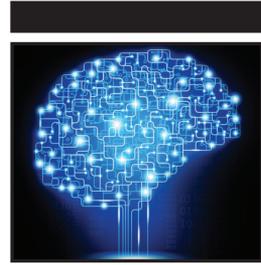# Machine Learning

HANDS-ON FOR DEVELOPERS AND TECHNICAL PROFESSIONALS

Jason Bell

WILEY

Machine Learning

Hands-On for Developers and
Technical Professionals

Jason Bell

WILEY

To Wendy and Clarissa.

Credits

Executive Editor
Carol Long

Project Editor
Charlotte Kughen

Technical Editor
Mitchell Wyle

Production Editor
Christine Mugnolo

Copy Editor
Katherine Burt

Production Manager
Kathleen Wisor

Manager of Content Development and Assembly
Mary Beth Wakefield

Director of Community Marketing
David Mayhew

Marketing Manager
Carrie Sherrill

Business Manager
Amy Knies

Professional Technology & Strategy Director
Barry Pruett

Associate Publisher
Jim Minatel

Project Coordinator, Cover
Patrick Redmond

Proofreader
Nancy Carrasco

Indexer
Johnna Dinse

Cover Designer
Wiley

Cover Image
© iStock.com/VLADGRIN

About the Author

Jason Bell has been working with point-of-sale and customer-loyalty data since 2002, and he has been involved in software development for more than 25 years. He is founder of Datasentiment, a UK business that helps companies worldwide with data acquisition, processing, and insight.

Acknowledgments

During the autumn of 2013, I was presented with some interesting options: either do a research-based PhD or co-author a book on machine learning. One would take six years and the other would take seven to eight months. Because of the speed the data industry was, and still is, progressing, the idea of the book was more appealing because I would be able to get something out while it was still fresh and relevant, and that was more important to me.

I say "co-author" because the original plan was to write a machine learning book with Aidan Rogers. Due to circumstances beyond his control he had to pull out. With Aidan's blessing, I continued under my own steam, and for that opportunity I can't thank him enough for his grace, encouragement, and support in that decision.

Many thanks goes to Wiley, especially Executive Editor, Carol Long, for letting me tweak things here and there with the original concept and bring it to a more practical level than a theoretical one; Project Editor, Charlotte Kughen, who kept me on the straight and narrow when there were times I didn't make sense; and Mitchell Wyle for reviewing the technical side of things. Also big thanks to the Wiley family as a whole for looking after me with this project.

Over the years I've met and worked with some incredible people, so in no particular order here goes: Garrett Murphy, Clare Conway, Colin Mitchell, David Crozier, Edd Dumbill, Matt Biddulph, Jim Weber, Tara Simpson, Marty Neill, John Girvin, Greg O'Hanlon, Clare Rowland, Tim Spear, Ronan Cunningham, Tom Grey, Stevie Morrow, Steve Orr, Kevin Parker, John Reid, James Blundell, Mary McKenna, Mark Nagurski, Alan Hook, Jon Brookes, Conal Loughrey, Paul Graham, Frankie Colclough, and countless others (whom I will be kicking myself that I've forgotten) for all the meetings, the chats, the ideas, and the collaborations.

Thanks to Tim Brundle, Matt Johnson, and Alan Thorburn for their support and for introducing me to the people who would inspire thoughts that would spur me on to bigger challenges with data. An enormous thank you to Thomas Spinks for having faith in me, without him there wouldn't have been a career in computing.

In relation to the challenge of writing a book I have to thank Ben Hammersley, Alistair Croll, Alasdair Allan, and John Foreman for their advice and support throughout the whole process.

I also must thank my dear friend, Colin McHale, who, on one late evening while waiting for the soccer data to refresh, taught me Perl on the back of a KitKat wrapper, thus kick-starting a journey of software development.

Finally, to my wife, Wendy, and my daughter, Clarissa, for absolutely everything and encouraging me to do this book to the best of my nerdy ability. I couldn't have done it without you both. And to the Bell family—George, Maggie and my sister Fern—who have encouraged my computing journey from a very early age.

During the course of writing this book, musical enlightenment was brought to me by St. Vincent, Trey Gunn, Suzanne Vega, Tackhead, Peter Gabriel, Doug Wimbish, King Crimson, and Level 42.

Contents

Introduction

Data, data, data. You can't have escaped the headlines, reports, white papers, and even television coverage on the rise of Big Data and data science. The push is to learn, synthesize, and act upon all the data that comes out of social media, our phones, our hardware devices (otherwise known as "The Internet of Things"), sensors, and basically anything that can generate data.

The emphasis of most of this marketing is about data volumes and the velocity at which it arrives. Prophets of the data flood tell us we can't process this data fast enough, and the marketing machine will continue to hawk the services we need to buy to achieve all such speed. To some degree they are right, but it's worth stopping for a second and having a proper think about the task at hand.

Data mining and machine learning have been around for a number of years already, and the huge media push surrounding Big Data has to do with data volume. When you look at it closely, the machine learning algorithms that are being applied aren't any different from what they were years ago; what is new is how they are applied at scale. When you look at the number of organizations that are creating the data, it's really, in my opinion, the minority. Google, Facebook, Twitter, Netflix, and a small handful of others are the ones getting the majority of mentions in the headlines with a mixture of algorithmic learning and tools that enable them to scale. So, the real question you should ask is, "How does all this apply to the rest of us?"

I admit there will be times in this book when I look at the Big Data side of machine learning—it's a subject I can't ignore—but it's only a small factor in the overall picture of how to get insight from the available data. It is important to remember that I am talking about tools, and the key is figuring out which tools are right for the job you are trying to complete. Although the "tech press"

might want Hadoop stories, Hadoop is not always the right tool to use for the task you are trying to complete.

Aims of This Book

This book is about machine learning and not about Big Data. It's about the various techniques used to gain insight from your data. By the end of the book, you will have seen how various methods of machine learning work, and you will also have had some practical explanations on how the code is put together, leaving you with a good idea of how you could apply the right machine learning techniques to your own problems.

There's no right or wrong way to use this book. You can start at the beginning and work your way through, or you can just dip in and out of the parts you need to know at the time you need to know them.

"Hands-On" Means Hands-On

Many books on the subject of machine learning that I've read in the past have been very heavy on theory. That's not a bad thing. If you're looking for in-depth theory with really complex looking equations, I applaud your rigor. Me? I'm more hands-on with my approach to learning and to projects. My philosophy is quite simple:

- Start with a question in mind.
- Find the theory I need to learn.
- Find lots of examples I can learn from.
- Put them to work in my own projects.

