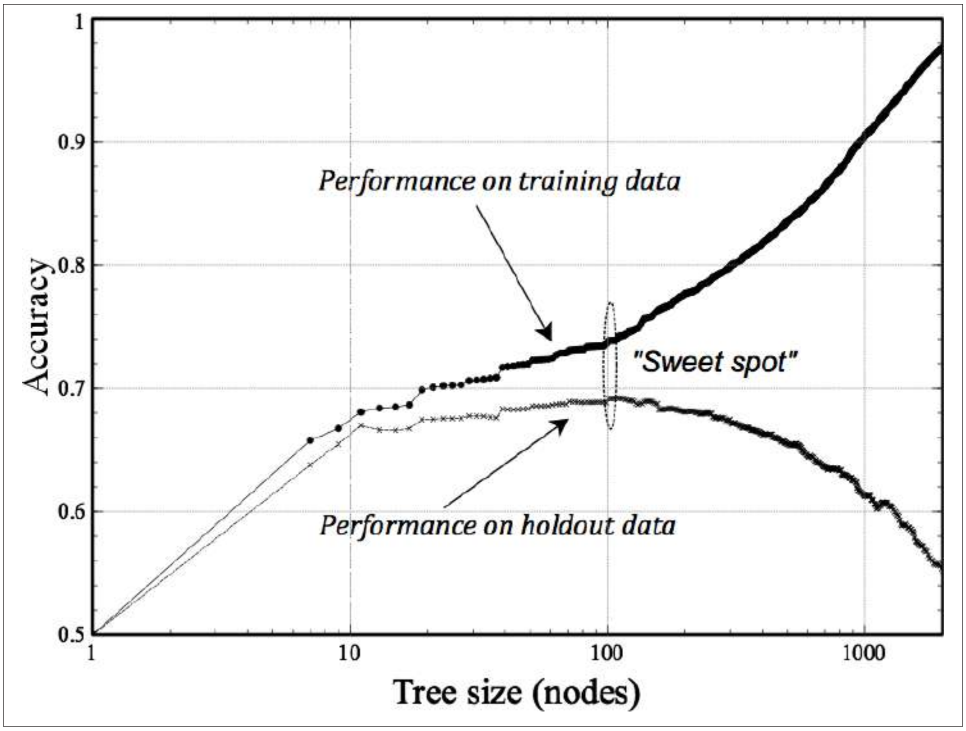


Figure 5-3. A typical fitting graph for tree induction.

Beginning at the left, the tree is very small and has poor performance. As it is allowed more and more nodes it improves rapidly, and both training-set accuracy and holdout-set accuracy improve. Also we see that training-set accuracy always is at least a little better than holdout-set accuracy, since we did get to look at the training data when building the model. But at some point the tree starts to overfit: it acquires details of the training set that are not characteristic of the population in general, as represented by the holdout set. In this example overfitting starts to happen at around $x = 100$ nodes, denoted the "sweet spot" in the graph. As the trees are allowed to get larger, the training-set accuracy continues to increase—in fact, it is capable of memorizing the entire training set if we let it, leading to an accuracy of 1.0 (not shown). But the holdout accuracy declines as the tree grows past its "sweet spot"; the data subsets at the leaves get smaller and smaller, and the model generalizes from fewer and fewer data. Such inferences will be increasingly error-prone and the performance on the holdout data suffers.

In summary, from this fitting graph we may infer that overfitting on this dataset starts to dominate at around 100 nodes, so we should restrict tree size to this value.[2] This represents the best trade-off between the extremes of (i) not splitting the data at all and simply using the average target value in the entire dataset, and (ii) building a complete tree out until the leaves are pure.

Unfortunately, no one has come up with a procedure to determine this exact sweet spot theoretically, so we have to rely on empirically based techniques. Before discussing those, let's examine overfitting in our second sort of modeling procedure.

Overfitting in Mathematical Functions

There are different ways to allow more or less complexity in mathematical functions. There are entire books on the topic. This section discusses one very important way, and "* Avoiding Overfitting for Parameter Optimization" on page 136 discuss a second one. We urge you to at least skim that advanced (starred) section because it introduces concepts and vocabulary in common use by data scientists these days, that can make a non-data scientist's head swim. Here we will summarize and give you enough to understand such discussions at a conceptual level.[3] But first, let's discuss a much more straightforward way in which functions can become too complex.

One way mathematical functions can become more complex is by adding more variables (more attributes). For example, say that we have a linear model as described in Equation 4-2:

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

As we add more x_i's, the function becomes more and more complicated. Each x_i has a corresponding w_i, which is a learned parameter of the model.

Modelers sometimes even change the function from being truly linear in the original attributes by adding new attributes that are nonlinear versions of original attributes. For example, I might add a fourth attribute $x_4 = x^2_1$. Also, we might expect that the ratio of x_2 and x_3 is important, so we add a new attribute $x_5 = x_2/x_3$. Now we're trying to find the parameters (weights) of:

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$$

2. Note that 100 nodes is not some special universal value. It is specific to this particular dataset. If we changed the data significantly, or even just used a different tree-building algorithm, we'd probably want to make another fitting graph to find the new sweet spot.

3. We also will have enough of a conceptual toolkit by that point to understand support vector machines a little better—as being almost equivalent to logistic regression with complexity (overfitting) control.

Either way, a dataset may end up with a very large number of attributes, and using all of them gives the modeling procedure much leeway to fit the training set. You might recall from geometry that in two dimensions you can fit a line to any two points and in three dimensions you can fit a plane to any three points. This concept generalizes: as you increase the dimensionality, you can perfectly fit larger and larger sets of arbitrary points. And even if you cannot fit the dataset perfectly, you can fit it better and better with more dimensions—that is, with more attributes.

Often, modelers carefully prune the attributes in order to avoid overfitting. Modelers will use a sort of holdout technique introduced above to assess the information in the individual attributes. Careful manual attribute selection is a wise practice in cases where considerable human effort can be spent on modeling, and where there are reasonably few attributes. In many modern applications, where large numbers of models are built automatically, and/or where there are very large sets of attributes, manual selection may not be feasible. For example, companies that do data science-driven targeting of online display advertisements can build thousands of models each week, sometimes with millions of possible features. In such cases there is no choice but to employ automatic feature selection (or to ignore feature selection all together).

Example: Overfitting Linear Functions

In "An Example of Mining a Linear Discriminant from Data" on page 88, we introduced a simple dataset called iris, comprising data describing two species of Iris flowers. Now let's revisit that to see the effects of overfitting in action.

Figure 5-4 shows the original Iris dataset graphed with its two attributes, Petal width and Sepal width. Recall that each instance is one flower and corresponds to one dot on the graph. The filled dots are of the species *Iris Setosa* and the circles are instances of the species *Iris Versicolor*. Note several things here: first, the two classes of iris are very distinct and separable. In fact, there is a wide gap between the two "clumps" of instances. Both logistic regression and support vector machines place separating boundaries (lines) in the middle. In fact, the two separating lines are so similiar that they're indistinguishable in the graph.

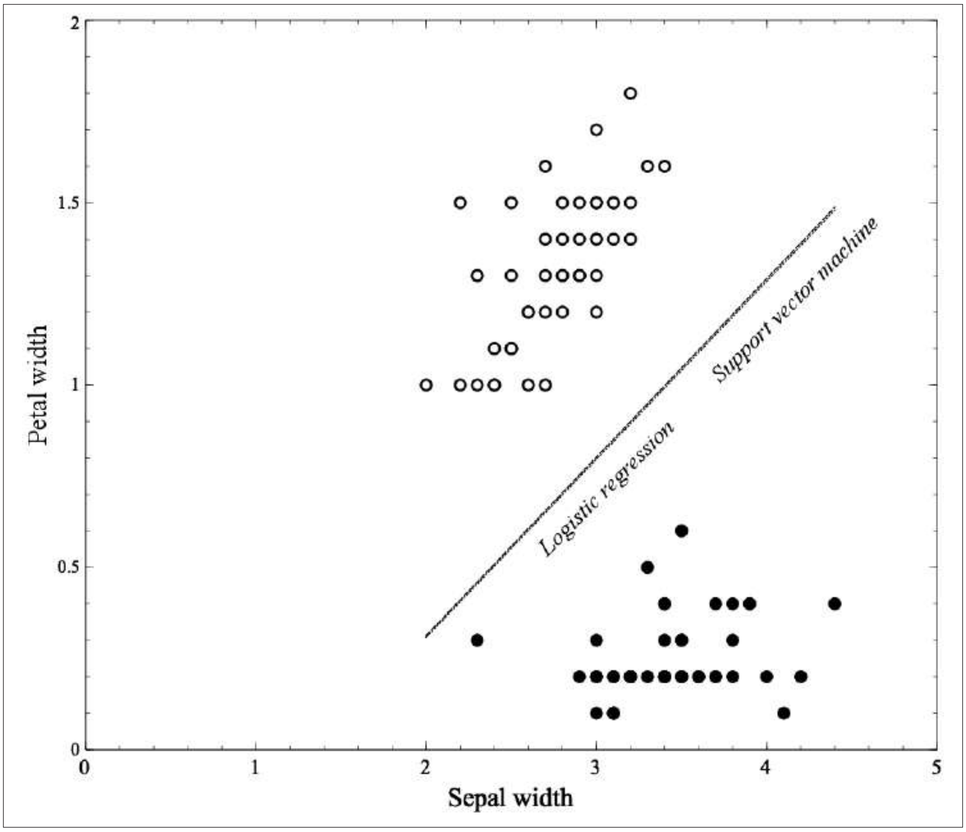


Figure 5-4. The original Iris dataset and the models (boundary lines) that two linear methods learn. In this case, both linear regression and a support vector machine learn the same model (the decision boundary, shown as a line).

In Figure 5-5, we've added a single new example: an *Iris Setosa* point at (3,1). Realistically, we might consider this example to be an outlier or an error since it's much closer to the *Versicolor* examples than the *Setosas*. Notice how the logistic regression line moves in response: it separates the two groups perfectly, while the SVM line barely moves at all.

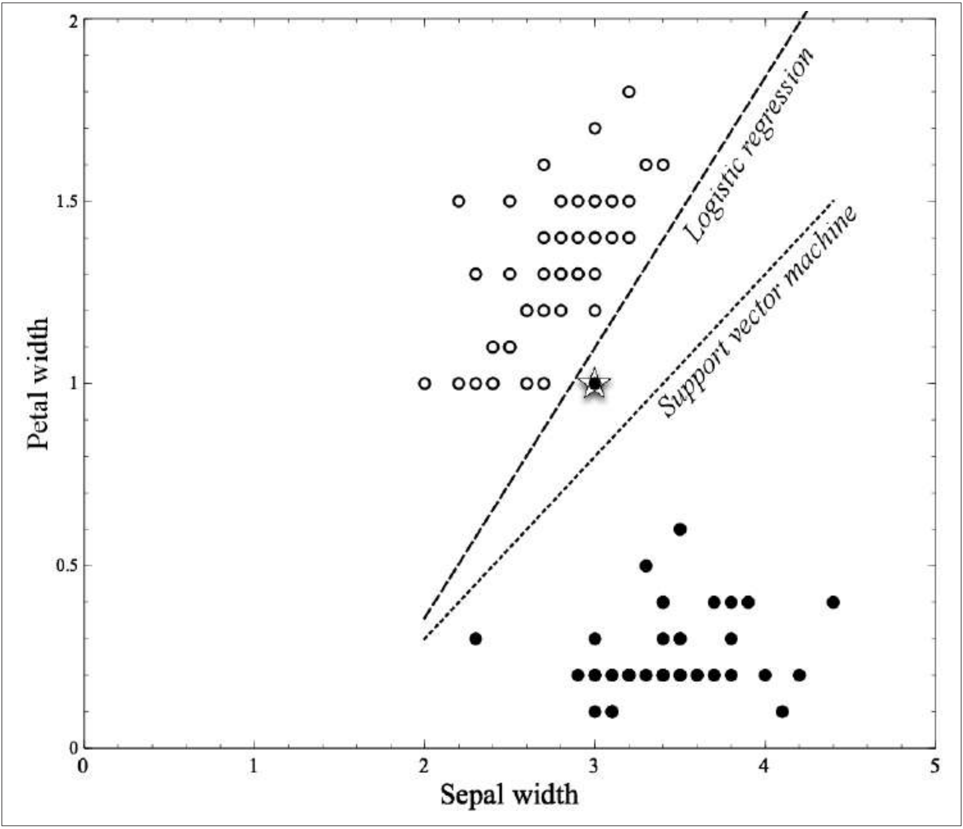


Figure 5-5. The Iris dataset from Figure 5-4 with a single new Iris Setosa example added (shown by star). Note how logistic regression has changed its model considerably.

In Figure 5-6 we've added a different outlier at (4,0.7), this time a *Versicolor* example down in the *Setosa* region. Again, the support vector machine line moves very little in response, but the logistic regression line moves considerably.

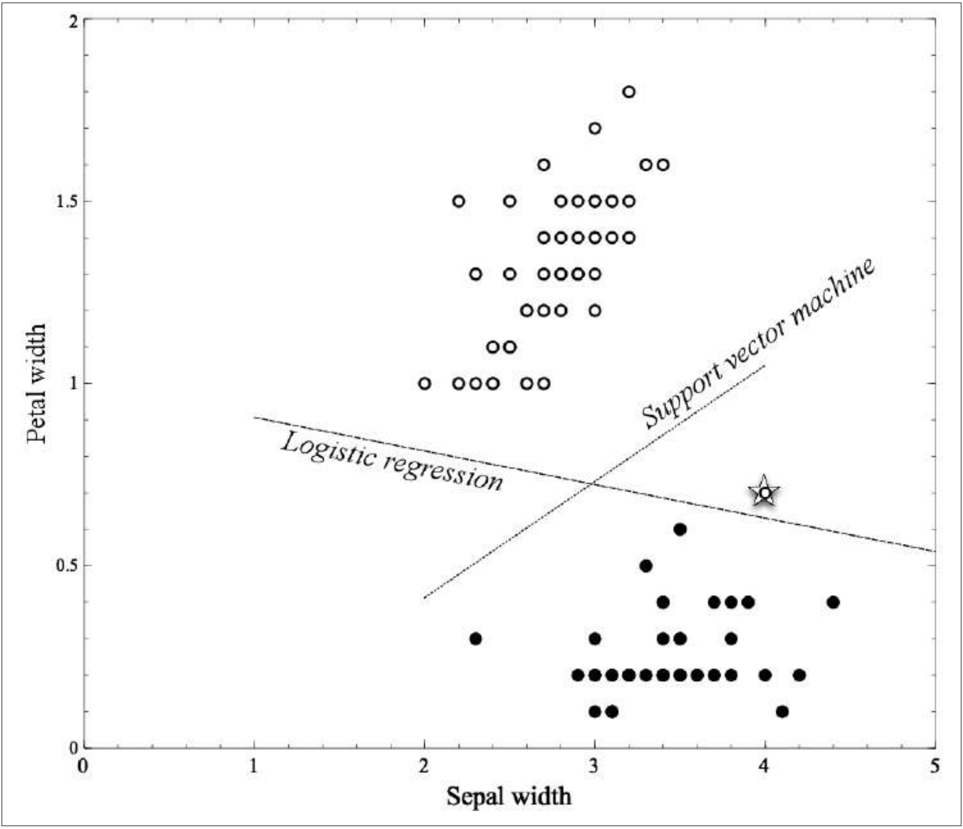


Figure 5-6. The Iris dataset from Figure 5-4 with a single new Iris Versicolor example added (shown by star). Note how logistic regression again changes its model considerably.

In Figure 5-5 and Figure 5-6, Logistic regression appears to be overfitting. Arguably, the examples introduced in each are outliers that should not have a strong influence on the model—they contribute little to the “mass” of the species examples. Yet in the case of logistic regression they clearly do. If a linear boundary exists, logistic regression will find it,[4] even if this means moving the boundary to accommodate outliers. The SVM tends to be less sensitive to individual examples. The SVM training procedure incorporates complexity control, which we will describe technically later.

4. Technically, only *some* logistic regression algorithms are guaranteed to find it. Some do not have this guarantee. However, this fact is not germane to the overfitting point we’re making here.

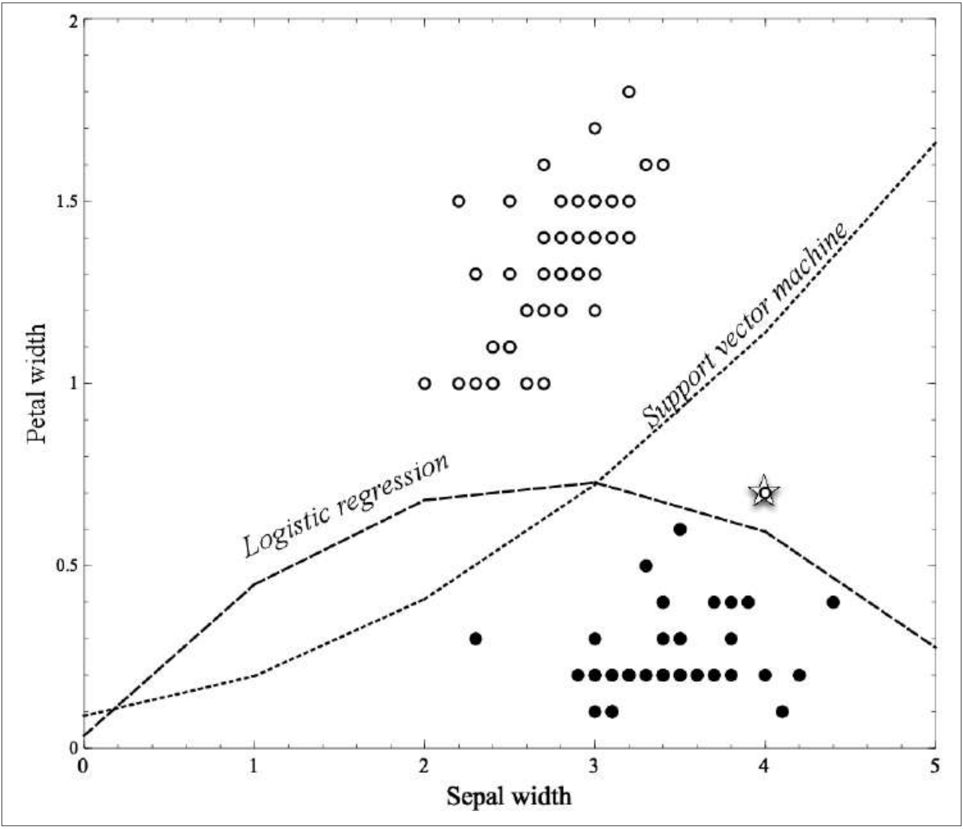


*Figure 5-7. The Iris dataset from Figure 5-6 with its Iris Versicolor example added (shown by star). In this figure, both logistic regression and support vector machine are given an additional feature, **Sepal width^2**, which allows both the freedom to create more complex, nonlinear models (boundaries).*

As we said earlier, another way mathematical functions can become more complex is by adding more variables. In Figure 5-7, we have done just this: we used the same dataset as in Figure 5-6 but we added a single extra attribute, the square of the Sepal width. Providing this attribute gives each method more flexibility in fitting the data because it may assign weights to the squared term. Geometrically, this means the separating boundary can be not just a line but a *parabola*. This additional freedom allows both methods to create curved surfaces that can fit the regions more closely. In cases where curved surfaces may be necessary, this freedom may be necessary, but it also gives the methods far more opportunity to overfit. Note however that the SVM, even though its boundary now is curved, the training procedure still has opted for the larger margin around the boundary, rather than the perfect separation of the positive different classes.