As a software developer, I personally like to see lots of examples. As a teacher, I like to get as much hands-on development time as possible but also get the message across to students as simply as possible. There's something about fingers on keys, coding away on your IDE, and getting things to work that's rather appealing, and it's something that I want to convey in the book.

Everyone has his or her own learning styles. I believe this book covers the most common methods, so everybody will benefit.

"What About the Math?"

Like arguing that your favorite football team is better than another, or trying to figure out whether Jimmy Page is a better guitarist than Jeff Beck

(I prefer Beck), there are some things that will be debated forever and a day. One such debate is how much math you need to know before you can start to do machine learning.

Doing machine learning and learning the theory of machine learning are two very different subjects. To learn the theory, a good grounding in math is required. This book discusses a hands-on approach to machine learning. With the number of machine learning tools available for developers now, the emphasis is not so much on how these tools work but how you can make these tools work for you. The hard work has been done, and those who did it deserve to be credited and applauded.

"But You Need a PhD!"

There's nothing like a statement from a peer to stop you dead in your tracks. A long-running debate rages about the level of knowledge you need before you can start doing analysis on data or claim that you are a "data scientist." (I'll rip that term apart in a moment.) Personally, I believe that if you'd like to take a number of years completing a degree, then pursuing the likes of a master's degree and then a PhD, you should feel free to go that route. I'm a little more pragmatic about things and like to get reading and start doing.

Academia is great; and with the large number of online courses, papers, websites, and books on the subject of math, statistics, and data mining, there's enough to keep the most eager of minds occupied. I dip in and out of these resources a lot.

For me, though, there's nothing like getting my hands dirty, grabbing some data, trying out some methods, and looking at the results. If you need to brush up on linear regression theory, then let me reassure you now, there's plenty out there to read, and I'll also cover that in this book.

Lastly, can one gentleman or lady ever be a "data scientist?" I think it's more likely for a team of people to bring the various skills needed for machine learning into an organization. I talk about this some more in Chapter 2.

So, while others in the office are arguing whether to bring some PhD brains in on a project, you can be coding up a decision tree to see if it's viable.

What Will You Have Learned by the End?

Assuming that you're reading the book from start to finish, you'll learn the common uses for machine learning, different methods of machine learning, and how to apply real-time and batch processing.

There's also nothing wrong with referencing a specific section that you want to learn. The chapters and examples were created in such a way that there's no dependency to learn one chapter over another.

The aim is to cover the common machine learning concepts in a practical manner. Using the existing free tools and libraries that are available to you, there's little stopping you from starting to gain insight from the existing data that you have.

Balancing Theory and Hands-On Learning

There are many books on machine learning and data mining available, and finding the balance of theory and practical examples is hard. When planning this book I stressed the importance of practical and easy-to-use examples, providing step-by-step instruction, so you can see how things are put together.

I'm not saying that the theory is light, because it's not. Understanding what you want to learn or, more importantly, how you want to learn, will determine how you read this book.

The first two chapters focus on defining machine learning and data mining, using the tools and their results in the real world, and planning for machine learning. The main chapters (3 through 8) concentrate on the theory of different types of machine learning, using walkthrough tutorials, code fragments with explanations, and other handy things to ensure that you learn and retain the information presented.

Finally, you'll look at real-time and batch processing application methods and how they can integrate with each other. Then you'll look at Apache Spark and R, which is the language rooted in statistics.

Outline of the Chapters

Chapter 1 considers the question, "What is machine learning?" and looks at the definition of machine learning, where it is used, and what type of algorithmic challenges you'll encounter. I also talk about the human side of machine learning and the need for future proofing your models and work.

Before any real coding can take place, you need to plan. Chapter 2, "How to Plan for Machine Learning," concentrates on planning for machine learning. Planning includes engaging with data science teams, processing, defining storage requirements, protecting data privacy, cleaning data, and understanding that there is rarely one solution that fits all elements of your task. In Chapter 2 you also work through some handy Linux commands that will help you maintain the data before it goes for processing.

A decision tree is a common machine learning practice. Using results or observed behaviors and various input data (signals, features) in models, you can predict outcomes when presented with new data. Chapter 3 looks at designing decision tree learning with data and coding an example using Weka.

Bayesian networks represent conditional dependencies against a set of random variables. In Chapter 4 you construct some simple examples to show you how Bayesian networks work and then look at some code to use.

Inspired by the workings of the central nervous system, neural network models are still used in deep learning systems. Chapter 5 looks at how this branch of machine learning works and shows you an example with inputs feeding information into a network.

If you are into basket analysis, then you'll like Chapter 6 on association rule learning and finding relations within large datasets. You'll have a close look at the Apriori algorithm and how it's used within the supermarket industry today.

Support vector machines are a supervised learning method to analyze data and recognize patterns. In Chapter 7 you look at text classification and other examples to see how it works.

Chapter 8 covers clustering—grouping objects—which is perfect for the likes of segmentation analysis in marketing. This approach is the best method of machine learning for attempting some trial-and-error suggestions during the initial learning phases.

Chapters 9 and 10 are walkthrough tutorials. The example in Chapter 9 concerns real-time processing. You use Spring XD, a "data ingesting engine," and the streaming Twitter API to gather tweets as they happen.

In Chapter 10, you look at machine learning as a batch process. With the data acquired in Chapter 9, you set up a Hadoop cluster and run various jobs. You also look at the common issue of acquiring data from databases with Sqoop, performing customer recommendations with Mahout, and analyzing annual customer data with Hadoop and Pig.

Chapter 11 covers one of the newer entrants to the machine learning arena. The chapter looks at Apache Spark and also introduces you to the Scala language and performing SQL-like queries with in-memory data.

For a long time the R language has been used by statistics people the world over. Chapter 12 examines at the R language. With it you perform some of the machine learning algorithms covered in the previous chapters.

Source Code for This Book

All the code that is explained in the chapters of the book has been saved on a Github repository for you to download and try. The address for the repository is `https://github.com/jasebell/mlbook`. You can also find it on the Wiley website at `www.wiley.com/go/machinelearning`.

The examples are all in Java. If you want to extend your knowledge into other languages, then a search around the Github site might lead you to some interesting examples.

Code has been separated by chapter; there's a folder in the repository for each of the chapters. If any extra libraries are required, there will be a note in the README file.

Using Git

Git is a version-control system that is widely used in business and the open source software community. If you are working in teams, it becomes very useful because you can create branches of codebase to work on then merge changes afterwards.