* Example: Why Is Overfitting Bad?

At the beginning of the chapter, we said that a model that only memorizes is useless because it always overfits and is incapable of generalizing. But technically this only demonstrates that overfitting hinders us from improving a model after a certain complexity. It does not explain why overfitting often causes models to become *worse*, as Figure 5-3 shows. This section goes into a detailed example showing how this happens and why. It may be skipped without loss of continuity.

Why does performance degrade? The short answer is that as a model gets more complex it is allowed to pick up harmful spurious correlations. These correlations are idiosyncracies of the specific training set used and do not represent characteristics of the population in general. The harm occurs when these spurious correlations produce *incorrect* generalizations in the model. This is what causes performance to decline when overfitting occurs. In this section we go through an example in detail to show how this can happen.

Table 5-1. A small set of training examples

Instance	x	y	Class
1	p	r	c_1
2	p	r	c_1
3	p	r	c_1
4	q	s	c_1
5	p	s	c_2
6	q	r	c_2
7	q	s	c_2
8	q	r	c_2

Consider a simple two-class problem with classes c_1 and c_2 and attributes x and y. We have a population of examples, evenly balanced between the classes. Attribute x has two values, p and q, and y has two values, r and s. In the general population, $x = p$ occurs 75% of the time in class c_1 examples and in 25% of the c_2 examples, so x provides some prediction of the class. By design, y has no predictive power at all, and indeed we see that in the data sample both of y's values occur in both classes equally. In short, the instances in this domain are difficult to separate, with only x providing some predictive power. The best we can achieve is 75% accuracy by looking at x.

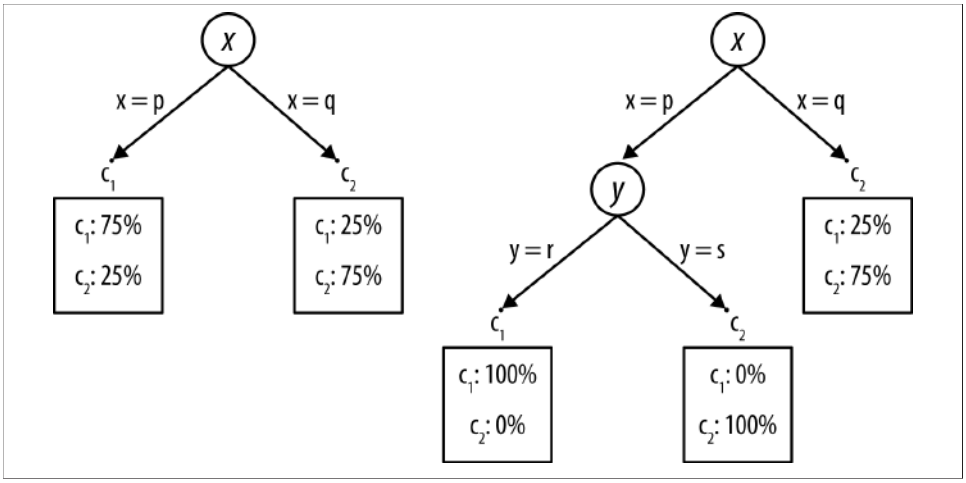


Figure 5-8. Classification trees for the overfitting example. (a) The optimal tree has only three nodes. (b) An overfit tree, which fits the training data better, has worse generalization accuracy because the extraneous structure makes suboptimal predictions.

Table 5-1 shows a very small training set of examples from this domain. What would a classification tree learner do with these? We won't go into the entropy calculations, but attribute *x* provides some leverage so a tree learner would split on it and create the tree shown in Figure 5-8. Since *x* provides the only leverage, this should be the optimal tree. Its error rate is 25%—equal to the theoretical minimum error rate.

However, observe from Table 5-1 that *in this particular* dataset *y*'s values of *r* and *s* are not evenly split between the classes, so *y* does seem to provide some predictiveness. Specifically, once we choose *x*=p (instances 1-4), we see that *y*=r predicts c_1 perfectly (instances 1-3). Hence, from this dataset, tree induction would achieve information gain by splitting on *y*'s values and create two new leaf nodes, shown in Figure 5-8.

Based on our training set, the tree in (b) performs well, better than (a). It classifies seven of the eight training examples correctly, whereas the tree in (a) classifies only six out of eight correct. But this is due to the fact that *y=r* purely by chance correlates with class c_1 in this data sample; in the general population there is no such correlation. We have been misled, and the extra branch in (b) is not simply extraneous, it is harmful. Recall that we defined the general population to have *x*=p occuring in 75% of the class c_1 examples and 25% of the c_2 examples. But the spurious *y=s* branch predicts c_2, which is wrong in the general population. In fact, we expect this spurious branch to contribute one in eight errors made by the tree. Overall, the (b) tree will have a total expected error rate of 30%, while (a) will have an error rate of 25%.

We conclude this example by emphasizing several points. First, this phenomenon is not particular to classification trees. Trees are convenient for this example because it is easy to point to a portion of a tree and declare it to be spurious, but all model types are susceptible to overfitting effects. Second, this phenomenon is not due to the training data in Table 5-1 being atypical or biased. Every dataset is a finite sample of a larger population, and every sample will have variations even when there is no bias in the sampling. Finally, as we have said before, there is no general analytic way to determine in advance whether a model has overfit or not. In this example we defined what the population looked like so we could declare that a given model had overfit. In practice, you will not have such knowledge and it will be necessary to use a holdout set to detect overfitting.

From Holdout Evaluation to Cross-Validation

Later we will present a general technique in broad use to try to avoid overfitting, which applies to attribute selection as well as tree complexity, and beyond. But first, we need to discuss holdout evaluation in more detail. Before we can work to avoid overfitting, we need to be able to avoid being fooled by overfitting. At the beginning of this chapter we introduced the idea that in order to have a fair evaluation of the generalization performance of a model, we should estimate its accuracy on holdout data—data not used in building the model, but for which we do know the actual value of the target variable. Holdout testing is similar to other sorts of evaluation in a "laboratory" setting.

While a holdout set will indeed give us an estimate of generalization performance, it is just a single estimate. Should we have any confidence in a single estimate of model accuracy? It might have just been a single particularly lucky (or unlucky) choice of training and test data. We will not go into the details of computing confidence intervals on such quantities, but it is important to discuss a general testing procedure that will end up helping in several ways.

Cross-validation is a more sophisticated holdout training and testing procedure. We would like not only a simple estimate of the generalization performance, but also some statistics on the estimated performance, such as the mean and variance, so that we can understand how the performance is expected to vary across datasets. This variance is critical for assessing confidence in the performance estimate, as you might have learned in a statistics class.

Cross-validation also makes better use of a limited dataset. Unlike splitting the data into one training and one holdout set, cross-validation computes its estimates over *all* the data by performing multiple splits and systematically swapping out samples for testing.

Sidebar: Building a modeling "laboratory"

Building the infrastructure for a modeling lab may be costly and time consuming, but after this investment many aspects of model performance can be evaluated quickly in a controlled environment. However, holdout testing cannot capture all the complexities of the real world where the model will be used. Data scientists should work to understand the actual use scenario so as to make the lab setting as much like it as possible, to avoid surprises when the two do not match. For example, consider a company that wants to use data science to improve its targeting of costly personally targeted advertisements. As a campaign progresses, more and more data arrive on people who make purchases after having seen the ad versus those who do not. These data can be used to build models to discriminate between those to whom we should and should not advertise. Examples can be put aside to evaluate how accurate the models are in predicting whether consumers will respond to the ad.

When the resultant models are put into production, targeting consumers "in the wild," the company is surprised that the models do not work as well as they did in the lab. Why not? There could be many reasons, but notice one in particular: the training and holdout data do not really match the data to which the model will be applied in the field. Specifically, the training data all are consumers who had been targeted in the campaign. Otherwise, we would not know the value of the target variable (whether they responded). Even before the data mining, the company did not simply target randomly; they had some criteria for targeting people they believed would respond. In the field, the model is applied to consumers more broadly—not just to consumers who meet these criteria. The fact that the training and deployment populations are different is a likely source of performance degradation.

This phenomenon is not limited to advertisement targeting. Consider credit scoring, where we would like to build models to predict the likelihood of a consumer defaulting on credit. Again, the data we have on write-offs versus non-write-offs are based on those to whom we previously extended credit, who presumably were those thought to be low risk.

In both of these cases, think about what you might do as a business to gather a more appropriate dataset from which to build predictive models. Remember to apply the fundamental concept introduced in Chapter 1: think of data as an asset in which you may want to *invest*.

Cross-validation begins by splitting a labeled dataset into *k* partitions called *folds*. Typically, *k* will be five or ten. The top pane of Figure 5-9 shows a labeled dataset (the original dataset) split into five folds. Cross-validation then iterates training and testing *k* times, in a particular way. As depicted in the bottom pane of Figure 5-9, in each iteration of the cross-validation, a different fold is chosen as the test data. In this iteration, the other *k*–1 folds are combined to form the training data. So, in each iteration we have (*k*–1)/*k* of the data used for training and 1/*k* used for testing.

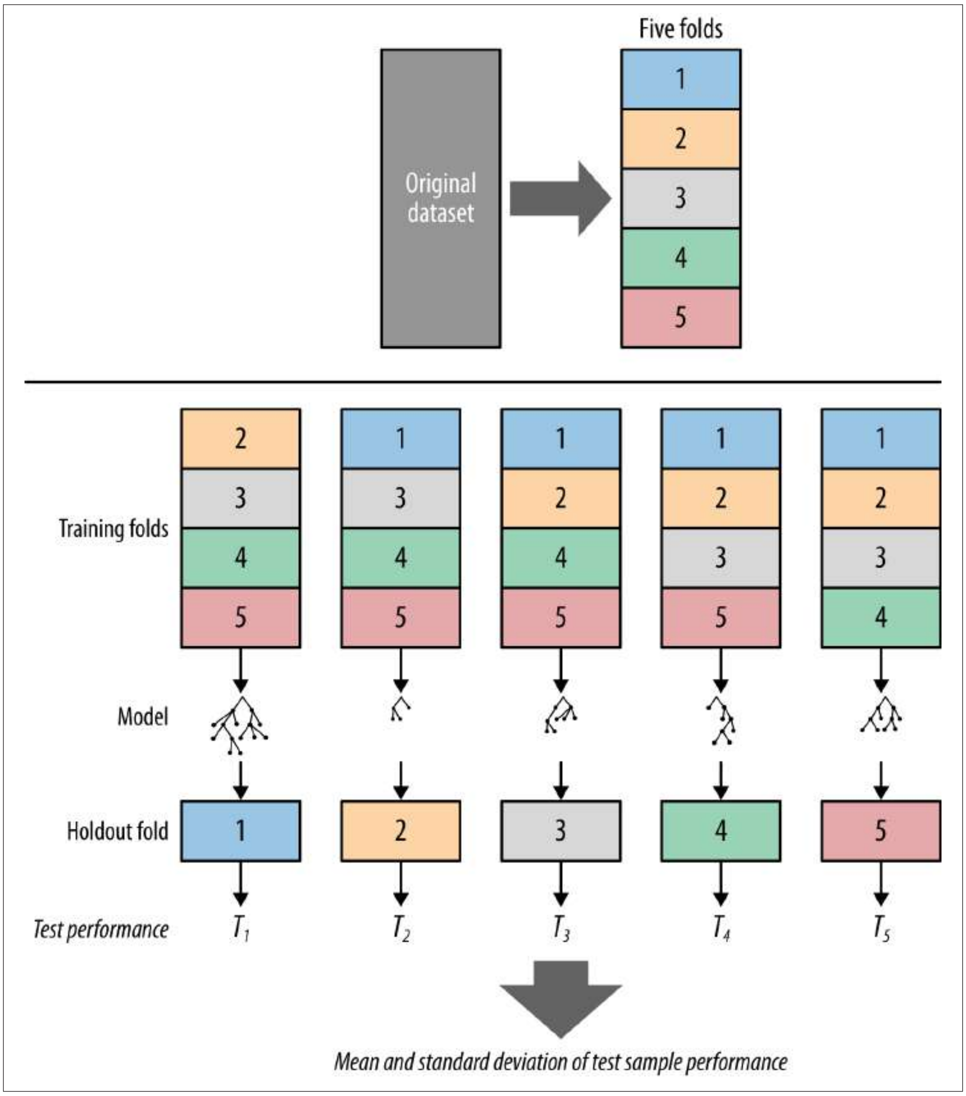


Figure 5-9. An illustration of cross-validation. The purpose of cross-validation is to use the original labeled data efficiently to estimate the performance of a modeling procedure. Here we show five-fold cross-validation: the original dataset is split randomly into five equal-sized pieces. Then, each piece is used in turn as the test set, with the other four used to train a model. The result is five different accuracy results, which then can be used to compute the average accuracy and its variance.

Each iteration produces one model, and thereby one estimate of generalization performance, for example, one estimate of accuracy. When cross-validation is finished, every example will have been used only once for testing but *k*–1 times for training. At this point we have performance estimates from all the *k* folds and we can compute the average and standard deviation.

The Churn Dataset Revisited

Consider again the churn dataset introduced in "Example: Addressing the Churn Problem with Tree Induction" on page 73. In that section we used the entire dataset both for training and testing, and we reported an accuracy of 73%. We ended that section by asking a question, *Do you trust this number?* By this point you should know enough to mistrust any performance measurement done on the training set, because overfitting is a very real possibility. Now that we have introduced cross-validation we can redo the evaluation more carefully.

Figure 5-10 shows the results of ten-fold cross-validation. In fact, two model types are shown. The top graph shows results with logistic regression, and the bottom graph shows results with classification trees. To be precise: the dataset was first shuffled, then divided into ten partitions. Each partition in turn served as a single holdout set while the other nine were collectively used for training. The horizontal line in each graph is the average of accuracies of the ten models of that type.

There are several things to observe here. First, the average accuracy of the folds with classification trees is 68.6%—significantly lower than our previous measurement of 73%. This means there was some overfitting occurring with the classification trees, and this new (lower) number is a more realistic measure of what we can expect. Second, there is variation in the performances in the different folds (the standard deviation of the fold accuracies is 1.1), and thus it is a good idea to average them to get a notion of the performance as well as the variation we can expect from inducing classification trees on this dataset.

Finally, compare the fold accuracies between logistic regression and classification trees. There are certain commonalities in both graphs—for example, neither model type did very well on Fold Three and both performed well on Fold Ten. But there are definite differences between the two. An important thing to notice is that logistic regression models show slightly lower average accuracy (64.1%) and with higher variation (standard deviation of 1.3) than the classification trees do. On this particular dataset, trees may be preferable to logistic regression because of their greater stability and performance. But this is not absolute; other datasets will produce different results, as we shall see.

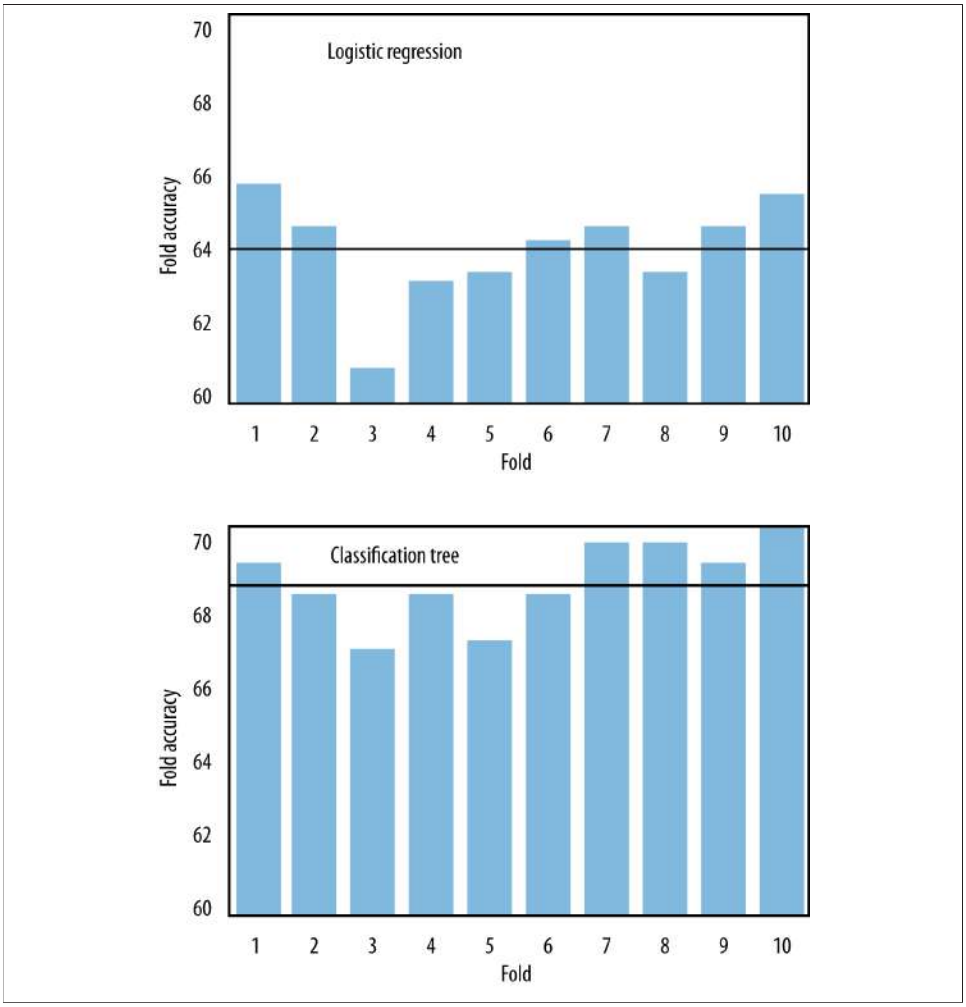


Figure 5-10. Fold accuracies for cross-validation on the churn problem. At the top are accuracies of logistic regression models trained on a dataset of 20,000 instances divided into ten folds. At the bottom are accuracies of classification trees on the same folds. In each graph the horizontal line shows the average accuracy of the folds. (Note the selection of the range of the y axis, which emphasizes the differences in accuracy.)

Learning Curves

If training set size changes, you may also expect different generalization performance from the resultant model. All else being equal, the generalization performance of data-driven modeling generally improves as more training data become available, up to a point. A plot of the generalization performance against the amount of training data is called a *learning curve*. The learning curve is another important analytical tool.

Learning curves for tree induction and logistic regression are shown in Figure 5-11 for the telecommunications churn problem.[5] Learning curves usually have a characteristic shape. They are steep initially as the modeling procedure finds the most apparent regularities in the dataset. Then as the modeling procedure is allowed to train on larger and larger datasets, it finds more accurate models. However, the marginal advantage of having more data decreases, so the learning curve becomes less steep. In some cases, the curve flattens out completely because the procedure can no longer improve accuracy even with more training data.

It is important to understand the difference between learning curves and fitting graphs (or fitting curves). A learning curve shows the generalization performance—the performance only on testing data, plotted against the *amount of training data* used. A fitting graph shows the generalization performance as well as the performance on the training data, but plotted against model *complexity*. Fitting graphs generally are shown for a fixed amount of training data.

Even on the same data, different modeling procedures can produce very different learning curves. In Figure 5-11, observe that for smaller training-set sizes, logistic regression yields better generalization accuracy than tree induction. However, as the training sets get larger, the learning curve for logistic regression levels off faster, the curves cross, and tree induction soon is more accurate. This performance relates back to the fact that with more flexibility comes more overfitting. Given the same set of features, classification trees are a more flexible model representation than linear logistic regression. This means two things: for smaller data, tree induction will tend to overfit more. Often, as we see for the data in Figure 5-11, this leads logistic regression to perform better for smaller datasets (not always, though). On the other hand, the figure also shows that the flexibility of tree induction can be an advantage with larger training sets: the tree can represent substantially nonlinear relationships between the features and the target. Whether the tree induction can actually capture those relationships needs to be evaluated empirically—using an analytical tool such as learning curves.

5. Perlich et al. (2003) show learning curves for tree induction and logistic regression for dozens of classification problems.

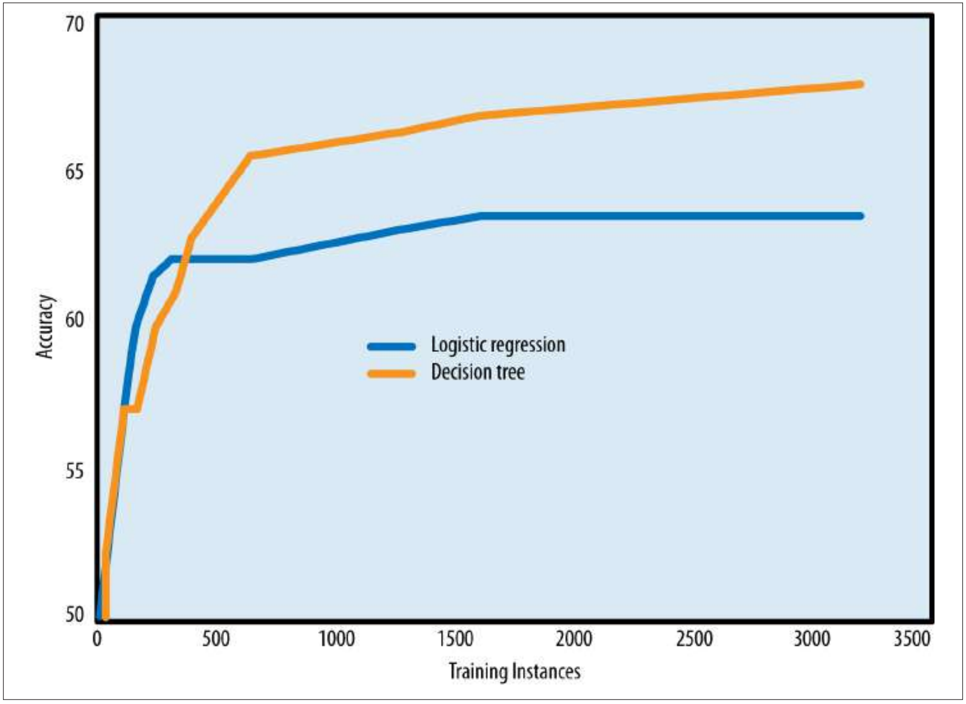


Figure 5-11. Learning curves for tree induction and logistic regression for the churn problem. As the training size grows (x axis), generalization performance (y axis) improves. Importantly, the improvement rates are different for the two induction technique, and change over time. Logistic regression has less flexibility, which allows it to overfit less with small data, but keeps it from modeling the full complexity of the data. Tree induction is much more flexible, leading it to overfit more with small data, but to model more complex regularities with larger training sets.

The learning curve has additional analytical uses. For example, we've made the point that data can be an asset. The learning curve may show that generalization perforance has leveled off so investing in more training data is probably not worthwhile; instead, one should accept the current performance or look for another way to improve the model, such as by devising better features. Alternatively, the learning curve might show generalization accuracy continuing to improve, so obtaining more training data could be a good investment.

Overfitting Avoidance and Complexity Control

To avoid overfitting, we control the complexity of the models induced from the data. Let's start by examining complexity control in tree induction, since tree induction has much flexibility and therefore will tend to overfit a good deal without some mechanism to avoid it. This discussion in the context of trees will lead us to a very general mechanism that will be applicable to other models.

Avoiding Overfitting with Tree Induction

The main problem with tree induction is that it will keep growing the tree to fit the training data until it creates pure leaf nodes. This will likely result in large, overly complex trees that overfit the data. We have seen how this can be detrimental. Tree induction commonly uses two techniques to avoid overfitting. These strategies are (i) to stop growing the tree before it gets too complex, and (ii) to grow the tree until it is too large, then "prune" it back, reducing its size (and thereby its complexity).

There are various methods for accomplishing both. The simplest method to limit tree size is to specify a minimum number of instances that must be present in a leaf. The idea behind this minimum-instance stopping criterion is that for predictive modeling, we essentially are using the data at the leaf to make a statistical estimate of the value of the target variable for future cases that would fall to that leaf. If we make predictions of the target based on a very small subset of data, we might expect them to be inaccurate —especially when we built the tree specifically to try to get pure leaves. A nice property of controlling complexity in this way is that tree induction will automatically grow the tree branches that have a lot of data and cut short branches that have fewer data—thereby automatically adapting the model based on the data distribution.

A key question becomes what threshold we should use. How few instances are we willing to tolerate at a leaf? Five instances? Thirty? One hundred? There is no fixed number, although practitioners tend to have their own preferences based on experience. However, researchers have developed techniques to decide the stopping point statistically. Statistics provides the notion of a "hypothesis test," which you might recall from a basic statistics class. Roughly, a hypothesis test tries to assess whether a difference in some statistic is not due simply to chance. In most cases, the hypothesis test is based on a "p-value," which gives a limit on the probability that the difference in statistic is due to chance. If this value is below a threshold (often 5%, but problem specific), then the hypothesis test concludes that the difference is likely not due to chance. So, for stopping tree growth, an alternative to setting a fixed size for the leaves is to conduct a hypothesis test at every leaf to determine whether the observed difference in (say) information gain could have been due to chance. If the hypothesis test concludes that it was likely not due to chance, then the split is accepted and the tree growing continues. (See "Sidebar: Beware of "multiple comparisons"" on page 139.)

The second strategy for reducing overfitting is to "prune" an overly large tree. Pruning means to cut off leaves and branches, replacing them with leaves. There are many ways to do this, and the interested reader can look into the data mining literature for details. One general idea is to estimate whether replacing a set of leaves or a branch with a leaf would reduce accuracy. If not, then go ahead and prune. The process can be iterated on progressive subtrees until any removal or replacement would reduce accuracy.

We conclude our example of avoiding overfitting in tree induction with the method that will generalize to many different data modeling techniques. Consider the following idea: what if we built trees with all sorts of different complexities? For example, say we stop building the tree after only one node. Then build a tree with two nodes. Then three nodes, etc. We have a set of trees of different complexities. Now, if only there were a way to estimate their generalization performance, we could pick the one that is (estimated to be) the best!

A General Method for Avoiding Overfitting

More generally, if we have a collection of models with different complexities, we could choose the best simply by estimating the generalization performance of each. But how could we estimate their generalization performance? On the (labeled) test data? There's one big problem with that: test data should be strictly *independent* of model building so that we can get an independent estimate of model accuracy. For example, we might want to estimate the ultimate business performance or to compare the best model we can build from one family (say, classification trees) against the best model from another family (say, logistic regression). If we don't care about comparing models or getting an independent estimate of the model accuracy and/or variance, then we could pick the best model based on the testing data.

However, even if we do want these things, we still can proceed. The key is to realize that there was nothing special about the first training/test split we made. Let's say we are saving the test set for a final assessment. We can take the training set and split it again into a training subset and a testing subset. Then we can build models on this training subset and pick the best model based on this testing subset. Let's call the former the *subtraining set* and the latter the *validation set* for clarity. The validation set is separate from the final test set, on which we are never going to make any modeling decisions. This procedure is often called *nested* holdout testing.

Returning to our classification tree example, we can induce trees of many complexities from the subtraining set, then we can estimate the generalization performance for each from the validation set. This would correspond to choosing the top of the inverted-U-shaped holdout curve in Figure 5-3. Say the best model by this assessment has a complexity of 122 nodes (the "sweet spot"). Then we could use this model as our best choice, possibly estimating the actual generalization performance on the final holdout test set. We also could add one more twist. This model was built on a subset of our training data, since we had to hold out the validation set in order to choose the complexity. But once we've chosen the complexity, why not induce a *new* tree with 122 nodes from the whole, original training set? Then we might get the best of both worlds: using the subtraining/validation split to pick the best complexity without tainting the test set, *and* building a model of this best complexity on the entire training set (subtraining plus validation).

This approach is used in many sorts of modeling algorithms to control complexity. The general method is to choose the value for some complexity parameter by using some sort of nested holdout procedure. Again, it is nested because a second holdout procedure is performed on the training set selected by the first holdout procedure.

Often, nested cross-validation is used. Nested cross-validation is more complicated, but it works as you might suspect. Say we would like to do cross-validation to assess the generalization accuracy of a new modeling technique, which has an adjustable complexity parameter *C*, but we do not know how to set it. So, we run cross-validation as described above. However, before building the model for each fold, we take the training set (refer to Figure 5-9) and first run an experiment: we run another entire cross-validation on just that training set to find the value of *C* estimated to give the best accuracy. The result of that experiment is used only to set the value of *C* to build the actual model for that fold of the cross-validation. Then we build another model using the entire training fold, using that value for *C*, and test on the corresponding test fold. The only difference from regular cross-validation is that for each fold we first run this experiment to find *C*, using another, smaller, cross-validation.

If you understood all that, you would realize that if we used 5-fold cross-validation in both cases, we actually have built 30 total models in the process (yes, *thirty*). This sort of experimental complexity-controlled modeling only gained broad practical application over the last decade or so, because of the obvious computational burden involved.

This idea of using the data to choose the complexity experimentally, as well as to build the resulting model, applies across different induction algorithms and different sorts of complexity. For example, we mentioned that complexity increases with the size of the feature set, so it is usually desirable to cull the feature set. A common method for doing this is to run with many different feature sets, using this sort of nested holdout procedure to pick the best.

For example, *sequential forward selection* (SFS) of features uses a nested holdout procedure to first pick the best individual feature, by looking at all models built using just one feature. After choosing a first feature, SFS tests all models that add a second feature to this first chosen feature. The best pair is then selected. Next the same procedure is done for three, then four, and so on. When adding a feature does not improve classification accuracy on the validation data, the SFS process stops. (There is a similar procedure called *sequential backward elimination* of features. As you might guess, it works by starting with all features and discarding features one at a time. It continues to discard features as long as there is no performance loss.)

This is a common approach. In modern environments with plentiful data and computational power, the data scientist routinely sets modeling parameters by experimenting using some tactical, nested holdout testing (often nested cross-validation).

The next section shows a different way that this method applies to controlling overfitting when learning numerical functions (as described in Chapter 4). We urge you to at least skim the following section because it introduces concepts and vocabulary in common use by data scientists these days.

* Avoiding Overfitting for Parameter Optimization

As just described, avoiding overfitting involves complexity control: finding the "right" balance between the fit to the data and the complexity of the model. In trees we saw various ways for trying to keep the tree from getting too big (too complex) when fitting the data. For equations, such as logistic regression, that unlike trees do not automatically select what attributes to include, complexity can be controlled by choosing a "right" set of attributes.

Chapter 4 introduced the popular family of methods that builds models by explicitly optimizing the fit to the data via a set of numerical parameters. We discussed various linear members of this family, including linear discriminant learners, linear regression, and logistic regression. Many nonlinear models are fit to the data in exactly the same way.

As might be expected given our discussion so far in this chapter and the figures in "Example: Overfitting Linear Functions" on page 119, these procedures also can overfit the data. However, their explicit optimization framework provides an elegant, if technical, method for complexity control. The general strategy is that instead of just optimizing the fit to the data, we optimize some combination of fit and simplicity. Models will be better if they fit the data better, but they also will be better if they are simpler. This general methodology is called *regularization*, a term that is heard often in data science discussions.

The rest of this section discusses briefly (and slightly technically) how regularization is done. Don't worry if you don't really understand the technical details. Do remember that regularization is trying to optimize not just the fit to the data, but a combination of fit to the data and simplicity of the model.

Recall from Chapter 4 that to fit a model involving numeric parameters *w* to the data we find the set of parameters that maximizes some "objective function" indicating how well it fits the data:

$$\underset{\mathbf{w}}{\arg\max}\ \text{fit}(\mathbf{x}, \mathbf{w})$$

(The arg $\max_{\mathrm{w}}$ just means that you want to maximize the fit over all possible arguments **w**, and are interested in the particular argument **w** that gives the maximum. These would be the parameters of the final model.)

Complexity control via regularization works by adding to this objective function a penalty for complexity:

$$\underset{\mathbf{w}}{\arg\max}\ [\text{fit}(\mathbf{x}, \mathbf{w}) - \lambda \cdot \text{penalty}(\mathbf{w})]$$

The λ term is simply a weight that determines how much importance the optimization procedure should place on the penalty, compared to the data fit. At this point, the modeler has to choose λ and the penalty function.

So, as a concrete example, recall from "* Logistic Regression: Some Technical Details" on page 99 that to learn a standard logistic regression model, from data, we find the numeric parameters **w** that yield the linear model most likely to have generated the observed data—the "maximum likelihood" model. Let' represent that as:

$$\underset{\mathbf{w}}{\arg\max}\ g_{\text{likelihood}}(\mathbf{x}, \mathbf{w})$$

To learn a "regularized" logistic regression model we would instead compute:

$$\underset{\mathbf{w}}{\arg\max}\ [g_{\text{likelihood}}(\mathbf{x}, \mathbf{w}) - \lambda \cdot \text{penalty}(\mathbf{w})]$$

There are different penalties that can be applied, with different properties.[6] The most commonly used penalty is the sum of the squares of the weights, sometimes called the "L2-norm" of *w*. The reason is technical, but basically functions can fit data better if they are allowed to have very large positive and negative weights. The sum of the squares of the weights gives a large penalty when weights have large absolute values.

6. The book *The Elements of Statistical Learning* (Hastie, Tibshirani, & Friedman, 2009) contains an excellent technical discussion of these.

If we incorporate the L2-norm penalty into standard least-squares linear regression, we get the statistical procedure called *ridge regression*. If instead we use the sum of the absolute values (rather than the squares), known as the L1-norm, we get a procedure known as the *lasso* (Hastie et al., 2009). More generally, this is called L1-regularization. For reasons that are quite technical, L1-regularization ends up zeroing out many coefficients. Since these coefficients are the multiplicative weights on the features, L1-regularization effectively performs an automatic form of feature selection.

Now we have the machinery to describe in more detail the linear support vector machine, introduced in "Support Vector Machines, Briefly" on page 91. There we waved our hands and told you that the support vector machine "maximizes the margin" between the classes by fitting the "fattest bar" between the classes. Separately we discussed that it uses hinge loss (see "Sidebar: Loss functions" on page 94) to penalize errors. We now can connect these together, and directly to logistic regression. Specifically, linear support vector machine learning is almost equivalent to the L2-regularized logistic regression just discussed; the only difference is that a support vector machine uses hinge loss instead of likelihood in its optimization. The support vector machine optimizes this equation:

$$\underset{\mathbf{w}}{\arg\max}\ \left[- g_{\text{hinge}}(\mathbf{x}, \mathbf{w}) - \lambda \cdot \text{penalty}(\mathbf{w}) \right]$$

where g_{hinge}, the hinge loss term, is negated because lower hinge loss is better.

Finally, you may be saying to yourself: all this is well and good, but a lot of magic seems to be hidden in this λ parameter, which the modeler has to choose. How in the world would the modeler choose that for some real domain like churn prediction, or online ad targeting, or fraud detection?

It turns out that we already have a straightforward way to choose λ. We've discussed how a good tree size and a good feature set can be chosen via nested cross-validation on the training data. We can choose λ the same way. This cross-validation would essentially conduct automated experiments on subsets of the training data and find a good λ value. Then this λ would be used to learn a regularized model on all the training data. This has become the standard procedure for building numerical models that give a good balance between data fit and model complexity. This general approach to optimizing the parameter values of a data mining procedure is known as grid search.

Sidebar: Beware of “multiple comparisons”

Consider the following scenario. You run an investment firm. Five years ago, you wanted to have some marketable small-cap mutual fund products to sell, but your analysts had been awful at picking small-cap stocks. So you undertook the following procedure. You started 1,000 different mutual funds, each including a small set of stocks randomly chosen from those that make up the Russell 2000 index (the main index for small-cap stocks). Your firm invested in all 1,000 of these funds, but told no one about them. Now, five years later, you look at their performance. Since they have different stocks in them, they will have had different returns. Some will be about the same as the index, some will be worse, and some will be better. The best one might be a lot better. Now, you liquidate all the funds but the best few, and you present these to the public. You can “honestly” claim that their 5-year return is substantially better than the return of the Russell 2000 index.

So, what’s the problem? The problem is that you randomly chose the stocks! You have no idea whether the stocks in these “best” funds performed better because they indeed are fundamentally better, or because you cherry-picked the best from a large set that simply varied in performance. If you flip 1,000 fair coins many times each, one of them will have come up heads much more than 50% of the time. However, choosing that coin as the “best” of the coins for later flipping obviously is silly. These are instances of “the problem of multiple comparisons,” a very important statistical phenomenon that business analysts and data scientists should always keep in mind. Beware whenever someone does many tests and then picks the results that look good. Statistics books will warn against running multiple statistical hypothesis tests, and then looking at the ones that give “significant” results. These usually violate the assumptions behind the statistical tests, and the actual significance of the results is dubious.

The underlying reasons for overfitting when building models from data are essentially problems of multiple comparisons (Jensen & Cohen, 2000). Note that even the procedures for avoiding overfitting themselves undertake multiple comparisons (e.g., choosing the best complexity for a model by comparing many complexities). There is no silver bullet or magic formula to truly get “the optimal” model to fit the data. Nonetheless, care can be taken to reduce overfitting as much as possible, by using the holdout procedures described in this chapter and if possible by looking carefully at the results before declaring victory. For example, if the fitting graph truly has an inverted-U-shape, one can be much more confident that the top represents a “good” complexity than if the curve jumps around randomly.

Summary

Data mining involves a fundamental trade-off between model complexity and the possibility of overfitting. A complex model may be necessary if the phenomenon producing the data is itself complex, but complex models run the risk of overfitting training data (i.e., modeling details of the data that are not found in the general population). An overfit model will not generalize to other data well, even if they are from the same population.