The uses for Git in this book are limited, but you need it for "cloning" the repository of examples if you want to use them.

To clone the examples for this book, use the following commands:

```
$mkdir mlbookexamples
$cd mlbookexamples
$git clone https://github.com/jasebell/mlbook.git
```

You see the progress of the cloning and, when it's finished, you're able to change directory to the newly downloaded folder and look at the code samples.

What Is Machine Learning?

Let's start at the beginning, looking at what machine learning actually is, its history, and where it is used in industry. This chapter also describes some of the software used throughout the book so you can have everything installed and be ready to get working on the practical things.

History of Machine Learning

So, what is the definition of machine learning? Over the last six decades, several pioneers of the industry have worked to steer us in the right direction.

Alan Turing

In his 1950 paper, "Computing Machinery and Intelligence," Alan Turing asked, "Can machines think?" (See `www.csee.umbc.edu/courses/471/papers/turing .pdf` for the full paper.) The paper describes the "Imitation Game," which involves three participants—a human acting as a judge, another human, and a computer that is attempting to convince the judge that it is human. The judge would type into a terminal program to "talk" to the other two participants. Both the human and the computer would respond, and the judge would decide which response came from the computer. If the judge couldn't consistently tell the difference between the human and computer responses then the computer won the game.

The test continues today in the form of the Loebner Prize, an annual competition in artificial intelligence. The aim is simple enough: Convince the judges that they are chatting to a human instead of a computer chat bot program.

Arthur Samuel

In 1959, Arthur Samuel defined machine learning as, "[A] Field of study that gives computers the ability to learn without being explicitly programmed." Samuel is credited with creating one of the self-learning computer programs with his work at IBM. He focused on games as a way of getting the computer to learn things.

The game of choice for Samuel was checkers because it is a simple game but requires strategy from which the program could learn. With the use of alpha-beta evaluation pruning (eliminating nodes that do not need evaluating) and minimax (minimizing the loss for the worst case) strategies, the program would discount moves and thus improve costly memory performance of the program.

Samuel is widely known for his work in artificial intelligence, but he was also noted for being one of the first programmers to use hash tables, and he certainly made a big impact at IBM.

Tom M. Mitchell

Tom M. Mitchell is the Chair of Machine Learning at Carnegie Mellon University. As author of the book *Machine Learning* (McGraw-Hill, 1997), his definition of machine learning is often quoted:

> **A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with the experience E.**

The important thing here is that you now have a set of objects to define machine learning:

- Task (T), either one or more
- Experience (E)
- Performance (P)

So, with a computer running a set of tasks, the experience should be leading to performance increases.

Summary Definition

Machine learning is a branch of artificial intelligence. Using computing, we design systems that can learn from data in a manner of being trained. The systems might learn and improve with experience, and with time, refine a model that can be used to predict outcomes of questions based on the previous learning.

Algorithm Types for Machine Learning

There are a number of different algorithms that you can employ in machine learning. The required output is what decides which to use. As you work through the chapters, you'll see the different algorithm types being put to work. Machine learning algorithms characteristically fall into one of two learning types: supervised or unsupervised learning.

Supervised Learning

Supervised learning refers to working with a set of labeled training data. For every example in the training data you have an input object and an output object. An example would be classifying Twitter data. (Twitter data is used a lot in the later chapters of the book.) Assume you have the following data from Twitter; these would be your input data objects:

```
Really loving the new St Vincent album!
#fashion I'm selling my Louboutins! Who's interested? #louboutins
I've got my Hadoop cluster working on a load of data. #data
```

In order for your supervised learning classifier to know the outcome result of each tweet, you have to manually enter the answers; for clarity, I've added the resulting output object at the start of each line.

```
music     Really loving the new St Vincent album!
clothing    #fashion I'm selling my Louboutins! Who's interested? #louboutins
bigdata    I've got my Hadoop cluster working on a load of data. #data
```

Obviously, for the classifier to make any sense of the data, when run properly, you have to work manually on a lot more input data. What you have, though, is a training set that can be used for later classification of data.

There are issues with supervised learning that must be taken into account. The *bias-variance dilemma* is one of them: how the machine learning model performs accurately using different training sets. High bias models contain restricted learning sets, whereas high variance models learn with complexity against noisy training data. There's a trade-off between the two models. The key is where to settle with the trade-off and when to apply which type of model.

Unsupervised Learning

On the opposite end of this spectrum is unsupervised learning, where you let the algorithm find a hidden pattern in a load of data. With unsupervised learning there is no right or wrong answer; it's just a case of running the machine learning algorithm and seeing what patterns and outcomes occur.

Unsupervised learning might be more a case of data mining than of actual learning. If you're looking at clustering data, then there's a good chance you're going to spend a lot of time with unsupervised learning in comparison to something like artificial neural networks, which are trained prior to being used.

The Human Touch

Outcomes will change, data will change, and requirements will change. Machine learning cannot be seen as a write-it-once solution to problems. Also, it requires human hands and intuition to write these algorithms. Remember that Arthur Samuel's checkers program basically improved on what the human had already taught it. The computer needed a human to get it started, and then it built on that basic knowledge. It's important that you remember that.

Throughout this book I talk about the importance of knowing what question you are trying to answer. The question is the cornerstone of any data project, and it starts with having open discussions and planning. (Read more about this in Chapter 2, "Planning for Machine Learning.")

It's only in rare circumstances that you can throw data at a machine learning routine and have it start to provide insight immediately.

Uses for Machine Learning

So, what can you do with machine learning? Quite a lot, really. This section breaks things down and describes how machine learning is being used at the moment.

Software

Machine learning is widely used in software to enable an improved experience with the user. With some packages, the software is learning about the user's behavior after its first use. After the software has been in use for a period of time it begins to predict what the user wants to do.

Spam Detection

For all the junk mail that gets caught, there's a good chance a Bayesian classification filter is doing the work to catch it. Since the early days of SpamAssassin to Google's work in Google Mail, there's been some form of learning to figure out whether a message is good or bad.

Spam detection is one of the classic uses of machine learning, and over time the algorithms have gotten better and better. Think about the e-mail program

that you use. When it sees a message it thinks is junk, it asks you to confirm whether it is junk or isn't. If you decide that the message is spam, the system learns from that message and from the experience. Future messages will, hopefully, be treated correctly from then on.

Voice Recognition

Apple's Siri service that is on many iOS devices is another example of software machine learning. You ask Siri a question, and it works out what you want to do. The result might be sending a tweet or a text message, or it could be setting a calendar appointment. If Siri can't work out what you're asking of it, it performs a Google search on the phrase you said.