All model types can be overfit. There is no single choice or technique to eliminate overfitting. The best strategy is to recognize overfitting by testing with a holdout set. Several types of curves can help detect and measure overfitting. A *fitting graph* has two curves showing the model performance on the training and testing data as a function of model complexity. A fitting curve on testing data usually has an approximate U or inverted-U-shape (depending on whether error or accuracy is plotted). The accuracy starts off low when the model is simple, increases as complexity increases, flattens out, then starts to decrease again as overfitting sets in. A *learning curve* shows model performance on testing data plotted against the *amount of training data* used. Usually model performance increases with the amount of data, but the rate of increase and the final asymptotic performance can be quite different between models.

A common experimental methodology called *cross-validation* specifies a systematic way of splitting up a single dataset such that it generates multiple performance measures. These values tell the data scientist what average behavior the model yields as well as the variation to expect.

The general method for reining in model complexity to avoid overfitting is called model *regularization*. Techniques include tree pruning (cutting a classification tree back when it has become too large), feature selection, and employing explicit complexity penalties into the objective function used for modeling.

CHAPTER 6

Similarity, Neighbors, and Clusters

Fundamental concepts: *Calculating similarity of objects described by data; Using similarity for prediction; Clustering as similarity-based segmentation.*

Exemplary techniques: *Searching for similar entities; Nearest neighbor methods; Clustering methods; Distance metrics for calculating similarity.*

Similarity underlies many data science methods and solutions to business problems. If two things (people, companies, products) are similar in some ways they often share other characteristics as well. Data mining procedures often are based on grouping things by similarity or searching for the "right" sort of similarity. We saw this implicitly in previous chapters where modeling procedures create boundaries for grouping instances together that have similar values for their target variables. In this chapter we will look at similarity directly, and show how it applies to a variety of different tasks. We include sections with some technical details, in order that the more mathematical reader can understand similarity in more depth; these sections can be skipped.

Different sorts of business tasks involve reasoning from similar examples:

- We may want to *retrieve* similar things directly. For example, IBM wants to find companies that are similar to their best business customers, in order to have the sales staff look at them as prospects. Hewlett-Packard maintains many high-performance servers for clients; this maintenance is aided by a tool that, given a server configuration, retrieves information on other similarly configured servers. Advertisers often want to serve online ads to consumers who are similar to their current good customers.
- Similarity can be used for doing *classification* and *regression*. Since we now know a good bit about classification, we will illustrate the use of similarity with a classification example below.
- We may want to group similar items together into *clusters*, for example to see whether our customer base contains groups of similar customers and what these

groups have in common. Previously we discussed supervised segmentation; this is unsupervised segmetation. After discussing the use of similarity for classification, we will discuss its use for clustering.

- Modern retailers such as Amazon and Netflix use similarity to provide *recommendations* of similar products or from similar people. Whenever you see statements like "People who like X also like Y" or "Customers with your browsing history have also looked at …" similarity is being applied. In Chapter 12, we will discuss how a customer can be similar to a movie, if the two are described by the same "taste dimensions." In this case, to make recommendations we can find the movies that are most similar to the customer (and which the customer has not already seen).
- Reasoning from similar cases of course extends beyond business applications; it is natural to fields such as medicine and law. A doctor may reason about a new difficult case by recalling a similar case (either treated personally or documented in a journal) and its diagnosis. A lawyer often argues cases by citing legal precedents, which are similar historical cases whose dispositions were previously judged and entered into the legal casebook. The field of Artificial Intelligence has a long history of building systems to help doctors and lawyers with such case-based reasoning. Similarity judgments are a key component.

In order to discuss these applications further, we need to take a minute to formalize similarity and its cousin, distance.

Similarity and Distance

Once an object can be represented as data, we can begin to talk more precisely about the similarity between objects, or alternatively the distance between objects. For example, let's consider the data representation we have used throughout the book so far: represent each object as a feature vector. Then, the closer two objects are in the space defined by the features, the more similar they are.

Recall that when we build and apply predictive models, the goal is to determine the value of a target characteristic. In doing so, we've used the implicit similarity of objects already. "Visualizing Segmentations" on page 67 discussed the geometric interpretation of some classification models and "Classification via Mathematical Functions" on page 83 discussed how two different model types divide up an instance space into regions based on closeness of instances with similar class labels. Many methods in data science may be seen in this light: as methods for organizing the space of data instances (representations of important objects) so that instances near each other are treated similarly for some purpose. Both classification trees and linear classifiers establish boundaries between regions of differing classifications. They have in common the view that instances sharing a common region in space should be similar; what differs between the methods is how the regions are represented and discovered.

So why not reason about the similarity or distance between objects directly? To do so, we need a basic method for measuring similarity or distance. What does it mean that two companies or two consumers are similar? Let's examine this carefully. Consider two instances from our simplified credit application domain:

Attribute	Person A	Person B
Age	23	40
Years at current address	2	10
Residential status (1=Owner, 2=Renter, 3=Other)	2	1

These data items have multiple attributes, and there's no single best method for reducing them to a single similarity or distance measurement. There are many different ways to measure the similarity or distance between Person A and Person B. A good place to begin is with measurements of distance from basic geometry.

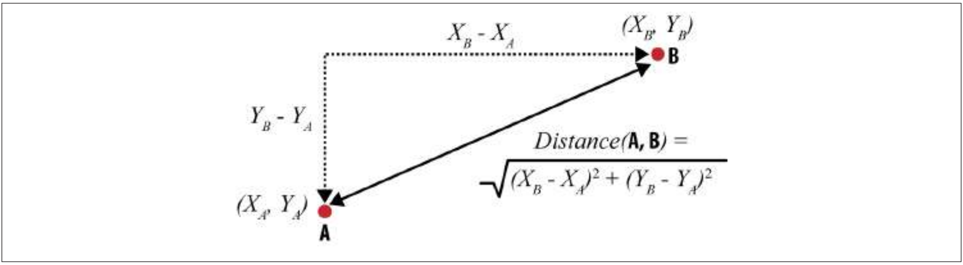


Figure 6-1. Euclidean distance.

Recall from our prior discussions of the geometric interpretation that if we have two (numeric) features, then each object is a point in a two-dimensional space. Figure 6-1 shows two data items, A and B, located on a two-dimensional plane. Object A is at coordinates (x_A, y_A) and B is at (x_B, y_B). At the risk of too much repetition, note that these coordinates are just the values of the two features of the objects. We can draw a right triangle between the two objects, as shown, whose base is the difference in the x's: $(x_A - x_B)$ and whose height is the difference in the y's: $(y_A - y_B)$. The Pythagorean theorem tells us that the distance between A and B is given by the length of the hypotenuse, and is equal to the square root of the summed squares of the lengths of the other two sides of the triangle, which in this case is $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$. Essentially, we can compute the overall distance by computing the distances of the individual dimensions—the individual features in our setting. This is called the *Euclidean distance* [1] between two points, and it's probably the most common geometric distance measure.

1. After Euclid, the 4th century B.C. Greek mathematician known as the Father of Geometry.

Euclidean distance is not limited to two dimensions. If A and B were objects described by three features, they could be represented by points in three-dimensional space and their positions would then be represented as (x_A, y_A, z_A) and (x_B, y_B, z_B). The distance between A and B would then include the term $(z_A - z_B)^2$. We can add arbitrarily many features, each a new dimension. When an object is described by n features, n dimensions $(d_1, d_2, \ldots, d_n)$, the general equation for Euclidean distance in n dimensions is shown in Equation 6-1:

Equation 6-1. General Euclidean distance

$$\sqrt{(d_{1,A} - d_{1,B})^2 + (d_{2,A} - d_{2,B})^2 + \ldots + (d_{n,A} - d_{n,B})^2}$$

We now have a metric for measuring the distance between any two objects described by vectors of numeric features—a simple formula based on the distances of the objects' individual features. Recalling persons A and B, above, their Euclidean distance is:

$$\begin{aligned} d(A, B) &= \sqrt{(23 - 40)^2 + (2 - 10)^2 + (2 - 1)^2} \\ &\approx 18.8 \end{aligned}$$

So the distance between these examples is about 19. This distance is just a number—it has no units, and no meaningful interpretation. It is only really useful for comparing the similarity of one pair of instances to that of another pair. It turns out that comparing similarities is extremely useful.

Nearest-Neighbor Reasoning

Now that we have a way to measure distance, we can use it for many different data-analysis tasks. Recalling examples from the beginning of the chapter, we could use this measure to find the companies most similar to our best corporate customers, or the online consumers most similar to our best retail customers. Once we have found these, we can take whatever action is appropriate in the business context. For corporate customers, IBM does this to help direct its sales force. Online advertisers do this to target ads. These most-similar instances are called *nearest neighbors*.

Example: Whiskey Analytics

Let's talk about a fresh example. One of us (Foster) likes single malt Scotch whiskey. If you've had more than one or two, you realize that there is a lot of variation among the hundreds of different single malts. When Foster finds a single malt he really likes, he wants to find other similar ones—both because he likes to explore the "space" of single malts, but also because any given liquor store or restaurant only has a limited selection.

He wants to be able to pick one he'll really like. For example, the other evening a dining companion recommended trying the single malt "Bunnahabhain."[2] It was unusual and very good. Out of all the many single malts, how could Foster find other ones like that?

Let's take a data science approach. Recall from Chapter 2 that we first should think about the exact question we would like to answer, and what are the appropriate data to answer it. How can we describe single malt Scotch whiskeys as feature vectors, in such a way that we think similar whiskeys will have similar taste? This is exactly the project undertaken by François-Joseph Lapointe and Pierre Legendre of the University of Montréal (Lapointe & Legendre, 1994). They were interested in several classification and organizational questions about Scotch whiskeys. We'll adopt some of their approach here.

It turns out that tasting notes are published for many whiskeys. For example, Michael Jackson is a well-known whiskey and beer connoisseur who has written *Michael Jackson's Malt Whisky Companion: A Connoisseur's Guide to the Malt Whiskies of Scotland* (Jackson, 1989), which describes 109 different single malt Scotches of Scotland. The descriptions are in the form of tasting notes on each Scotch, such as: *"Appetizing aroma of peat smoke, almost incense-like, heather honey with a fruity softness."*

As data scientists, we are making progress. We have found a potentially useful source of data. However, we do not yet have whiskeys described by feature vectors, only by tasting notes. We need to press on with our data formulation. Following Lapointe and Legendre (1994), let's create some numeric features that, for any whiskey, will summarize the information in the tasting notes. Define five general whiskey attributes, each with many possible values:

1.	**Color:** *yellow, very pale, pale, pale gold, gold, old gold, full gold, amber,* etc.	(14 values)
2.	**Nose:** *aromatic, peaty, sweet, light, fresh, dry, grassy,* etc.	(12 values)
3.	**Body:** *soft, medium, full, round, smooth, light, firm, oily.*	(8 values)
4.	**Palate:** *full, dry, sherry, big, fruity, grassy, smoky, salty,* etc.	(15 values)
5.	**Finish:** *full, dry, warm, light, smooth, clean, fruity, grassy, smoky,* etc.	(19 values)

It is important to note that these category values are *not* mutually exclusive (e.g., Aberlour's palate is described as medium, full, soft, round and smooth). In general, any of the values can co-occur (though some of them, like Color being both light and smoky, never do) but because they can co-occur, each value of each variable was coded as a separate feature by Lapointe and Legendre. Consequently there are 68 binary features of each whiskey.

2. No, he can't pronounce it properly either.

Foster likes Bunnahabhain, so we can use Lapointe and Legendre's representation of whiskeys with Euclidean distance to find similar ones for him. For reference, here is their description of Bunnahabhain:

- *Color:* gold
- *Nose:* fresh and sea
- *Body:* firm, medium, and light
- *Palate:* sweet, fruity, and clean
- *Finish:* full

Here is Bunnahabhain's description and the five single-malt Scotches most similar to Bunnahabhain, by increasing distance:

Whiskey	Distance	Descriptors
Bunnahabhain	—	*gold; firm,med,light; sweet,fruit,clean; fresh,sea; full*
Glenglassaugh	0.643	gold; firm,light,smooth; sweet,grass; fresh,grass
Tullibardine	0.647	gold; firm,med,smooth; sweet,fruit,full,grass,clean; sweet; big,arome,sweet
Ardberg	0.667	sherry; firm,med,full,light; sweet; dry,peat,sea;salt
Bruichladdich	0.667	pale; firm,light,smooth; dry,sweet,smoke,clean; light; full
Glenmorangie	0.667	p.gold; med,oily,light; sweet,grass,spice; sweet,spicy,grass,sea,fresh; full,long

Using this list we could find a Scotch similar to Bunnahabhain. At any particular shop we might have to go down the list a bit to find one they stock, but since the Scotches are ordered by similarity we can easily find the most similar Scotch (and also have a vague idea as to how similar the closest available Scotch is as compared to the alternatives that are not available).

This is an example of the direct application of similarity to solve a problem. Once we understand this fundamental notion, we have a powerful conceptual tool for approaching a variety of problems, such as those laid out above (finding similar companies, similar consumers, etc.). As we see in the whiskey example, the data scientist often still has work to do to actually define the data so that the similarity will be with respect to a useful set of characteristics. Later we will present some other notions of similarity and distance. Now, let's move on to another very common use of similarity in data science.

Nearest Neighbors for Predictive Modeling

We also can use the idea of nearest neighbors to do predictive modeling in a different way. Take a minute to recall everything you now know about predictive modeling from prior chapters. To use similarity for predictive modeling, the basic procedure is beautifully simple: given a new example whose target variable we want to predict, we scan through all the training examples and choose several that are the most similar to the

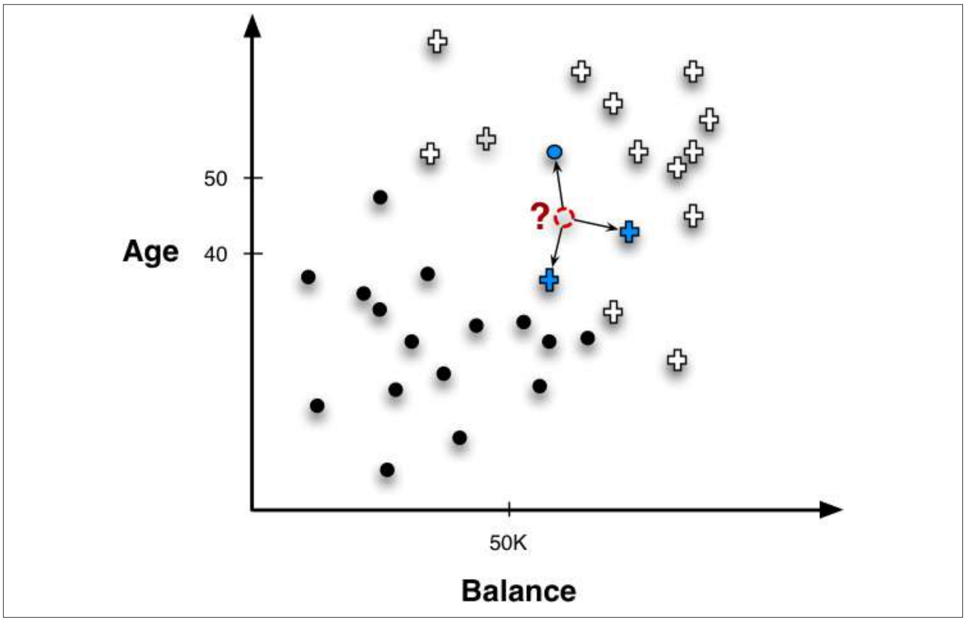


Figure 6-2. Nearest neighbor classification. The point to be classified, labeled with a question mark, would be classified + because the majority of its nearest (three) neighbors are +.

new example. Then we predict the new example's target value, based on the nearest neighbors' (known) target values. How to do that last step needs to be defined; for now, let's just say that we have some *combining function* (like voting or averaging) operating on the neighbors' known target values. The combining function will give us a prediction.

Classification

Since we have focused a great deal on classification tasks so far in the book, let's begin by seeing how neighbors can be used to classify a new instance in a super-simple setting. Figure 6-2 shows a new example whose label we want to predict, indicated by a "?." Following the basic procedure introduced above, the nearest neighbors (in this example, three of them) are retrieved and their known target variables (classes) are consulted. In this this case, two examples are positive and one is negative. What should be our combining function? A simple combining function in this case would be majority vote, so the predicted class would be positive.

Adding just a little more complexity, consider a credit card marketing problem. The goal is to predict whether a new customer will respond to a credit card offer based on how other, similar customers have responded. The data (still oversimplified of course) are shown in Table 6-1.

Table 6-1. Nearest neighbor example: Will David respond or not?

Customer	Age	Income (1000s)	Cards	Response (target)	Distance from David
David	*37*	*50*	*2*	*?*	0
John	35	35	3	Yes	$\sqrt{(35 - 37)^2 + (35 - 50)^2 + (3 - 2)^2} = 15.16$
Rachael	22	50	2	No	$\sqrt{(22 - 37)^2 + (50 - 50)^2 + (2 - 2)^2} = 15$
Ruth	63	200	1	No	$\sqrt{(63 - 37)^2 + (200 - 50)^2 + (1 - 2)^2} = 152.23$
Jefferson	59	170	1	No	$\sqrt{(59 - 37)^2 + (170 - 50)^2 + (1 - 2)^2} = 122$
Norah	25	40	4	Yes	$\sqrt{(25 - 37)^2 + (40 - 50)^2 + (4 - 2)^2} = 15.74$

In this example data, there are five existing customers we previously have contacted with a credit card offer. For each of them we have their name, age, income, the number of cards they already have, and whether they responded to the offer. For a new person, David, we want to predict whether he will respond to the offer or not.

The last column in Table 6-1 shows a distance calculation, using Equation 6-1, of how far each instance is from David. Three customers (John, Rachael, and Norah) are fairly similar to David, with a distance of about 15. The other two customers (Ruth and Jefferson) are much farther away. Therefore, David's three nearest neighbors are Rachael, then John, then Norah. Their responses are No, Yes, and Yes, respectively. If we take a majority vote of these values, we predict Yes (David will respond). This touches upon some important issues with nearest-neighbor methods: how many neighbors should we use? Should they have equal weights in the combining function? We discuss these later in the chapter.

Probability Estimation

We've made the point that it's usually important not just to classify a new example but to estimate its probability—to assign a score to it, because a score gives more information than just a Yes/No decision. Nearest neighbor classification can be used to do this fairly easily. Consider again the classification task of deciding whether David will be a responder or not. His nearest neighbors (Rachael, John, and Norah) have classes of No, Yes, and Yes, respectively. If we score for the Yes class, so that Yes=1 and No=0, we can average these into a score of *2/3* for David. If we were to do this in practice, we might want to use more than just three nearest neighbors to compute the probability estimates (and recall the discussion of estimating probabilities from small samples in "Probability Estimation" on page 71).

Regression

Once we can retrieve nearest neighbors, we can use them for any predictive mining task by combining them in different ways. We just saw how to do classification by taking a majority vote of a target. We can do regression in a similar way.

Assume we had the same dataset as in Table 6-1, but this time we want to predict David's Income. We won't redo the distance calculation, but assume that David's three nearest neighbors were again Rachael, John, and Norah. Their respective incomes are 50, 35, and 40 (in thousands). We then use these values to generate a prediction for David's income. We could use the average (about 42) or the median (40).