Siri is an impressive service that uses a device and cloud-based statistical model to analyze your phrase and the order of the words in it to come up with a resulting action for the device to perform.

Stock Trading

There are lots of platforms that aim to help users make better stock trades. These platforms have to do a large amount of analysis and computation to make recommendations. From a machine learning perspective, decisions are being made for you on whether to buy or sell a stock at the current price. It takes into account the historical opening and closing prices and the buy and sell volumes of that stock.

With four pieces of information (the low and high prices plus the daily opening and closing prices) a machine learning algorithm can learn trends for the stock. Apply this with all stocks in your portfolio, and you have a system to aid you in the decision whether to buy or sell.

Bitcoins are a good example of algorithmic trading at work; the virtual coins are bought and sold based on the price the market is willing to pay and the price at which existing coin owners are willing to sell.

The media is interested in the high-speed variety of algorithmic trading. The ability to perform many thousands of trades each second based on algorithmic prediction is a very compelling story. A huge amount of money is poured into these systems and how close they can get the machinery to the main stock trading exchanges. Milliseconds of network latency can cost the trading house millions in trades if they aren't placed in time.

About 70 percent of trades are performed by machine and not by humans on the trading floor. This is all very well when things are going fine, but when a problem occurs it can be minutes before the fault is noticed, by which time many trades have happened. The flash crash in May 2010, when the Dow Jones

industrial average dove 600 points, is a good example of when this problem occurred.

Robotics

Using machine learning, robots can acquire skills or learn to adapt to the environment in which they are working. Robots can acquire skills such as object placement, grasping objects, and locomotion skills through either automated learning or learning via human intervention.

With the increasing amount of sensors within robotics, other algorithms could be employed outside of the robot for further analysis.

Medicine and Healthcare

The race is on for machine learning to be used in healthcare analytics. A number of startups are looking at the advantages of using machine learning with Big Data to provide healthcare professionals with better-informed data to enable them to make better decisions.

IBM's famed Watson supercomputer, once used to win the television quiz program *Jeopardy* against two human contestants, is being used to help doctors. Using Watson as a service on the cloud, doctors can access learning on millions of pages of medical research and hundreds of thousands of pieces of information on medical evidence.

With the number of consumers using smartphones and the related devices for collating a range of health information—such as weight, heart rate, pulse, pedometers, blood pressure, and even blood glucose levels—it's now possible to track and trace user health regularly and see patterns in dates and times. Machine learning systems can recommend healthier alternatives to the user via the device.

Although it's easy enough to analyze data, protecting the privacy of user health data is another story. Obviously, some users are more concerned about how their data is used, especially in the case of it being sold to third-party companies. The increased volume of analytics in healthcare and medicine is new, but the privacy debate will be the deciding factor about how the algorithms will ultimately be used.

Advertising

For as long as products have been manufactured and services have been offered, companies have been trying to influence people to buy their products. Since 1995, the Internet has given marketers the chance to advertise directly to our screens without needing television or large print campaigns. Remember the

thought of cookies being on our computers with the potential to track us? The race to disable cookies from browsers and control who saw our habits was big news at the time.

Log file analysis is another tactic that advertisers use to see the things that interest us. They are able to cluster results and segment user groups according to who may be interested in specific types of products. Couple that with mobile location awareness and you have highly targeted advertisements sent directly to you.

There was a time when this type of advertising was considered a huge invasion of privacy, but we've gradually gotten use to the idea, and some people are even happy to "check in" at a location and announce their arrival. If you're thinking your friends are the only ones watching, think again. In fact, plenty of companies are learning from your activity. With some learning and analysis, advertisers can do a very good job of figuring out where you'll be on a given day and attempt to push offers your way.

Retail and E-Commerce

Machine learning is heavily used in retail, both in e-commerce and bricks-and-mortar retail. At a high level, the obvious use case is the loyalty card. Retailers that issue loyalty cards often struggle to make sense of the data that's coming back to them. Because I worked with one company that analyzes this data, I know the pain that supermarkets go through to get insight.

UK supermarket giant Tesco is the leader when it comes to customer loyalty programs. The Tesco Clubcard is used heavily by customers and gives Tesco a great view of customer purchasing decisions. Data is collected from the point of sale (POS) and fed back to a data warehouse. In the early days of the Clubcard, the data couldn't be mined fast enough; there was just too much. As processing methods improved over the years, Tesco and marketing company Dunn Humby have developed a good strategy for understanding customer behavior and shopping habits and encouraging customers to try products similar to their usual choices.

An American equivalent is Target, which runs a similar sort of program that tracks every customer engagement with the brand, including mailings, website visits, and even in-store visits. From the data warehouse, Target can fine-tune how to get the right communication method to the right customers in order for them to react to the brand. Target learned that not every customer wants an e-mail or an SMS message; some still prefer receiving mail via the postal service.

The uses for machine learning in retail are obvious: Mining baskets and segmenting users are key processes for communicating the right message to the customer. On the other hand, it can be too accurate and cause headaches. Target's "baby club" story, which was widely cited in the press as a huge privacy

danger in Big Data, showed us that machine learning can easily determine that we're creatures of habit, and when those habits change they will get noticed.

TARGET'S PRIVACY ISSUE

Target's statistician, Andrew Pole, analyzed basket data to see whether he could determine when a customer was pregnant. A select number of products started to show up in the analysis, and Target developed a pregnancy prediction score. Coupons were sent to customers who were predicted to be pregnant according to the newly mined score. That was all very well until the father of a teenage girl contacted his local store to complain about the baby coupons that were being sent to his daughter. It turned out that Target predicted the girl's pregnancy before she had told her father that she was pregnant.

For all the positive uses of machine learning, there are some urban myths, too. For example, you might have heard the "beer and diapers" story associated with Walmart and other large retailers. The idea is that the sales of beer and diapers both increase on Fridays, suggesting that mothers were going out and dads would stock up on beer for themelves and diapers for the little ones they were looking after. It turned out to be a myth, but this still doesn't stop marketing companies from wheeling out the story (and believing it's true) to organizations who want to learn from their data.

Another myth is that the heavy metal band Iron Maiden would mine bittorrent data to figure out which countries were illegally downloading their songs and then fly to those locations to play concerts. That story got the marketers and media very excited about Big Data and machine learning, but sadly it's untrue. That's not to say that these things can't happen someday; they just haven't happened yet.

Gaming Analytics

We've already established that checkers is a good candidate for machine learning. Do you remember those old chess computer games with the real plastic pieces? The human player made a move and then the computer made a move. Well, that's a case of machine learning planning algorithms in action. Fastforward a few decades (the chess computer still feels like yesterday to me) to today when the console market is pumping out analytics data every time you play your favorite game.

Microsoft has spent time studying the data from Halo 3 to see how players perform on certain levels and also to figure out when players are using cheats. Fixes have been created based on the analysis of data coming back from the consoles.

Microsoft also worked on Drivatar, which is incorporated into the driving game Forza Motorsport. When you first play the game, it knows nothing about your driving style. Over a period of practice laps the system learns your style, consistency, exit speeds on corners, and your positioning on the track. The sampling happens over three laps, which is enough time to see how your profile behaves. As time progresses the system continues to learn from your driving patterns. After you've let the game learn your driving style the game opens up new levels and lets you compete with other drivers and even your friends.

If you have children, you might have seen the likes of Nintendogs (or cats), a game in which a person is tasked with looking after an on-screen pet. (Think Tamagotchi, but on a larger scale.) Algorithms can work out when the pet needs to play, how to react to the owner, and how hungry the pet is.

It's still the early days of game companies putting machine learning into infrastructure to make the games better. With more and more games appearing on small devices, such as those with the iOS and Android platforms, the real learning is in how to make players come back and play more and more. Analysis can be performed about the "stickiness" of the game—do players return to play again or do they drop off over a period of time in favor of something else? Ultimately there's a trade-off between the level of machine learning and gaming performance, especially in smaller devices. Higher levels of machine learning require more memory within the device. Sometimes you have to factor in the limit of what you can learn from within the game.

The Internet of Things

Connected devices that can collate all manner of data are sprouting up all over the place. Device-to-device communication is hardly new, but it hadn't really hit the public minds until fairly recently. With the low cost of manufacture and distribution, now devices are being used in the home just as much as they are in industry.

Uses include home automation, shopping, and smart meters for measuring energy consumption. These things are in their infancy, and there's still a lot of concern on the security aspects of these devices. In the same way mobile device location is a concern, companies can pinpoint devices by their unique IDs and eventually associate them to a user.

On the plus side, the data is so rich that there's plenty of opportunity to put machine learning in the heart of the data and learn from the devices' output. This may be as simple as monitoring a house to sense ambient temperature—for example, is it too hot or too cold?

It's very early days for the Internet of things, but there's a lot of groundwork happening that is leading to some interesting outcomes. With the likes of Arduino and Raspberry Pi computers, it's relatively cheap to get started measuring the

likes of motion, temperature, and sound and then extracting the data for analysis, either after it's been collated or in real time.

Languages for Machine Learning

This book uses the Java programming language for the working examples. The reasons are simple: It's a widely used language, and the libraries are well supported. Java isn't the only language to be used for machine learning—far from it. If you're working for an existing organization, you may be restricted to the languages used within it.

With most languages, there is a lot of crossover in functionality. With the languages that access the Java Virtual Machine (JVM) there's a good chance that you'll be accessing Java-based libraries. There's no such thing as one language being "better" than another. It's a case of picking the right tool for the job. The following sections describe some of the other languages that you can use for machine learning.

Python

The Python language has increased in usage, because it's easy to learn and easy to read. It also has some good machine learning libraries, such as scikit-learn, PyML, and pybrain. Jython was developed as a Python interpreter for the JVM, which may be worth investigating.

R

R is an open source statistical programming language. The syntax is not the easiest to learn, but I do encourage you to have a look at it. It also has a large number of machine learning packages and visualization tools. The RJava project allows Java programmers to access R functions from Java code. For a basic introduction to R, have a look at Chapter 12.

Matlab

The Matlab language is used widely within academia for technical computing and algorithm creation. Like R, it also has a facility for plotting visualizations and graphs.

Scala

A new breed of languages is emerging that takes advantage of Java's runtime environment, which potentially increases performance, based on the threading

architecture of the platform. Scala (which is an acronym for *Sca*lable *La*nguage) is one of these, and it is being widely used by a number of startups.

There are machine learning libraries, such as ScalaNLP, but Scala can access Java jar files, and it can also implement the likes of Classifier4J and Mahout, which are covered in this book. It's also core to the Apache Spark project, which is covered in Chapter 11.

Clojure

Another JVM-based language, Clojure, is based on the Lisp programming language. It's designed for concurrency, which makes it a great candidate for machine learning applications on large sets of data.

Ruby

Many people know about the Ruby language by association with the Ruby On Rails web development framework, but it's also used as a standalone language. The best way to integrate machine learning frameworks is to look at JRuby, which is a JVM-based alternative that enables you to access the Java machine learning libraries.

Software Used in This Book

The hands-on elements in the book use a number of programs and packages to get the algorithms and machine learning working.

To keep things easy, I strongly advise that you create a directory on your system to install all these packages. I'm going to call mine `mlbook`:

```
$mkdir ~/mlbook
$cd ~/mlbook
```

Checking the Java Version

As the programs used in the book rely on Java, you need to quickly check the version of Java that you're using. The programs require Java 1.6 or later. To check your version, open a terminal window and run the following:

```
$ java -version
java version "1.7.0_40"
Java(TM) SE Runtime Environment (build 1.7.0_40-b43)
Java HotSpot(TM) 64-Bit Server VM (build 24.0-b56, mixed mode)
```

If you are running a version older than 1.6, then you need to upgrade your Java version. You can download the current version from www.oracle.com/technetwork/java/javase/downloads/index.html.

Weka Toolkit

Weka (Waikato Environment for Knowledge Acquisition) is a machine learning and data mining toolkit written in Java by the University of Waikato in New Zealand. It provides a suite of tools for learning and visualization via the supplied workbench program or the command line. Weka also enables you to retrieve data from existing data sources that have a JDBC driver. With Weka you can do the following:

- Preprocessing data
- Clustering
- Classification
- Regression
- Association rules

The Weka toolkit is widely used and now supports the Big Data aspects by interfacing with Hadoop for clustered data mining.

You can download Weka from the University of Waikato website at `www.cs.waikato.ac.nz/ml/weka/downloading.html`. There are versions of Weka available for Linux, Mac OSX, and Windows. To install Weka on Linux, you just need to unzip the supplied file to a directory. On Mac OSX and Windows, an installer program is supplied that will unzip all the required files for you.

Mahout

The Mahout machine learning libraries are an open source project that are part of the Apache project. The key feature of Mahout is its *scalability*; it works either on a single node or a cluster of machines. It has tight integration with the Hadoop Map/Reduce paradigm to enable large-scale processing.