It is important to note that in retrieving neighbors we do not use the target variable because we're trying to predict it. Thus Income would not enter into the distance calculation as it does in Table 6-1. However, we're free to use any other variables whose values are available to determine distance.

How Many Neighbors and How Much Influence?

In the course of explaining how classification, regression, and scoring may be done, we have used an example with only three neighbors. Several questions may have occurred to you. First, why *three* neighbors, instead of just one, or five, or one hundred? Second, should we treat all neighbors the same? Though all are called "nearest" neighbors, some are nearer than others, and shouldn't this influence how they're used?

There is no simple answer to how many neighbors should be used. Odd numbers are convenient for breaking ties for majority vote classification with two-class problems. Nearest neighbor algorithms are often referred to by the shorthand k-NN, where the k refers to the number of neighbors used, such as 3-NN.

In general, the greater k is the more the estimates are smoothed out among neighbors. If you have understood everything so far, with a little thought you should realize that if we increase k to the maximum possible (so that $k = n$) the entire dataset would be used for every prediction. Elegantly, this simply predicts the average over the entire dataset for any example. For classification, this would predict the majority class in the entire dataset; for regression, the average of all the target values; for class probability estimation, the "base rate" probability (see Note: Base rate in "Holdout Data and Fitting Graphs" on page 113).

Even if we're confident about the number of neighbor examples we should use, we may realize that neighbors have different similarity to the example we're trying to predict. Shouldn't this influence how they're used?

For classification we started with a simple strategy of *majority* voting, retrieving an odd number of neighbors to break ties. However, this ignores an important piece of information: how close each neighbor is to the instance. For example, consider what would happen if we used $k = 4$ neighbors to classify David. We would retrieve the responses (Yes, No, Yes, No), causing the responses to be evenly mixed. But the first three are very close to David (distance ≈ 15) while the fourth is much further away (distance ≈ 122).

Intuitively, this fourth instance shouldn't contribute as much to the vote as the first three. To incorporate this concern, nearest-neighbor methods often use *weighted voting* or *similarity moderated voting* such that each neighbor's contribution is scaled by its similarity.

Consider again the data in Table 6-1, involving predicting whether David will respond to a credit card offer. We showed that if we predict David's class by majority vote it depends greatly on the number of neighbors we choose. Let's redo the calculations, this time using *all* neighbors but scaling each by its similarity to David, using as the scaling weight the reciprocal of the square of the distance. Here are the neighbors ordered by their distance from David:

Name	Distance	Similarity weight	Contribution	Class
Rachael	15.0	0.004444	0.344	No
John	15.2	0.004348	0.336	Yes
Norah	15.7	0.004032	0.312	Yes
Jefferson	122.0	0.000067	0.005	No
Ruth	152.2	0.000043	0.003	No

The Contribution column is the amount that each neighbor contributes to the final calculation of the target probability prediction (the contributions are proportional to the weights, but adding up to one). We see that distances greatly effect contributions: Rachael, John and Norah are most similar to David and effectively determine our prediction of his response, while Jefferson and Ruth are so far away that they contribute virtually nothing. Summing the contributions for the positive and negative classes, the final probability estimates for David are 0.65 for Yes and 0.35 for No.

This concept generalizes to other sorts of prediction tasks, for example regression and class probability estimation. Generally, we can think of the procedure as weighted *scoring*. Weighted scoring has a nice consequence in that it reduces the importance of deciding how many neighbors to use. Because the contribution of each neighbor is moderated by its distance, the influence of neighbors naturally drops off the farther they are from the instance. Consequently, when using weighted scoring the exact value of k is much less critical than with majority voting or unweighted averaging. Some methods avoiding committing to a k by retrieving a very large number of instances (e.g., all instances, $k = n$) and depend upon distance weighting to moderate the influences.

Sidebar: Many names for nearest-neighbor reasoning

As with many things in data mining, different terms exist for nearest-neighbor classifiers, in part because similar ideas were pursued independently. Nearest-neighbor classifiers were established long ago in statistics and pattern recognition (Cover & Hart, 1967). The idea of classifying new instances directly by consulting a database (a "memory") of instances has been termed *instance-based learning* (Aha, Kibler, & Albert, 1991) and *memory-based learning* (Lin & Vitter, 1994). Because no model is built during "training" and most effort is deferred until instances are retrieved, this general idea is known as *lazy learning* (Aha, 1997).

A related technique in artificial intelligence is *Case-Based Reasoning* (Kolodner, 1993; Aamodt & Plaza, 1994), abbreviated CBR. Past cases are commonly used by doctors and lawyers to reason about new cases, so case-based reasoning has a well-established history in these fields.

However, there are also significant differences between case-based reasoning and nearest-neighbor methods. Cases in CBR are typically not simple feature vector instances but instead are very detailed summaries of an episode, including items such as a patient's symptoms, medical history, diagnosis, treatment, and outcome; or the details of a legal case including plaintiff and defendant arguments, precedents cited, and judgement. Because cases are so detailed, in CBR they are used not just to provide a class label but to provide diagnostic and planning information that can be used to deal with the case after it is retrieved. Adapting historical cases to be used in a new situation is usually a complex process that requires significant effort.

Geometric Interpretation, Overfitting, and Complexity Control

As with other models we've seen, it is instructive to visualize the classification regions created by a nearest-neighbor method. Although no explicit boundary is created, there are implicit regions created by instance neighborhoods. These regions can be calculated by systematically probing points in the instance space, determining each point's classification, and constructing the boundary where classifications change.

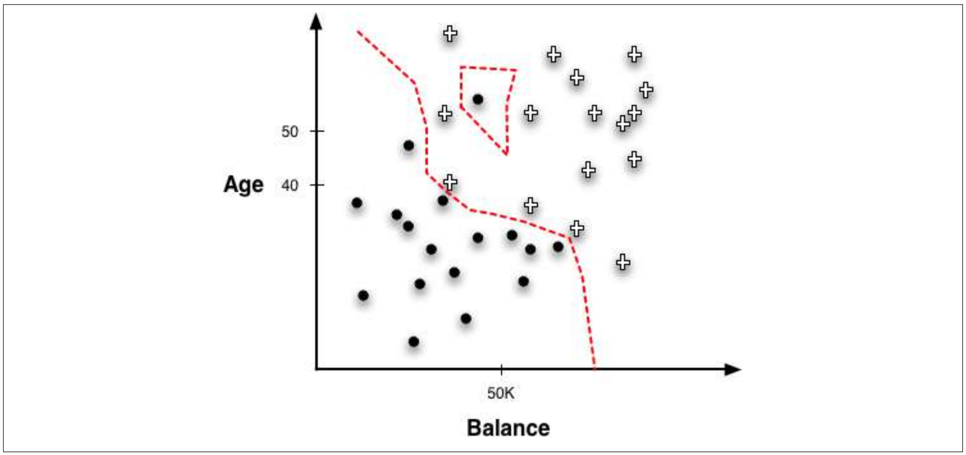


Figure 6-3. Boundaries created by a 1-NN classifier.

Figure 6-3 illustrates such a region created by a 1-NN classifier around the instances of our "Write-off" domain. Compare this with the classification tree regions from Figure 3-15 and the regions created by the linear boundary in Figure 4-3.

Notice that the boundaries are not lines, nor are they even any recognizable geometric shape; they are erratic and follow the frontiers between training instances of different classes. The nearest-neighbor classifier follows very specific boundaries around the training instances. Note also the one negative instance isolated inside the positive instances creates a "negative island" around itself. This point might be considered noise or an outlier, and another model type might smooth over it.

Some of this sensitivity to outliers is due to the use of a 1-NN classifier, which retrieves only single instances, and so has a more erratic boundary than one that averages multiple neighbors. We will return to that in a minute. More generally, irregular concept boundaries are characteristic of all nearest-neighbor classifiers, because they do not impose any particular geometric form on the classifier. Instead, they form boundaries in instance space tailored to the specific data used for training.

This should recall our discussions of overfitting and complexity control from Chapter 5. If you're thinking that 1-NN must overfit very strongly, then you are correct. In fact, think about what would happen if you evaluated a 1-NN classifier on the training data. When classifying each training data point, any reasonable distance metric would lead to the retrieval of that training point itself as its own nearest neighbor! Then its own value for the target variable would be used to predict itself, and voilà, perfect classification. The same goes for regression. The 1-NN memorizes the training data. It does a little better than our strawman lookup table from the beginning of Chapter 5, though. Since the lookup table did not have any notion of similarity, it simply predicted perfectly for exact training examples, and gave some default prediction for all others. The 1-NN classifier predicts perfectly for training examples, but it also can make an often reasonable prediction on other examples: it uses the most similar training example.

Thus, in terms of overfitting and its avoidance, the *k* in a *k*-NN classifier is a complexity parameter. At one extreme, we can set $k = n$ and we do not allow much complexity at all in our model. As described previously, the *n*-NN model (ignoring similarity weighting) simply predicts the average value in the dataset for each case. At the other extreme, we can set $k = 1$, and we will get an extremely complex model, which places complicated boundaries such that every training example will be in a region labeled by its own class.

Now let's return to an earlier question: how should one choose *k*? We can use the same procedure discussed in "A General Method for Avoiding Overfitting" on page 134 for setting other complexity parameters: we can conduct cross-validation or other nested holdout testing on the training set, for a variety of different values of *k*, searching for one that gives the best performance on the training data. Then when we have chosen a value of *k*, we build a *k*-NN model from the entire training set. As discussed in detail in Chapter 5, since this procedure only uses the training data, we can still evaluate it on the test data and get an unbiased estimate of its generalization performance. Data mining tools usually have the ability to do such nested cross-validation to set *k* automatically.

Figure 6-4 and Figure 6-5 show different boundaries created by nearest-neighbor classifiers. Here a simple three-class problem is classified using different numbers of neighbors. In Figure 6-4, only a single neighbor is used, and the boundaries are erratic and very specific to the training examples in the dataset. In Figure 6-5, 30 nearest neighbors are averaged to form a classification. The boundaries are obviously different from Figure 6-4 and are much less jagged. Note, however, that in neither case are the boundaries smooth curves or regular piecewise geometric regions that we would expect to see with a linear model or a tree-structured model. The boundaries for *k*-NN are more strongly defined by the data.

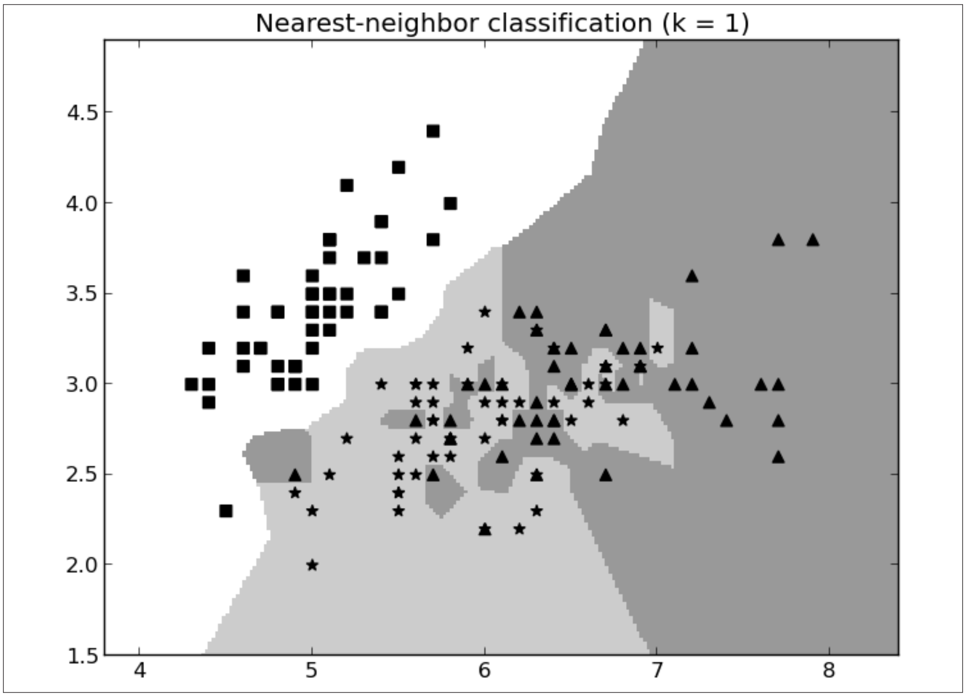


Figure 6-4. Classification boundaries created on a three-class problem created by 1-NN (single nearest neighbor).

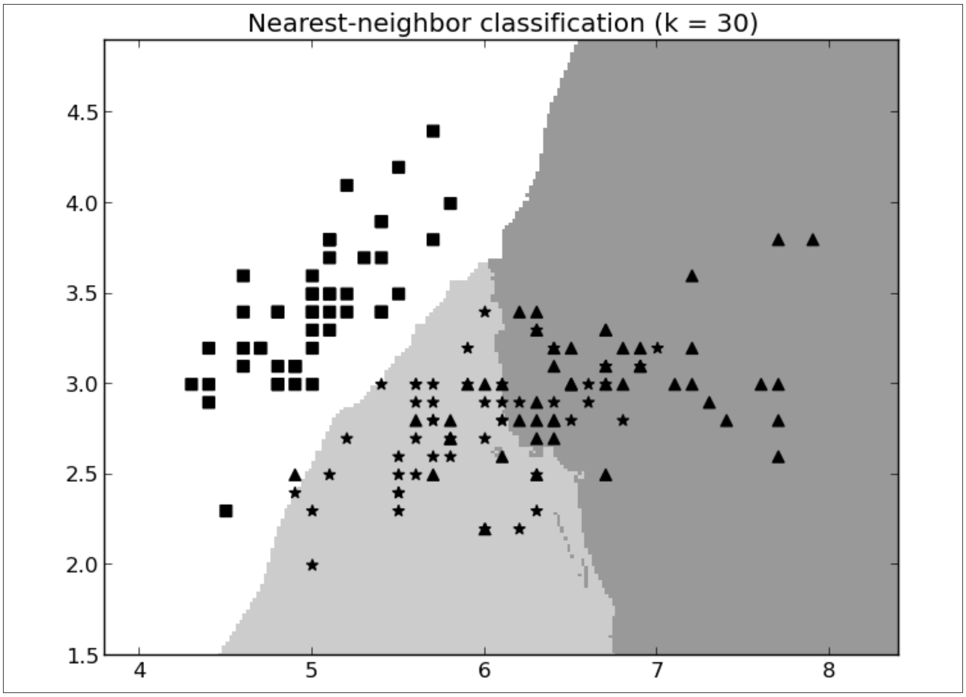


Figure 6-5. Classification boundaries created on a three-class problem created by 30-NN (averaging 30 nearest neighbors).

Issues with Nearest-Neighbor Methods

Before concluding a discussion of nearest-neighbor methods as predictive models, we should mention several issues regarding their use. These often come into play in real-world applications.

Intelligibility

Intelligibility of nearest-neighbor classifiers is a complex issue. As mentioned, in some fields such as medicine and law, reasoning about similar historical cases is a natural way of coming to a decision about a new case. In such fields, a nearest-neighbor method may be a good fit. In other areas, the lack of an explicit, interpretable model may pose a problem.

There are really two aspects to this issue of intelligibility: the justification of a specific *decision* and the intelligibility of an entire *model*.

With *k*-NN, it usually is easy to describe how a single instance is decided: the set of neighbors participating in the decision can be presented, along with their contributions. This was done for the example involving the prediction of whether David would respond, earlier in Table 6-1. Some careful phrasing and judicious presentation of nearest neighbors are useful. For example, Netflix uses a form of nearest-neighbor classification for their recommendations, and explains their movie recommendations with sentences like:

> "The movie *Billy Elliot* was recommended based on your interest in *Amadeus*, *The Constant Gardener* and *Little Miss Sunshine*"

Amazon presents recommendations with phrases like: "Customers with similar searches purchased…" and "Related to Items You've Viewed."

Whether such justifications are adequate depends on the application. An Amazon customer may be satisfied with such an explanation for why she got a recommendation. On the other hand, a mortgage applicant may not be satisfied with the explanation, "We declined your mortgage application because you remind us of the Smiths and the Mitchells, who both defaulted." Indeed, some legal regulations restrict the sorts of models that can be used for credit scoring to models for which very simple explanations can be given based on specific, important variables. For example, with a linear model, one may be able to say: "all else being equal, if your income had been $20,000 higher you would have been granted this particular mortgage."

It also is easy to explain how the entire nearest-neighbor model generally decides new cases. The idea of finding the most similar cases and looking at how they were classified, or what value they had, is intuitive to many.

What is difficult is to explain more deeply what "knowledge" has been mined from the data. If a stakeholder asks "What did your system learn from the data about my customers? On what basis does it make its decisions?" there may be no easy answer because there is no explicit model. Strictly speaking, the nearest-neighbor "model" consists of the entire case set (the database), the distance function, and the combining function. In two dimensions we can visualize this directly as we did in the prior figures. However, this is not possible when there are many dimensions. The knowledge embedded in this model is not usually understandable, so if model intelligibility and justification are critical, nearest-neighbor methods should be avoided.

Dimensionality and domain knowledge

Nearest-neighbor methods typically take into account all features when calculating the distance between two instances. "Heterogeneous Attributes" on page 157 below discusses one of the difficulties with attributes: numeric attributes may have vastly different ranges, and unless they are scaled appropriately the effect of one attribute with a wide range can swamp the effect of another with a much smaller range. But apart from this, there is a problem with having too many attributes, or many that are irrelevant to the similarity judgment.

For example, in the credit card offer domain, a customer database could contain much incidental information such as number of children, length of time at job, house size, median income, make and model of car, average education level, and so on. Conceivably some of these could be relevant to whether the customer would accept the credit card offer, but probably most would be irrelevant. Such problems are said to be high-dimensional—they suffer from the so-called *curse of dimensionality*—and this poses problems for nearest neighbor methods. Much of the reason and effects are quite technical,[3] but roughly, since all of the attributes (dimensions) contribute to the distance calculations, instance similarity can be confused and misled by the presence of too many irrelevant attributes.

There are several ways to fix the problem of many, possibly irrelevant attributes. One is *feature selection*, the judicious determination of features that should be included in the data mining model. Feature selection can be done manually by the data miner, using background knowledge as what attributes are relevant. This is one of the main ways in which a data mining team injects domain knowledge into the data mining process. As discussed in Chapter 3 and Chapter 5 there are also automated feature selection methods that can process the data and make judgments about which attributes give information about the target.

Another way of injecting domain knowledge into similarity calculations is to tune the similarity/distance function manually. We may know, for example, that the attribute *Number of Credit Cards* should have a strong influence on whether a customer accepts an offer for another one. A data scientist can tune the distance function by assigning different weights to the different attributes (e.g., giving a larger weight to *Number of Credit Cards*). Domain knowledge can be added not only because we believe we know what will be more predictive, but more generally because we know something about the similar entities we want to find. When looking for similar whiskeys, I may know that "peatiness" is important to my judging a single malt as tasting similar, so I could give *peaty* a higher weight in the similarity calculation. If another taste variable is unimportant, I could remove it or simply give it a low weight.