Mahout supports a number of algorithms including

- Naive Bayes Classifier
- K Means Clustering
- Recommendation Engines
- Random Forest Decision Trees
- Logistic Regression Classifier

There's no workbench in Mahout like there is in the Weka toolkit, but the emphasis is on integrating machine learning library code within your projects. There are a wealth of examples and ready-to-run programs that can be used with your existing data.

You can download Mahout from `www.apache.org/dyn/closer.cgi/mahout/`. As Mahout is platform independent, there's one download that covers all the operating systems. To install the download, all you have to do is unzip Mahout into a directory and update your path to find the executable files.

SpringXD

Whereas Weka and Mahout concentrate on algorithms and producing the knowledge you need, you must also think about acquiring and processing data.

Spring XD is a "data ingestion engine" that reads in, processes, and stores raw data. It's highly customizable with the ability to create processing units. It also integrates with all the other tools mentioned in this chapter.

Spring XD is relatively new, but it's certainly useful. It not only relates to Internet-based data, it can also ingest network and system messages across a cluster of machines.

You can download the Spring XD distribution from `http://projects.spring.io/spring-xd/`. The link for the zip file is in the Quick Start section.

After the zip file has downloaded you need to unzip the distribution into a directory. For a detailed walkthrough of using Spring XD, read Chapter 9, "Machine Learning in Real Time with Spring XD."

Hadoop

Unless you've been living on some secluded island without power and an Internet connection, you will have heard about the savior of Big Data: Hadoop. Hadoop is very good for processing Big Data, but it's not a required tool. In this book, it comes into play in Chapter 10, "Machine Learning as a Batch Process."

Hadoop is a framework for processing data in parallel. It does this using the MapReduce pattern, where work is divided into blocks and is distributed across a cluster of machines. You can use Hadoop on a single machine with success; that's what this book covers.

There are two versions of Hadoop. This book uses version 1.2.1.

The Apache Foundation runs a series of mirror download servers and refers you to the ones relevant to your location. The main download page is at `www.apache.org/dyn/closer.cgi/hadoop/common/`.

After you have picked your mirror site, navigate your way to hadoop-1.2.1 releases and download `hadoop-1.2.1-bin.tar.gz`. Unzip and untar the distribution to a directory.

If you are running a Red Hat or Debian server, you can download the respective `.rpm` or `.deb` files and install them via the package installer for your operating system. If preferred, Debian and Ubuntu users can install Hadoop with the `apt-get` or `yum` command.

Using an IDE

Some discussions seem to spark furious debate in certain circles—for example, favorite actor/actress, best football team, and best integrated development environment (IDE).

I'm an Eclipse user. I'm also an IDEA user, and I have NetBeans as well. Basically, I use all three. There's no hard rule that IDE you should use, as they all do the same thing very well. The examples in this book use Eclipse (Juno release).

Data Repositories

One question that comes up again and again in my classes is "Where can I get data?" There are a few answers to this question, but the best answer depends on what you are trying to learn.

Data comes in all shapes and sizes, which is something discussed further in the next chapter. I strongly suggest that you take some time to hunt around the Internet for different data sets and look through them. You'll get a feel for how these things are put together. Sometimes you'll find comma separated variable (CSV) data, or you might find JSON or XML data.

Remember, some of the best learning comes from playing with the data. Having a question in mind that you are trying to answer with the data is a good start (and something you will see me refer to a number of times in this book), but learning comes from experimentation and improvement on results. So, I'm all for playing around with the data first and seeing what works. I hail from a very pragmatic background when it comes to development and learning. Although the majority of publications about machine learning have come from people with academic backgrounds—and I fully endorse and support them—we shouldn't discourage learning by doing.

The following sections describe some places where you can get plenty of data with which to play.

UC Irvine Machine Learning Repository

This machine learning repository consists of more than 270 data sets. Included in these sets are notes on the variable name, instances, and tasks the data would be associated with. You can find this repository at `http://archive.ics.uci .edu/ml/datasets`.

Infochimps

The data marketplace at Infochimps has been around for a few years. Although the company has expanded to cloud-based offerings, the data is still available to download at `www.infochimps.com/datasets`.

Kaggle

The competitions that Kaggle run have gained a lot of interest over the last couple of years. The 101 section on the site offers some data sets with which to experiment. You can find them at `www.kaggle.com/competitions.`

Summary

This chapter looked at what machine learning is, how it can be applied to different areas of business, and what tools you need to follow along with the remainder of the book.

Chapter 2 introduces you to planning for machine learning. It covers data science teams, cleaning, and different methods of processing data.

Planning for Machine Learning

This chapter looks at planning your machine learning projects, storage types, processing options and data input. The chapter also covers data quality and methods to validate and clean data before you do any analysis.

The Machine Learning Cycle

A machine learning project is basically a cycle of actions that need to be performed. (See Figure 2-1.)

Acquisition	• Collate the data
Prepare	• Data cleaning and quality
Process	• Run machine tools
Report	• Present the results
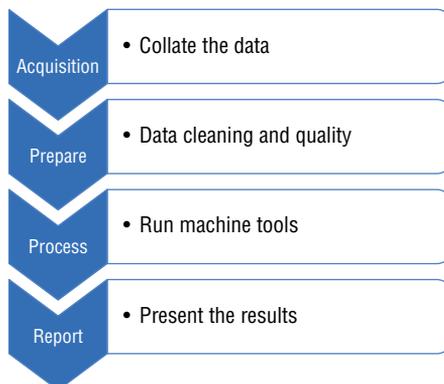
Figure 2-1: The machine learning process

You can acquire data from many sources; it might be data that's held by your organization or open data from the Internet. There might be one dataset, or there could be ten or more.

You must come to accept that data will need to be cleaned and checked for quality before any processing can take place. These processes occur during the prepare phase.

The processing phase is where the work gets done. The machine learning routines that you have created perform this phase.

Finally, the results are presented. Reporting can happen in a variety of ways, such as reinvesting the data back into a data store or reporting the results as a spreadsheet or report.

It All Starts with a Question

There seems to be a misconception that machine learning, like Big Data, is a case of throwing enough data at the problem that the answers magically appear. As much as I'd like to say this happens all the time, it doesn't. Machine learning projects start with a question or a hunch that needs investigating. I've encountered this quite a few times in speaking to people about their companies' data ambitions and what they are looking to achieve with the likes of machine learning and Hadoop.

Using a whiteboard, sticky notes, or even a sheet of paper, start asking questions like the following:

- Is there a correlation between our sales and the weather?
- Do sales on Saturday and Sunday generate the majority of revenue to the business compared to the other five days of the week?
- Can we plan what fashions to stock in the next three months by looking at Twitter data for popular hashtags?
- Can we tell when our customers become pregnant?

All these examples are reasonable questions, and they also provide the basis for proper discussion. Stakeholders will usually come up with the questions, and then the data project team (which might be one person—you!) can spin into action.

Without knowing the question, it's difficult to know where to start. Anyone who thinks the answers just pop out of thin air needs a polite, but firm, explanation of what has to happen for the answers to be discovered.

I Don't Have Data!

This sounds like a silly statement when you have a book on machine learning in your hands, but sometimes people just don't have the data.

In an ideal world, we expect companies to have well-groomed customer relationship management (CRM) systems and neat repositories of data that could be retrieved on a whim and copied nicely into a Hadoop filesystem, so countless MapReduce jobs could run (read more about Hadoop and MapReduce in Chapter 10, "Machine Learning as a Batch Process").

Data comes from a variety of sources. Plenty of open data initiatives are available, so you have a good chance of being able to find some data to work with.

Starting Local

Perhaps you could make a difference in your local community; see what data they have open with which you can experiment. New York City has a whole portal of open data with more than 1,100 datasets for citizens to download and learn from. Hackathons and competitions to encourage people to get involved and give back to the community. The results of the hackathons make a difference, because insights about how the local community is run are fed back to the event organizers. If you can't find the dataset you want then you are also encouraged to request it.

Competitions

If you fancy a real challenge, then think about entering competitions. One of the most famous was the Netflix Prize, which was a competition to improve the recommendation algorithm for the Netflix film service.

Teams who were competing downloaded sample sets of user data and worked on an algorithm to improve the predictions of movies that customers would like. The winning team was the one that improved the results by 10 percent. In 2009, the $1 million prize was awarded to "BellKor's Pragmatic Chaos." This triggered a new wave of competitions, letting the data out into the open so collaborative teams could improve things.

In 2010, Anthony Goldbloom founded Kaggle.com, which is a platform for predictive modeling and analytics competitions. Each competition posted has sample datasets and a brief of the desired outcome. Either teams or individuals can enter, and the most effective algorithms, very similar to the Netflix Prize, decided the winner.

Is competition effective? It seems to be. Kaggle has more than 100,000 data scientists registered from across the world. Organizations such as Facebook, NASA, GE, Wikipedia, and AllState have used the service to improve their products and even head-hunt top talent.

One Solution Fits All?

Machine learning is built up from a varying set of tools, languages, and techniques. It's fair to say that there is no one solution that fits most projects. As you will find in this chapter and throughout the book, I'll refer to various tools to get certain aspects of the job done. For example, there might be data in a relational database that needs extracting to a file before you can process.

Over the last few years, I've seen managers and developers with faces of complete joy and happiness when a data project is assigned. It's new, it's hip and, dare I say it, funky to be working on data projects. Then after the scale of the project comes into focus, I've seen the color drain from their faces. Usually this happens after the managers and developers see how many different elements are required to get things working for the project to succeed. And, like any major project, the specification from the stakeholders will change things along the way.

Defining the Process

Making anything comes down to process, whether that's baking a cake, brewing a cup of coffee, or planning a machine learning project. Processes can be refined as time goes on, but if you've never developed one before, then you can use the following process as a template.

Planning

During the late 1980s, I wrote many assignments and papers on the upcoming trend of the paperless office and how computers would one day transform the way day-to-day operations would be performed. Even without the Internet, it was easy to see that computers were changing how things were being done.

Skip ahead to the present day and you'll see that my desk is littered with paper, notebooks, sticky notes, and other scraps of information. The paperless office didn't quite make the changes I was expecting, and you need no more evidence than the state of my desk. I would show you a photograph, but it might prove embarrassing.

What I have found is that all projects start on paper. For me, it doesn't work to jump in and code; I find that method haphazard and error prone. I need to plan first. I use A5 Moleskin notebooks for notes and A4 and A3 artist drawing pads for large diagrams. They're on my desk, in my bag, and in my jacket pocket.

Whiteboards are good, too. Whiteboards hold lots of ideas and diagrams, but I find they can get out of control and messy after a while. There was once an office wall in Santa Clara that I covered in sticky notes. (I did take them down once I was finished. The team thought I was mad.)

Planning might take into account where the data is coming from, if it needs to be cleaned, what learning methods to use and what the output is going to look like. The main point is that these things can be changed at any time—the earlier in the process they change, the better. So it's worth taking the time to sit around a table with stakeholders and the team and figure out what you are trying achieve.

Developing

This process might involve algorithm development or code development. The more iterations you perform on the code the better it will be. Agile development processes work best; in agile development, you only work on what needs to be done without trying to future-proof the software as you go along. It's worth using some form of code repository site like Github or BitBucket to keep all your work private; it also means you can roll back to earlier versions if you're not happy with the way things are going.

Testing

In this case, *testing* means testing with data. You might use a random sample of the data or the full set. The important thing is to remind yourself that you're testing the process, so it's okay for things to not go as planned. If you push things straight to production, then you won't really know what's going to happen. With testing you can get an idea of the pain points. You might find data-loading issues, data-processing issues, or answers that just don't make sense. When you test, you have time to change things.

Reporting

Sit down with the stakeholders and discuss the test results. Do the results make sense? The developers and mathematicians might want to amend algorithms or the code. Stakeholders might have a new question to ask (this happens a lot), or perhaps you want to introduce some new data to get another angle on the

answers. Regardless of the situation, make sure the original people from the planning phase are back around the table again.

Refining

When everyone is happy with the way the process is going it's time to refine code and, if possible, the algorithms. With huge volumes of data, squeeze every ounce of performance you can from your code and the quicker the overall processing time will be. Think of a bobsled run; a slower start converts to a much slower finish.

Production

When all is tested, reviewed, and refined by the team, moving to production shouldn't be a big job. Be sure to give consideration to when this project will be run—is it an hourly/daily/weekly/monthly job? Will the data change wildly between the project going in to production and the next run?

Make sure the team reviews the first few production runs to ensure the results are as expected, and then look at the project as a whole and see if it's meeting the criteria of the stakeholders. Things might need to be refined. As you probably already know, software is rarely finished.