Computational efficiency

One benefit of nearest-neighbor methods is that training is very fast because it usually involves only storing the instances. No effort is expended in creating a model. The main computational cost of a nearest neighbor method is borne by the prediction/classification step, when the database must be queried to find nearest neighbors of a new instance. This can be very expensive, and the classification expense should be a consideration. Some applications require extremely fast predictions; for example, in online advertisement targeting, decisions may need to be made in a few tens of milliseconds. For such applications, a nearest neighbor method may be impractical.

3. For example, it turns out that for technical reasons, with large numbers of features, certain particular instances appear extremely frequently in other instances' sets of *k* nearest neighbors. These particular instances thereby have a very large influence on many classifications.

There are techniques for speeding up neighbor retrievals. Specialized data structures like kd-trees and hashing methods (Shakhnarovich, Darrell, & Indyk, 2005; Papadopoulos & Manolopoulos, 2005) are employed in some commerical database and data mining systems to make nearest neighbor queries more efficient. However, be aware that many small-scale and research data mining tools usually do not employ such techniques, and still rely on naive brute-force retrieval.

Some Important Technical Details Relating to Similarities and Neighbors

Heterogeneous Attributes

Up to this point we have been using Euclidean distance, showing that it was easy to calculate. If attributes are numeric and are directly comparable, the distance calculation is indeed straightforward. When examples contain complex, heterogeneous attributes things become more complicated. Consider another example in the same domain but with a few more attributes:

Attribute	Person A	Person B
Sex	Male	Female
Age	23	40
Years at current address	2	10
Residential status (1=Owner, 2=Renter, 3=Other)	2	1
Income	50,000	90,000

Several complications now arise. First, the equation for Euclidean distance is numeric, and Sex is a categorical (symbolic) attribute. It must be encoded numerically. For binary variables, a simple encoding like M=0, F=1 may be sufficient, but if there are multiple values for a categorical attribute this will not be good enough.

Also important, we have variables that, though numeric, have very different scales and ranges. Age might have a range from 18 to 100, while Income might have a range from \$10 to \$10,000,000. Without scaling, our distance metric would consider ten dollars of income difference to be as significant as ten years of age difference, and this is clearly wrong. For this reason nearest-neighbor-based systems often have variable-scaling front ends. They measure the ranges of variables and scale values accordingly, or they apportion values to a fixed number of bins. The general principle at work is that care must be taken that the similarity/distance computation is meaningful for the application.

* Other Distance Functions

For simplicity, up to this point we have used only a single metric, Euclidean distance. Here we include more details about distance functions and some alternatives.

It is important to note that the similarity measures presented here represent only a tiny fraction of all the similarity measures that have been used. These ones are particularly popular, but both the data scientist and the business analyst should keep in mind that it is important to use a meaningful similarity metric with respect to the business problem at hand. This section may be skipped without loss of continuity.

As noted previously, Euclidean distance is probably the most widely used distance metric in data science. It is general, intuitive and computationally very fast. Because it employs the *squares* of the distances along each individual dimension, it is sometimes called the *L2 norm* and sometimes represented by $|| \cdot ||_2$. Equation 6-2 shows how it looks formally.

Equation 6-2. Euclidean distance (L2 norm)

$$d_{\text{Euclidean}}(\mathbf{X}, \mathbf{Y}) = \| \mathbf{X} - \mathbf{Y} \|_2 = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots}$$

Though Euclidean distance is widely used, there are many other distance calculations. The *Dictionary of Distances* by Deza & Deza (Elsevier Science, 2006) lists several hundred, of which maybe a dozen or so are used regularly for mining data. The reason there are so many is that in a nearest-neighbor method the distance function is critical. It basically reduces a comparison of two (potentially complex) examples into a single number. The data types and specifics of the domain of application greatly influence how the differences in individual attributes should combine.

The *Manhattan distance* or *L1-norm* is the sum of the (*unsquared*) pairwise distances, as shown in Equation 6-3.

Equation 6-3. Manhattan distance (L1 norm)

$$d_{\text{Manhattan}}(\mathbf{X}, \mathbf{Y}) = \| \mathbf{X} - \mathbf{Y} \|_1 = | x_1 - y_1 | + | x_2 - y_2 | + \cdots$$

This simply sums the differences along the different dimensions between *X* and *Y*. It is called Manhattan (or taxicab) distance because it represents the total street distance you would have to travel in a place like midtown Manhattan (which is arranged in a grid) to get between two points—the total east-west distance traveled plus the total north-south distance traveled.

Researchers studying the whiskey analytics problem introduced above used another common distance metric.[4] Specifically, they used *Jaccard distance*. Jaccard distance treats the two objects as *sets* of characteristics. Thinking about the objects as sets allows one to think about the size of the union of all the characteristics of two objects X and Y, $|X \cup Y|$, and the size of the set of characteristics shared by the two objects (the intersection), $|X \cap Y|$. Given two objects, X and Y, the Jaccard distance is the proportion of all the characteristics (that either has) that are shared by the two. This is appropriate for problems where the possession of a common characteristic between two items is important, but the common *absence* of a characteristic is not. For example, in finding similar whiskeys it is significant if two whiskeys are both peaty, but it may not be significant that they are both not *salty*. In set notation, the Jaccard distance metric is shown in Equation 6-4.

Equation 6-4. Jaccard distance

$$d_{\text{Jaccard}}(X, Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

Cosine distance is often used in text classification to measure the similarity of two documents. It is defined in Equation 6-5.

Equation 6-5. Cosine distance

$$d_{cosine}(\mathbf{X}, \mathbf{Y}) = 1 - \frac{\mathbf{X} \cdot \mathbf{Y}}{\| \mathbf{X} \|_2 \cdot \| \mathbf{Y} \|_2}$$

where $\|\cdot\|_2$ again represents the L2 norm, or Euclidean length, of each feature vector (for a vector this is simply the distance from the origin).

The information retrieval literature more commonly talks about *cosine similarity*, which is simply the fraction in Equation 6-5. Alternatively, it is 1 – cosine distance.

4. See Lapointe and Legendre (1994), Section 3 ("Classification of Pure Malt Scotch Whiskies"), for a detailed discussion of how they engineered their problem formulation. Available here.

In text classification, each word or token corresponds to a dimension, and the location of a document along each dimension is the number of occurrences of the word in that document. For example, suppose document A contains seven occurrences of the word *performance*, three occurrences of *transition*, and two occurrences of *monetary*. Document B contains two occurrences of *performance*, three occurrences of *transition*, and no occurrences of *monetary*. The two documents would be represented as vectors of counts of these three words: A = <7,3,2> and B = <2,3,0>. The cosine distance of the two documents is:

$$\begin{aligned} d_{\text{cosine}}(A, B) &= 1 - \frac{\langle 7, 3, 2\rangle \cdot \langle 2, 3, 0\rangle}{\| \langle 7, 3, 2\rangle \|_2 \cdot \| \langle 2, 3, 0\rangle \|_2} \\ &= 1 - \frac{7 \cdot 2 + 3 \cdot 3 + 2 \cdot 0}{\sqrt{49 + 9 + 4} \cdot \sqrt{4 + 9}} \\ &= 1 - \frac{23}{28.4} \approx 0.19 \end{aligned}$$

Cosine distance is particularly useful when you want to ignore differences in scale across instances—technically, when you want to ignore the magnitude of the vectors. As a concrete example, in text classification you may want to ignore whether one document is much longer than another, and just concentrate on the textual content. So in our example above, suppose we have a third document, C, which has seventy occurrences of the word *performance*, thirty occurrences of *transition*, and twenty occurrences of *monetary*. The vector representing C would be C = <70, 30, 20>. If you work through the math you'll find that the cosine distance between A and C is zero—because C is simply A multiplied by 10.

As a final example illustrating the variety of distance metrics, let's again consider text but in a very different way. Sometimes you may want to measure the distance between two strings of characters. For example, often a business application needs to be able to judge when two data records correspond to the same person. Of course, there may be misspellings. We would want to be able to say how similar two text fields are. Let's say we have two strings:

1. `1113 Bleaker St.`
2. `113 Bleecker St.`

We want to determine how similar these are. For this purpose, another type of distance function is useful, called *edit distance* or the *Levenshtein metric*. This metric counts the minimum number of edit operations required to convert one string into the other, where an edit operation consists of either inserting, deleting, or replacing a character (one could choose other edit operators). In the case of our two strings, the first could be transformed into the second with this sequence of operations:

1. Delete a `1`,
2. Insert a `c`, and
3. Replace an `a` with an `e`.

So these two strings have an edit distance of three. We might compute a similar edit distance calculation for other fields, such as name (thereby dealing with missing middle initials, for example), and then calculate a higher-level similarity that combines the various edit-distance similarities.

Edit distance is also used commonly in biology where it is applied to measure the genetic distance between strings of alleles. In general, edit distance is a common choice when data items consist of strings or sequences where order is very important.

* Combining Functions: Calculating Scores from Neighbors

For completeness, let us also briefly discuss "combining functions"—the formulas used for calculating the prediction of an instance from a set of the instance's nearest neighbors.

We began with majority voting, a simple strategy. This decision rule can be seen in Equation 6-6:

Equation 6-6. Majority vote classification

$$c(\mathbf{x}) = \underset{c \in \text{classes}}{\arg\max}\ \text{score}(c, \text{neighbors}_k(\mathbf{x}))$$

Here *neighbors*$_k$($\mathbf{x}$) returns the k nearest neighbors of instance $\mathbf{x}$, *arg max* returns the argument (c in this case) that maximizes the quantity that follows it, and the score function is defined, shown in Equation 6-7.

Equation 6-7. Majority scoring function

$$\text{score}(c, N) = \sum_{\mathbf{y} \in N} [\mathit{class}(\mathbf{y}) = c]$$

Here the expression *[class(***y***)=c]* has the value one if $\mathit{class}(\mathbf{y}) = c$ and zero otherwise.

Similarity-moderated voting, discussed in "How Many Neighbors and How Much Influence?" on page 149, can be accomplished by modifying Equation 6-6 to incorporate a weight, as shown in Equation 6-8.

Equation 6-8. Similarity-moderated classification

$$\text{score}(c, N) = \sum_{\mathbf{y} \in N} w(\mathbf{x}, \mathbf{y}) \times [\text{class}(\mathbf{y}) = c]$$

where w is a weighting function based on the similarity between examples $\mathbf{x}$ and $\mathbf{y}$. The inverse of the square of the distance is commonly used:

$$w(\mathbf{x}, \mathbf{y}) = \frac{1}{dist(\mathbf{x}, \mathbf{y})^2}$$

where *dist* is whatever distance function is being used in the domain.

It is straightforward to alter Equation 6-6 and Equation 6-8 to produce a score that can be used as a probability estimate. Equation 6-8 already produces a score so we just have to scale it by the total scores contributed by all neighbors so that it is between zero and one, as shown in Equation 6-9.

Equation 6-9. Similarity-moderated scoring

$$p(c \mid \mathbf{x}) = \frac{\sum_{\mathbf{y} \in \text{neighbors}(\mathbf{x})} w(\mathbf{x}, \mathbf{y}) \times [\text{class}(\mathbf{y}) = c]}{\sum_{\mathbf{y} \in \text{neighbors}(\mathbf{x})} w(\mathbf{x}, \mathbf{y})}$$

Finally, with one more step we can generalize this equation to do regression. Recall that in a regression problem, instead of trying to estimate the class of a new instance x we are trying to estimate some value *f(x)* given the *f* values of the neighbors of $\mathbf{x}$. We can simply replace the bracketed class-specific part of Equation 6-9 with numeric values. This will estimate the regression value as the weighted average of the neighbors' target values (although depending on the application, alternative combining functions might be sensible, such as the median).

Equation 6-10. Similarity-moderated regression

$$f(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in \text{neighbors}(\mathbf{x})} w(\mathbf{x}, \mathbf{y}) \times t(\mathbf{y})}{\sum_{\mathbf{y} \in \text{neighbors}(\mathbf{x})} w(\mathbf{x}, \mathbf{y})}$$

where *t*(**y**) is the target value for example **y**.

So, for example, for estimating the expected spending of a prospective customer with a particular set of characteristics, Equation 6-10 would estimate this amount as the distance-weighted average of the neighbors' historical spending amounts.

Clustering

As noted at the beginning of the chapter, the notions of similarity and distance underpin much of data science. To increase our appreciation of this, let's look at a very different sort of task. Recall the first application of data science that we looked at deeply: supervised segmentation—finding groups of objects that differ with respect to some target characteristic of interest. For example, find groups of customers that differ with respect to their propensity to leave the company when their contracts expire. Why, in talking about supervised segmentation, do we always use the modifier "supervised"?

In other applications we may want to find groups of objects, for example groups of customers, but not driven by some prespecified target characteristic. Do our customers naturally fall into different groups? This may be useful for many reasons. For example, we may want to step back and consider our marketing efforts more broadly. Do we understand who our customers are? Can we develop better products, better marketing campaigns, better sales methods, or better customer service by understanding the natural subgroups? This idea of finding natural groupings in the data may be called unsupervised segmentation, or more simply *clustering*.

Clustering is another application of our fundamental notion of similarity. The basic idea is that we want to find groups of objects (consumers, businesses, whiskeys, etc.), where the objects within groups are similar, but the objects in different groups are not so similar.

Supervised modeling involves discovering patterns to predict the value of a specified target variable, based on data where we know the values of the target variable. Unsupervised modeling does not focus on a target variable. Instead it looks for other sorts of regularities in a set of data.

Example: Whiskey Analytics Revisited

Before getting into details, let's revisit our example problem of whiskey analytics. We discussed using similarity measures to find similar single malt scotch whiskeys. Why might we want to take a step further and find clusters of similar whiskeys?

One reason we might want to find clusters of whiskeys is simply to understand the problem better. This is an example of exploratory data analysis, to which data-rich businesses should continually devote some energy and resources, as such exploration can lead to useful and profitable discoveries. In our example, if we are interested in Scotch whiskeys, we may simply want to understand the natural groupings by taste—because we want to understand our "business," which might lead to a better product or service. Let's say that we run a small shop in a well-to-do neighborhood, and as part of our business strategy we want to be known as the place to go for single-malt scotch whiskeys. We may not be able to have the largest selection, given our limited space and ability to invest in inventory, but we might choose a strategy of having a broad and eclectic collection. If we understood how the single malts grouped by taste, we could (for example) choose from each taste group a popular member and a lesser-known member. Or an expensive member and a more affordable member. Each of these is based on having a good understanding of how the whiskeys group by taste.

Let's now talk about clustering more generally. We will present the two main sorts of clustering, illustrating the concept of similarity in action. In the process, we can examine actual clusters of whiskeys.

Hierarchical Clustering

Let's start with a very simple example. At the top of Figure 6-6 we see six points, A-F, arranged on a plane (i.e., a two-dimensional instance space). Using Euclidean distance renders points more similar to each other if they are closer to each other in the plane. Circles labeled 1-5 are placed over the points to indicate *clusters*. This diagram shows the key aspects of what is called "hierarchical" clustering. It is a *clustering* because it groups the points by their similarity. Notice that the only overlap between clusters is when one cluster contains other clusters. Because of this structure, the circles actually represent a hierarchy of clusterings. The most general (highest-level) clustering is just the single cluster that contains everything—cluster 5 in the example. The lowest-level clustering is when we remove all the circles, and the points themselves are six (trivial) clusters. Removing circles in decreasing order of their numbers in the figure produces a collection of different clusterings, each with a larger number of clusters.

The graph on the bottom of the figure is called a *dendrogram*, and it shows explicitly the hierarchy of the clusters. Along the x axis are arranged (in no particular order except to avoid line crossings) the individual data points. The y axis represents the distance between the clusters (we'll talk more about that presently). At the bottom ($y = 0$) each point is in a separate cluster. As y increases, different groupings of clusters fall within the distance constraint: first A and C are clustered together, then B and E are merged, then the BE cluster is merged with D, and so on, until all clusters are merged at the top. The numbers at the joins of the dendrograms correspond to the numbered circles in the top diagram.

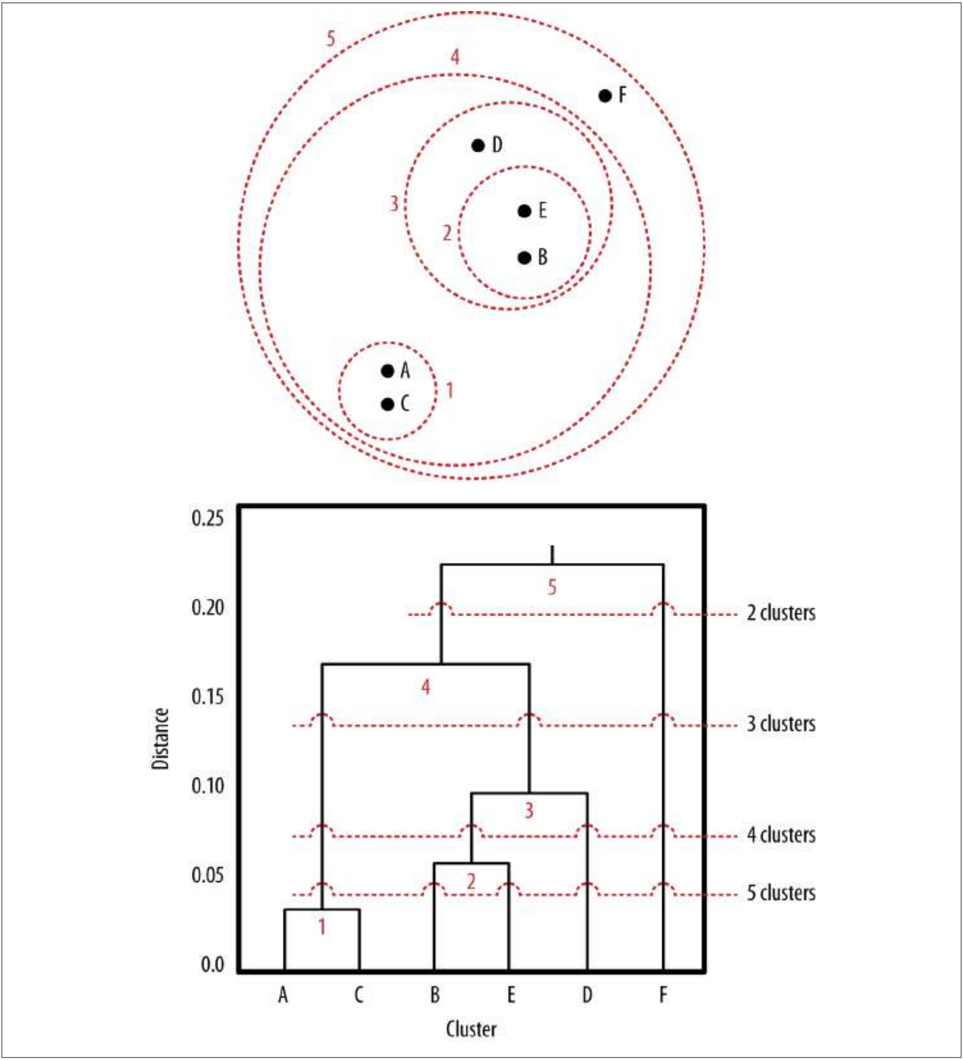


Figure 6-6. Six points and their possible clusterings. At top are shown six points, A-F, with circles 1-5 showing different distance-based groupings that could be imposed. These groups form an implicit hierarchy. At the bottom is a dendrogram corresponding to the groupings, which makes the hierarchy explicit.

Both parts of Figure 6-6 show that hierarchical clustering doesn't just create "a clustering," or a single set of groups of objects. It creates a collection of ways to group the points. To see this clearly, consider "clipping" the dendrogram with a horizontal line, ignoring everything above the line. As the line moves downward, we get different clusterings with increasing numbers of clusters, as shown in the figure. Clip the dendrogram at the line labeled "2 clusters," and below that we see two different groups; here, the singleton point F and the group containing all the other points. Referring back to the top part of the figure, we see that indeed F stands apart from the rest. Clipping the dendrogram at the 2-cluster point corresponds to removing circle 5. If we move down to the horizonal line labeled "3 clusters," and clip the dendrogram there, we see that the dendrogram is left with three groups below the line (AC, BED, F), which corresponds in the plot to removing circles 5 and 4, and we then see the same three clusters. Intuitively, the clusters make sense. F is still off by itself. A and C form a close group. B, E, and D form a close group.

An advantage of hierarchical clustering is that it allows the data analyst to see the groupings—the "landscape" of data similarity—before deciding on the number of clusters to extract. As shown by the horizontal dashed lines, the diagram can be cut across at any point to give any desired number of clusters. Note also that once two clusters are joined at one level, they remain joined in all higher levels of the hierarchy.

Hierarchical clusterings generally are formed by starting with each node as its own cluster. Then clusters are merged iteratively until only a single cluster remains. The clusters are merged based on the similarity or distance function that is chosen. So far we have discussed distance between instances. For hierarchical clustering, we need a distance function between clusters, considering individual instances to be the smallest clusters. This is sometimes called the *linkage* function. So, for example, the linkage function could be "the Euclidean distance between the closest points in each of the clusters," which would apply to any two clusters.

Note: Dendrograms

Two things can usually be noticed in a dendrogram. Because the *y* axis represents the distance between clusters, the dendrogram can give an idea of where natural clusters may occur. Notice in the dendrogram of Figure 6-6 there is a relatively long distance between cluster 3 (at about 0.10) and cluster 4 (at about 0.17). This suggests that this segmentation of the data, yielding three clusters, might be a good division. Also notice point F in the dendrogram. Whenever a single point merges high up in a dendrogram, this is an indication that it seems different from the rest, which we might call an "outlier," and want to investigate it.

One of the best known uses of hierarchical clustering is in the "Tree of Life" (Sugden et al., 2003; Pennisi, 2003), a hierarchical phylogenetic chart of all life on earth. This chart is based on a hierarchical clustering of RNA sequences. A portion of a tree from the Interactive Tree of Life is shown in Figure 6-7 (Letunic & Bork, 2006). Large hierarchical trees are often displayed radially to conserve space, as is done here. This diagram shows a global phylogeny (taxonomy) of fully sequenced genomes, automatically reconstructed by Francesca Ciccarelli and colleagues (2006). The center is the “last universal ancestor” of all life on earth, from which branch the three domains of life (eukaryota, bacteria, and archaea). Figure 6-8 shows a magnified portion of this tree containing the particular bacterium Helicobacter pylori, which causes ulcers.

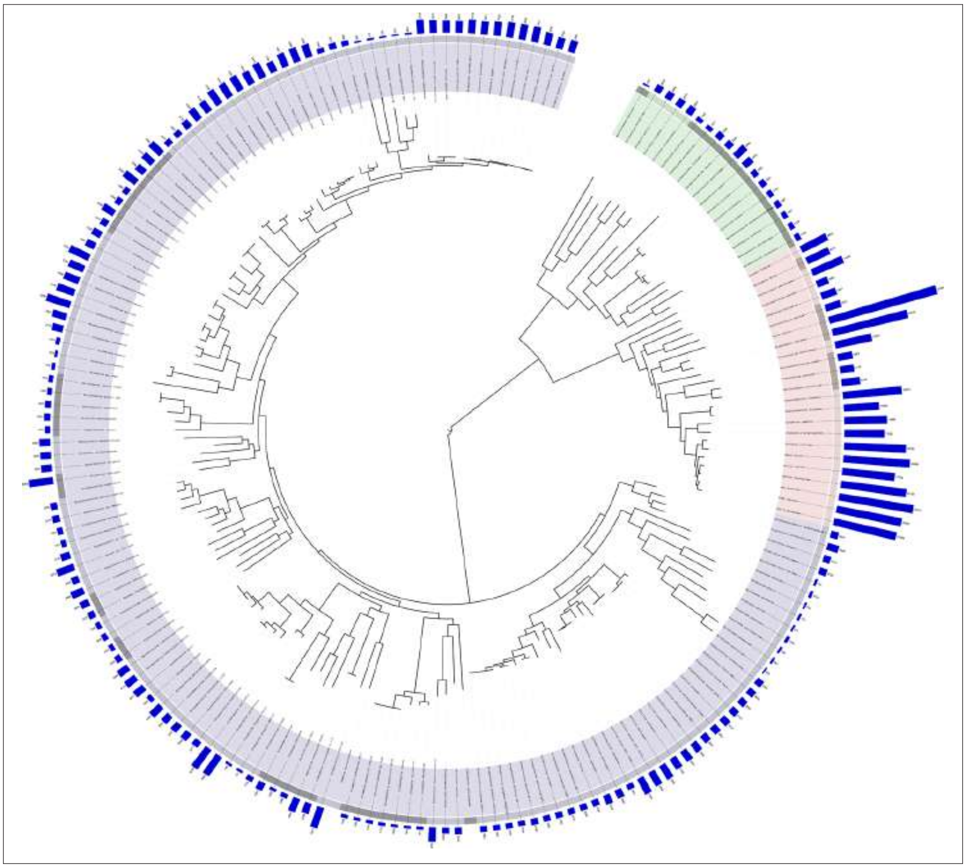


Figure 6-7. The phylogenetic Tree of Life, a huge hierarchical clustering of species, displayed radially.

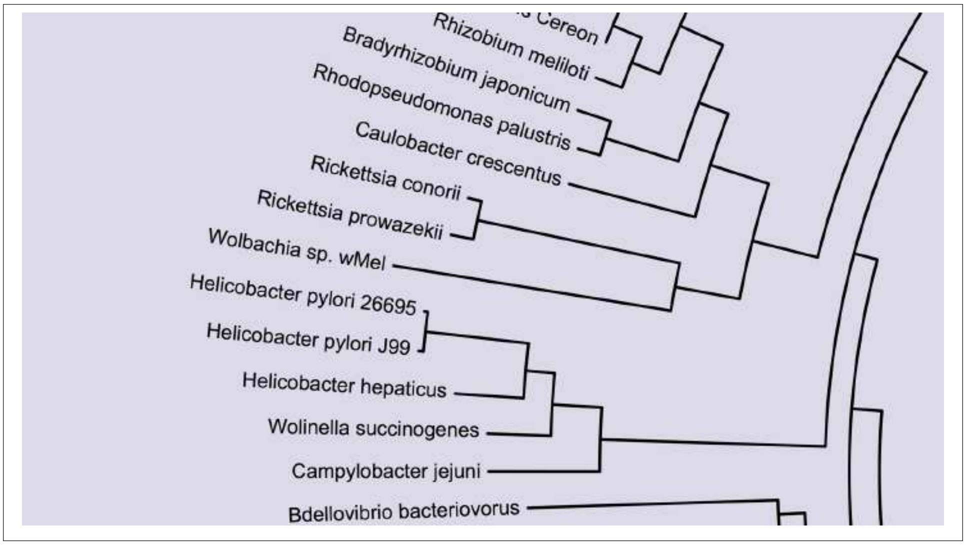


Figure 6-8. A portion of the Tree of Life.

Returning to our example from the outset of the chapter, the top of Figure 6-9 shows, as a dendrogram, the 50 single malt Scotch whiskeys clustered using the methodology described by Lapointe and Legendre (1994). By clipping the dendrogram we can obtain any number of clusters we would like, so for example, removing the top-most 11 connecting segments leaves us with 12 clusters.

At the bottom of Figure 6-9 is a close up of a portion of the hierarchy, focusing on Foster's new favorite, Bunnahabhain. Previously in "Example: Whiskey Analytics" on page 144 we retrieved whiskeys similar to it. This excerpt shows that most of its nearest neighbors (Tullibardine, Glenglassaugh, etc.) do indeed cluster near it in the hierarchy. (You may wonder why the clusters don't correspond *exactly* to the similarity ranking. The reason is that, while the five whiskeys we found are the most similar to Bunnahabhain, some of these five are more similar to other whiskeys in the dataset, so they are clustered with these closer neighbors before joining Bunnahabhain.)

Interestingly from the point of view of whiskey classification, the groups of single malts resulting from this taste-based clustering do not correspond neatly with regions of Scotland—the basis of the usual categorizations of Scotch whiskeys. There is a correlation, however, as Lapointe and Legendre (1994) point out.

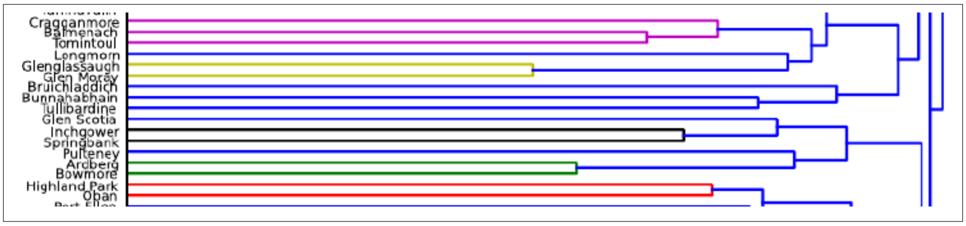


Figure 6-9. Hierarchical clustering of Scotch whiskeys. At top is the entire dendrogram; at bottom is a small excerpt including Bunnahabhain and neighbors.

So instead of simply stocking the most recognizable Scotches, or a few Highland, Lowland, and Islay brands, our specialty shop owner could choose to stock single malts from the different clusters. Alternatively, one could create a guide to Scotch whiskeys that might help single malt lovers to choose whiskeys.[5] For example, since Foster loves the Bunnahabhain recommended to him by his friend at the restaurant the other night, the clustering suggests a set of other "most similar" whiskeys (Bruichladdich, Tullibardine, etc.) The most unusual tasting single malt in the data appears to be Aultmore, at the very top, which is the last whiskey to join any others.

Nearest Neighbors Revisited: Clustering Around Centroids

Hierarchical clustering focuses on the similarities between the individual instances and how similarities link them together. A different way of thinking about clustering data is to focus on the clusters themselves—the groups of instances. The most common method for focusing on the clusters themselves is to represent each cluster by its "cluster center," or *centroid*. Figure 6-10 illustrates the idea in two dimensions: here we have three clusters, whose instances are represented by the circles. Each cluster has a centroid, represented by the solid-lined star. The star is not necessarily one of the instances; it is the geometric center of a group of instances. This same idea applies to any number of dimensions, as long as we have a numeric instance space and a distance measure (of course, we can't visualize the clusters so nicely, if at all, in high-dimensional space).

The most popular centroid-based clustering algorithm is called *k-means* clustering (MacQueen, 1967; Lloyd, 1982; MacKay, 2003), and the main idea behind it deserves some discussion as *k*-means clustering is mentioned frequently in data science. In *k*-means the "means" are the centroids, represented by the arithmetic means (averages) of the values along each dimension for the instances in the cluster. So in Figure 6-10, to compute the centroid for each cluster, we would average all the *x* values of the points in the cluster to form the *x* coordinate of the centroid, and average all the *y* values to form

5. This has been done: see David Wishart's (2006) book *Whisky Classified: Choosing Single Malts by Flavour Pavilion*.

the centroid's *y* coordinate. Generally, the centroid is the average of the values for each feature of each example in the cluster. The result is shown in Figure 6-11.

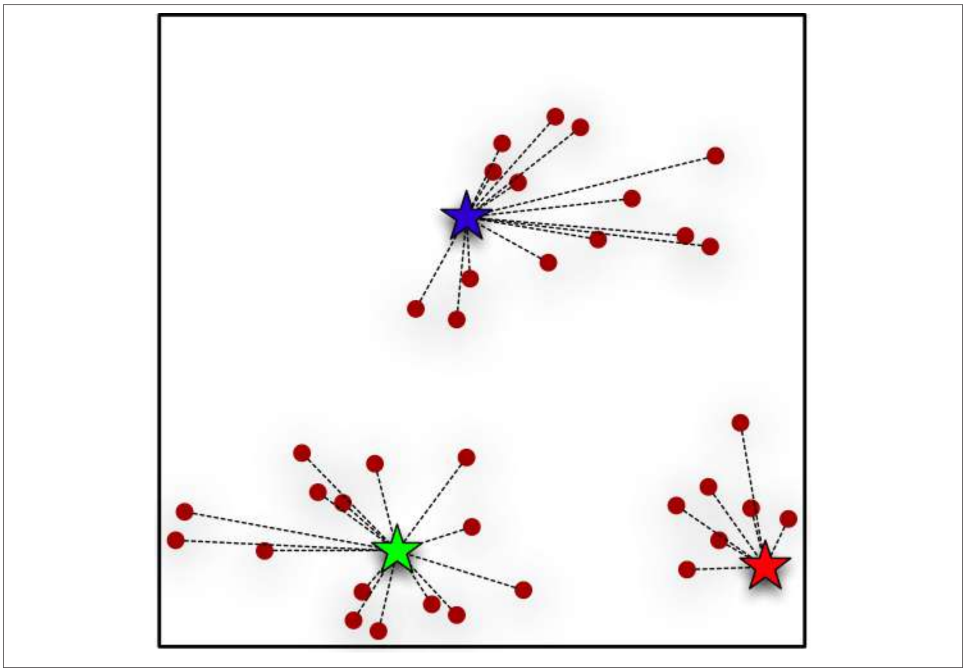


Figure 6-10. The first step of the k-means algorithm: find the points closest to the chosen centers (possibly chosen randomly). This results in the first set of clusters.

The *k* in *k*-means is simply the number of clusters that one would like to find in the data. Unlike hierarchical clustering, *k*-means starts with a desired number of clusters *k*. So, in Figure 6-10, the analyst would have specified *k*=3, and the *k*-means clustering method would return (i) the three cluster centroids when cluster method terminates (the three solid-lined stars in Figure 6-11), plus (ii) information on which of the data points belongs to each cluster. This is sometimes referred to as nearest-neighbor clustering because the answer to (ii) is simply that each cluster contains those points that are nearest to its centroid (rather than to one of the other centroids).

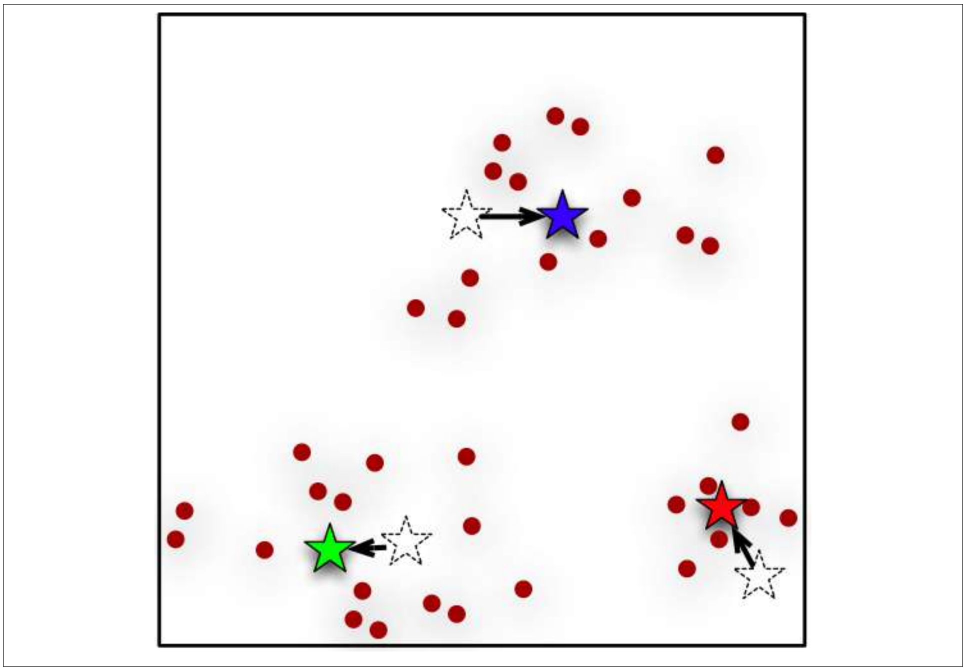


Figure 6-11. The second step of the k-means algorithm: find the actual center of the clusters found in the first step.

The *k*-means algorithm for finding the clusters is simple and elegant, and therefore is worth mentioning. It is represented by Figure 6-10 and Figure 6-11. The algorithm starts by creating *k* initial cluster centers, usually randomly, but sometimes by choosing *k* of the actual data points, or by being given specific initial starting points by the user, or via a pre-processing of the data to determine a good set of starting centers (Arthur & Vassilvitskii, 2007). Think of the stars in Figure 6-10 as being these initial (*k*=3) cluster centers. Then the algorithm proceeds as follows. As shown in Figure 6-10, the clusters corresponding to these cluster centers are formed, by determining which is the closest center to each point.

Next, for each of these clusters, its center is recalculated by finding the actual centroid of the points in the cluster. As shown in Figure 6-11, the cluster centers typically shift; in the figure, we see that the new solid-lined stars are indeed closer to what intuitively seems to be the center of each cluster. And that's pretty much it. The process simply iterates: since the cluster centers have shifted, we need to recalculate which points belong to each cluster (as in Figure 6-10). Once these are reassigned, we might have to shift the cluster centers again. The *k*-means procedure keeps iterating until there is no change in the clusters (or possibly until some other stopping criterion is met).

Figure 6-12 and Figure 6-13 show an example run of *k*-means on 90 data points with *k*=3. This dataset is a little more realistic in that it does not have such well-defined clusters as in the previous example. Figure 6-12 shows the initial data points before clustering. Figure 6-13 shows the final results of clustering after 16 iterations. The three (erratic) lines show the path from each centroid's initial (random) location to its final location. The points in the three clusters are denoted by different symbols (circles, x's, and triangles).

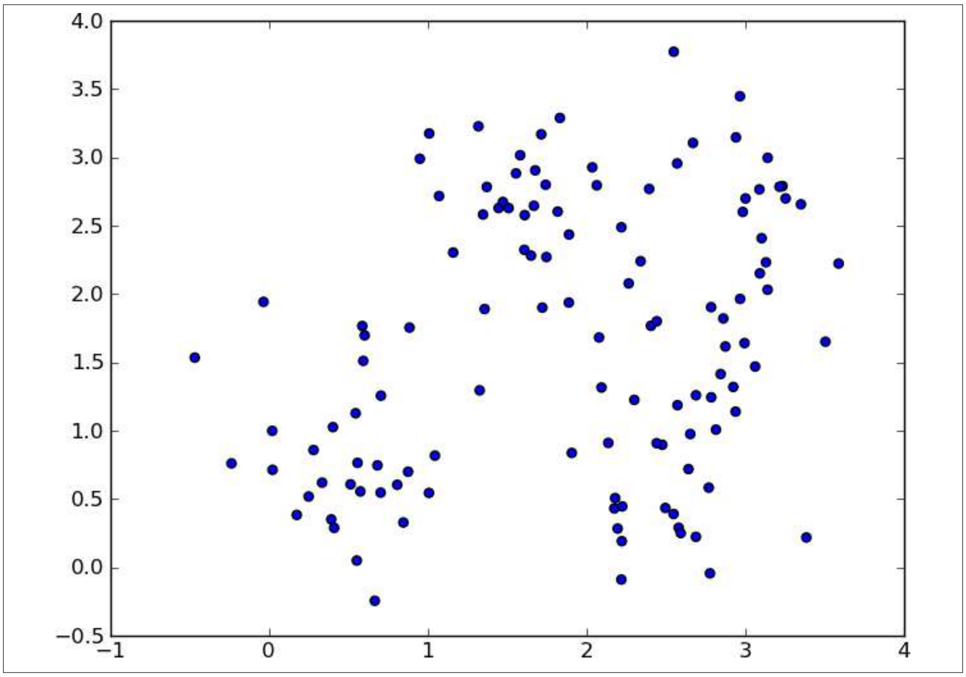


Figure 6-12. A k-means clustering example using 90 points on a plane and k=3 centroids. This figure shows the initial set of points.

There is no guarantee that a single run of the *k*-means algorithm will result in a good clustering. The result of a single clustering run will find a local optimum—a locally best clustering—but this will be dependent upon the initial centroid locations. For this reason, *k*-means is usually run many times, starting with different random centroids each time. The results can be compared by examining the clusters (more on that in a minute), or by a numeric measure such as the clusters' *distortion*, which is the sum of the squared differences between each data point and its corresponding centroid. In the latter case, the clustering with the lowest distortion value can be deemed the best clustering.

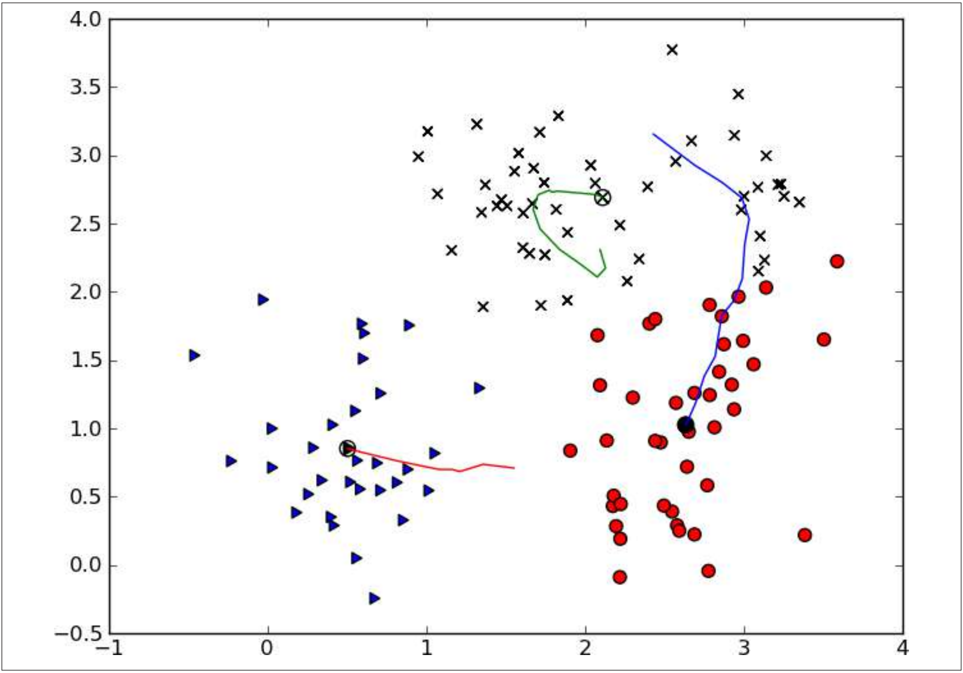


Figure 6-13. A k-means clustering example using 90 points on a plane and k=3 centroids. This figure shows the movement paths of centroids (each of three lines) through 16 iterations of the clustering algorithm. The marker shape of each point represents the cluster identity to which it is finally assigned.

In terms of run time, the *k*-means algorithm is efficient. Even with multiple runs it is generally relatively fast, because it only computes the distances between each data point and the cluster centers on each interation. Hierarchical clustering is generally slower, as it needs to know the distances between all pairs of clusters on each iteration, which at the start is all pairs of data points.

A common concern with centroid algorithms such as *k*-means is how to determine a good value for *k*. One answer is simply to experiment with different *k* values and see which ones generate good results. Since *k*-means is often used for exploratory data mining, the analyst must examine the clustering results anyway to determine whether the clusters make sense. Usually this can reveal whether the number of clusters is appropriate. The value for *k* can be decreased if some clusters are too small and overly specific, and increased if some clusters are too broad and diffuse.

For a more objective measure, the analyst can experiment with increasing values of *k* and graph various metrics (sometimes obliquely called *indices*) of the quality of the resulting clusterings. As *k* increases the quality metrics should eventually stabilize or plateau, either bottoming out if the metric is to be minimized or topping out if maximized. Some judgment will be required, but the minimum *k* where the stabilization begins is often a good choice. Wikipedia's article *Determining the number of clusters in a data set* describes various metrics for evaluating sets of candidate clusters.

Example: Clustering Business News Stories

As a concrete example of centroid-based clustering, consider the task of identifying some natural groupings of business news stories released by a news aggregator. The objective of this example is to identify, informally, different groupings of news stories released about a particular company. This may be useful for a specific application, for example: to get a quick understanding of the news about a company without having to read every news story; to categorize forthcoming news stories for a news prioritization process; or simply to understand the data before undertaking a more focused data mining project, such as relating business news stories to stock performance.

For this example we chose a large collection of (text) news stories: the Thomson Reuters Text Research Collection (TRC2), a corpus of news stories created by the Reuters news agency, and made available to researchers. The entire corpus comprises 1,800,370 news stories from January of 2008 through February of 2009 (14 months). To make the example tractable but still realistic, we're going to extract only those stories that mention a particular company—in this case, Apple (whose stock symbol is AAPL).

Data preparation

For this example, it is useful to discuss data preparation in a little detail, as we will be treating text as data, and we have not previously discussed that. See Chapter 10 for more details on mining text.

In this corpus, large companies are always mentioned when they are the primary subject of a story, such as in earnings reports and merger announcements; but they are often mentioned peripherally in weekly business summaries, lists of active stocks, and stories mentioning significant events within their industry sectors. For example, many stories about the personal computer industry mention how HP's and Dell's stock prices reacted on that day even if neither company was involved in the event. For this reason, we extracted stories whose headlines specifically mentioned Apple—thus assuring that the story is very likely news about Apple itself. There were 312 such stories but they covered a wide variety of topics, as we shall see.

Prior to clustering, the stories underwent basic web text preprocessing, with HTML and URLs stripped out and the text case-normalized. Words that occurred rarely (fewer than two documents) or too commonly (more than 50% documents) in the corpus were eliminated, and the rest formed the *vocabulary* for the next step. Then each document was represented by a numeric feature vector using "TFIDF scores" scoring for each vocabulary word in the document. TFIDF (Term Frequency times Inverse Document

Frequency) scores represent the frequency of the word in the document, penalized by the frequency of the word in the corpus. TFIDF is explained in detail later in Chapter 10.

The news story clusters

We chose to cluster the stories into nine groups (so *k*=9 for *k*-means). Here we present a description of the clusters, along with some headlines of the stories contained in that cluster. It is important to remember that the entire news story was used in the clustering, not just the headline.

Cluster 1. These stories are analysts' announcements concerning ratings changes and price target adjustments:

- `RBC RAISES APPLE <AAPL.O> PRICE TARGET TO $200 FROM $190; KEEPS OUT PERFORM RATING`
- `THINKPANMURE ASSUMES APPLE <AAPL.O> WITH BUY RATING; $225 PRICE TARGET`
- `AMERICAN TECHNOLOGY RAISES APPLE <AAPL.O> TO BUY FROM NEUTRAL`
- `CARIS RAISES APPLE <AAPL.O> PRICE TARGET TO $200 FROM $170; RATING ABOVE AVERAGE`
- `CARIS CUTS APPLE <AAPL.O> PRICE TARGET TO $155 FROM $165; KEEPS ABOVE AVERAGE RATING`

Cluster 2. This cluster contains stories about Apple's stock price movements, during and after each day of trading:

- `Apple shares pare losses, still down 5 pct`
- `Apple rises 5 pct following strong results`
- `Apple shares rise on optimism over iPhone demand`
- `Apple shares decline ahead of Tuesday event`
- `Apple shares surge, investors like valuation`

Cluster 3. In 2008, there were many stories about Steve Jobs, Apple's charismatic CEO, and his struggle with pancreatic cancer. Jobs' declining health was a topic of frequent discussion, and many business stories speculated on how well Apple would continue without him. Such stories clustered here:

- `ANALYSIS-Apple success linked to more than just Steve Jobs`
- `NEWSMAKER-Jobs used bravado, charisma as public face of Apple`
- `COLUMN-What Apple loses without Steve: Eric Auchard`
- `Apple could face lawsuits over Jobs' health`

- `INSTANT VIEW 1-Apple CEO Jobs to take medical leave`
- `ANALYSIS-Investors fear Jobs-less Apple`

Cluster 4. This cluster contains various Apple announcements and releases. Superficially, these stories were similar, though the specific topics varied:

- `Apple introduces iPhone "push" e-mail software`
- `Apple CFO sees 2nd-qtr margin of about 32 pct`
- `Apple says confident in 2008 iPhone sales goal`
- `Apple CFO expects flat gross margin in 3rd-quarter`
- `Apple to talk iPhone software plans on March 6`

Cluster 5. This cluster's stories were about the iPhone and deals to sell iPhones in other countries:

- `MegaFon says to sell Apple iPhone in Russia`
- `Thai True Move in deal with Apple to sell 3G iPhone`
- `Russian retailers to start Apple iPhone sales Oct 3`
- `Thai AIS in talks with Apple on iPhone launch`
- `Softbank says to sell Apple's iPhone in Japan`

Cluster 6. One class of stories reports on stock price movements outside of normal trading hours (known as Before and After the Bell):

- `Before the Bell-Apple inches up on broker action`
- `Before the Bell-Apple shares up 1.6 pct before the bell`
- `BEFORE THE BELL-Apple slides on broker downgrades`
- `After the Bell-Apple shares slip`
- `After the Bell-Apple shares extend decline`

Centroid 7. This cluster contained little thematic consistency:

- `ANALYSIS-Less cheer as Apple confronts an uncertain 2009`
- `TAKE A LOOK - Apple Macworld Convention`
- `TAKE A LOOK-Apple Macworld Convention`
- `Apple eyed for slim laptop, online film rentals`
- `Apple's Jobs finishes speech announcing movie plan`

Cluster 8. Stories on iTunes and Apple's position in digital music sales formed this cluster:

- `PluggedIn-Nokia enters digital music battle with Apple`
- `Apple's iTunes grows to No. 2 U.S. music retailer`
- `Apple may be chilling iTunes competition`
- `Nokia to take on Apple in music, touch-screen phones`
- `Apple talking to labels about unlimited music`

Cluster 9. A particular kind of Reuters news story is a News Brief, which is usually just a few itemized lines of very terse text (e.g. "`• Says purchase new movies on itunes same day as dvd release`"). The contents of these New Briefs varied, but because of their very similar form they clustered together:

- `BRIEF-Apple releases Safari 3.1`
- `BRIEF-Apple introduces ilife 2009`
- `BRIEF-Apple announces iPhone 2.0 software beta`
- `BRIEF-Apple to offer movies on iTunes same day as DVD release`
- `BRIEF-Apple says sold one million iPhone 3G's in first weekend`

As we can see, some of these clusters are interesting and thematically consistent while others are not. Some are just collections of superficially similar text. There is an old cliché in statistics: *Correlation is not causation*, meaning that just because two things co-occur doesn't mean that one causes another. A similar caveat in clustering could be: *Syntactic similarity is not semantic similarity*. Just because two things—particularly text passages—have common surface characteristics doesn't mean they're necessarily related semantically. We shouldn't expect every cluster to be meaningful and interesting. Nevertheless, clustering is often a useful tool to uncover structure in our data that we did not foresee. Clusters can suggest new and interesting data mining opportunities.

Understanding the Results of Clustering

Once we have formulated the instances and clustered them, then what? As we mentioned above, the result of clustering is either a dendrogram or a set of cluster centers plus the corresponding data points for each cluster. How can we understand the clustering? This is particularly important because clustering often is used in exploratory analysis, so the whole point is to understand whether something was discovered, and if so, what?

How to understand clusterings and clusters depends on the sort of data being clustered and the domain of application, but there are several methods that apply broadly. We have seen some of them in action already.

Consider our whiskey example. Our whiskey researchers Lapointe and Legendre cut their dendrogram into 12 clusters; here are two of them:

Group A

Scotches: Aberfeldy, Glenugie, Laphroaig, Scapa

Group H

Scotches: Bruichladdich, Deanston, Fettercairn, Glenfiddich, Glen Mhor, Glen Spey, Glentauchers, Ladyburn, Tobermory

Thus, to examine the clusters, we simply can look at the whiskeys in each cluster. That seems rather easy, but remember that this whiskey example was chosen as an illustration in a book. What is it about the application that allowed relatively easy examination of the clusters (and thereby made it a good example in the book)? We might think, well, there are only a small number of whiskeys in total; that allows us to actually look at them all. This is true, but it actually is not so critical. If we had had massive numbers of whiskeys, we still could have sampled whiskeys from each cluster to show the composition of each.

The more important factor to understanding these clusters—at least for someone who knows a little about single malts—is that the elements of the cluster can be represented by the *names* of the whiskeys. In this case, the names of the data points are meaningful in and of themselves, and convey meaning to an expert in the field.

This gives us a guideline that can be applied to other applications. For example, if we are clustering customers of a large retailer, probably a list of the names of the customers in a cluster would have little meaning, so this technique for understanding the result of clustering would not be useful. On the other hand, if IBM is clustering business customers it may be that the names of the businesses (or at least many of them) carry considerable meaning to a manager or member of the sales force.

What can we do in cases where we cannot simply show the names of our data points, or for which showing the names does not give sufficient understanding? Let's look again at our whiskey clusters, but this time looking at more information on the clusters:

Group A

- Scotches: Aberfeldy, Glenugie, Laphroaig, Scapa
- The best of its class: Laphroaig (Islay), 10 years, 86 points
- Average characteristics: full gold; fruity, salty; medium; oily, salty, sherry; dry

Group H

- Scotches: Bruichladdich, Deanston, Fettercairn, Glenfiddich, Glen Mhor, Glen Spey, Glentauchers, Ladyburn, Tobermory
- The best of its class: Bruichladdich (Islay), 10 years, 76 points

- Average characteristics: white wyne, pale; sweet; smooth, light; sweet, dry, fruity, smoky; dry, light

Here we see two additional pieces of information useful for understanding the results of clustering. First, in addition to listing out the members, an "exemplar" member is listed. Here it is the "best of its class" whiskey, taken from Jackson (1989) (this additional information was not provided to the clustering algorithm). Alternatively, it could be the best known or highest-selling whiskey in the cluster. These techniques could be especially useful when there are massive numbers of instances in each cluster, so randomly sampling some may not be as telling as carefully selecting exemplars. However, this still presumes that the names of the instances are meaningful. Our other example, clustering the business news stories, shows a slight twist on this general idea: show exemplar stories and their headlines, because there the headlines can be meaningful summaries of the stories.

The example also illustrates a different way of understanding the result of the clustering: it shows the average characteristics of the members of the cluster—essentially, it shows the cluster centroid. Showing the centroid can be applied to any clustering; whether it is meaningful depends on whether the data values themselves are meaningful.

* Using Supervised Learning to Generate Cluster Descriptions

This section describes a way to automatically generate cluster descriptions. It is more complicated than the ones already discussed. It involves mixing unsupervised learning (the clustering) with supervised learning in order to create differential descriptions of the clusters. If this chapter is your first introduction to clustering and unsupervised learning, this may seem confusing to you, so we've made it a starred (advanced material) chapter. It may be skipped without loss of continuity.

However clustering was done, it provides us with a list of assignments indicating which examples belong to which cluster. A cluster centroid, in effect, describes the average cluster member. The problem is that these descriptions may be very detailed and they don't tell us how the clusters differ. What we may want to know is, for each cluster, *what differentiates this cluster from all the others?* This is essentially what supervised learning methods do so we can use them here.

The general strategy is this: we use the cluster assignments to label examples. Each example will be given a label of the cluster it belongs to, and these can be treated as class labels. Once we have a labeled set of examples, we run a supervised learning algorithm on the example set to generate a classifier for each class/cluster. We can then inspect the classifier descriptions to get a (hopefully) intelligible and concise description of the corresponding cluster. The important thing to note is that these will be *differential* descriptions: for each cluster, what differentiates it from the others?

In this section, from this point on we equate clusters with classes. We will use the terms interchangeably.

In principle we could use any predictive (supervised) learning method for this, but what is important here is *intelligibility*: we're going to use the learned classifier definition as a cluster description so we want a model that will serve this purpose. "Trees as Sets of Rules" on page 71 showed how rules could be extracted from classification trees, so this is a useful method for the task.

There are two ways to set up the classification task. We have *k* clusters so we could set up a *k*-class task (one class per cluster). Alternatively, we could set up a *k* separate learning tasks, each trying to differentiate one cluster from all the other *(k–1)* clusters.

We'll use the second approach on the whiskey-clustering task, using Lapointe and Legendre's cluster assignments (Appendix A of *A Classification of Pure Malt Scotch Whiskies*). This gives us 12 whiskey clusters labeled A through L. We go back to our raw data and append each whiskey description with its cluster assignment. We're going to use the binary approach: choose each cluster in turn to classify against the others. We'll choose cluster J, which Lapointe and Legendre describe this way:

Group J

- Scotches: Glen Albyn, Glengoyne, Glen Grant, Glenlossie, Linkwood, North Port, Saint Magdalene, Tamdhu
- The best of its class: Linkwood (Speyside), 12 years, 83
- Average characteristics: full gold; dry, peaty, sherry; light to medium, round; sweet; dry

You may recall from "Example: Whiskey Analytics" on page 144 that each whiskey is described using 68 binary features. The dataset now has a label (**J** or **not_J**) for each whiskey indicating whether it belongs to the J cluster. An excerpt of the dataset looks like this:

```
0,0,0,...,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,J      % Glen Grant
0,0,0,...,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,not_J  % Glen Keith
0,0,0,...,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,not_J  % Glen Mhor
```

The text after the "%" is a comment indicating the name of the whiskey.

This dataset is passed to a classification tree learner.[6] The result is shown in Figure 6-14.

6. Specifically, the J48 procedure of Weka with pruning turned off.