Building a Data Team

A *data scientist* is someone who can bring the facets of data processing, analytics, statistics, programming, and visualization to a project. With so many skill sets in action, even for the smallest of projects, it's a lot to ask for one person to have all the necessary skills. In fact, I'd go as far to say that such a person might not exist—or is at least extremely rare. A data science team might touch on some, or all, of the following areas of expertise.

Mathematics and Statistics

Someone on the team needs to have a good head for mathematics—someone who isn't going to flinch when the words "linear regression" are mentioned in the interview. I'm not saying there's a minimum level of statistics you should know before embarking on any project, but knowledge of descriptive statistics (the mean, the mode, and the median), distributions, and outliers will give you a good grounding to start.

The debate will rage on about the level of mathematics needed in any machine learning project, but my opinion is that every project comes with its own set of complications. If new information needs to be learned, then there are plenty of sources out there from which you can learn.

If you have access to talented mathematicians, then your data team is a blessed group indeed.

Programming

Good programming talent is hard to come by, but I'm assuming that if you have this book in your hand then there's a good chance you're a programmer already. Taking algorithms and being able to transfer that to workable code can take time and planning. It's also worth knowing some of the Big Data tools, such as the Hadoop framework and Spring XD. (Read Chapters 9 and 10 for a comprehensive walkthrough on both technologies.)

Graphic Design

Visualizing data is important; it tells the story of your findings to the stakeholders or end users. Although much emphasis has been placed on the web for presentation with technologies such as D3 and Processing, don't forget the likes of BIRT, Jasper Reports, and Crystal Reports.

This book doesn't touch on visualization, but Appendix D, "Further Reading," includes some titles that will point you in the right direction.

Domain Knowledge

If, for example, you are working with medical data, then it would be beneficial to have someone who knows the medical field well. The same goes for retail; there's not much point trawling through rows of transactions if no one knows how to interpret how customers behave. Domain experts are the vital heroes in guiding the team through a project. There are some decisions that the domain expert will instinctively know.

Think of a border crossing with passport control. There might be many permutations of rules that are given depending on nationality, immigration rules, and so on. A domain expert would have this knowledge in place and make your life as the developer much easier and would help to get a solution up and running more quickly.

There's a notion that we don't need domain experts. I'm of the mind that we do, even if you only sit down and have coffee with someone who knows the domain. Always take a notebook and keep notes.

Data Processing

After you have a team in place and a rough idea of how all of this is going to get put together, it's time to turn your attention to what is going to do all the work for you. You must give thought to the frequency of the data process jobs

that will take place. If it will occur only once in a while, then it might be false economy investing in hardware over the long term. It makes more sense to start with what you have in hand and then add as you go along and as you notice growth in processing times and frequency.

Using Your Computer

Yes, you can use your own machine, either a desktop or a laptop. I do my development on an Apple MacBook Pro. I run the likes of Hadoop on this machine as it's pretty fast, and I'm not using terabytes of data. There's nothing to stop you from using your own machine; it's available and it saves financial outlay to get more machines. Obviously, there can be limitations. Processing a heavy job might mean you have to turn your attention to less processor-intensive things, but never rule out the option of using your own machine.

Operating systems like Linux and Mac OSX tend to be preferred over Windows, especially for Big Data–based operations. The best choice comes down to what you know best and what suits the project best in order to get the job done efficiently. I don't believe there's only one right way to do things.

A Cluster of Machines

Eventually you'll come across a scenario that requires you to use a cluster of machines to do the work. Frameworks like Hadoop are designed for use over clusters of machines, which make it possible for the distribution of work to be done in parallel. Ideally the machines should be on the same network to reduce network traffic latency.

At this point in time, it's also worthwhile to add a good system administrator to the data science team. Any performance that can be improved over the cluster will bring marked performance against the whole project.

Cloud-Based Services

If the thought of maintaining and paying for your own hardware does not appeal, then consider using some form of cloud-based service. Vendors such as Amazon, Rackspace, and others provide scalable servers where you can increase, or decrease, the number of machines and amount of power that you require. The advantage of these services is that they are "turn on/turn off" technology, enabling you to use only what you need.

Keep a close eye on the cost of cloud-based services, as they can sometimes prove more expensive than just using a standard hosting option over longer time periods. Some companies provide dedicated Big Data services if you require the likes of Hadoop to do your processing. With cloud-based services, it's always

important to turn the instance off, otherwise you'll be charged for the usage while the instance is active.

Data Storage

There are some decisions to make on how the data is going to be stored. This might be on a physical disc or deployed on a cloud-based solution.

Physical Discs

The most common form of storage is the one that you will more than likely have in your computer to start off with. The hard disc is adequate for testing and small jobs. You will notice a difference in performance between physical discs and solid state drives (SSD); the latter provides much faster performance. External drives are cheap, too, and provide a good storage solution for when data volumes increase.

Cloud-Based Storage

Plenty of cloud-based storage facilities are available to store your data as required. If you are looking at cloud-based processing, then you'll more than likely be purchasing some form of cloud-based storage to go with it. For example, if you use Amazon's Elastic Map Reduce (EMR) system, then you would be using it alongside the S3 storage solution.

Like cloud processing, storage based on the cloud will cost you on a monthly or annual basis. You also have to think about the bandwidth implications of moving large volumes of data from your office location to the cloud system, which is another cost to keep in mind.

Data Privacy

Data is power and with it comes an awful lot of responsibility. The privacy issue will always rage on in the hearts and minds of the users and the general public. Everyone has an opinion on the matter, and often people err on the side of caution.

Cultural Norms

Cultural expectations are difficult to measure. As the World Wide Web has progressed since the mid-1990s, there has been a privacy battle about everything

from how cookies were stored on your computer to how a multitude of companies are tracking locations, social interactions, ratings, and purchasing decisions through your mobile devices.

If you're collecting data via a website or mobile application, then there's an expectation that you will be giving something in return for user information. When you collect that information, it's only right to tell the user what you intend to do with the data.

Supermarket loyalty card schemes are a simple data-collecting exercise. For every basket that goes through the checkout, there's the potential that the customer has a loyalty card. In associating that customer with that basket of products you can start to apply machine learning. Over time you will be able to see the shopping habits of that customer—her average spend, the day of the week she shops—and the customer expects some form of discount promotion for telling you all this information.

So, how do you keep cultural norms onside? By giving customers a very clear opt-in or opt-out strategy.

Generational Expectations

During sessions of my iPhone development class, I open up with a discussion about personal data. I can watch the room divide instantly, and I can easily see the deciding factor: age.

Some people are more than happy to share with their friends, and the rest of the world, their location, what they are doing, and with whom. These people post pictures of their activities and tag them so they could be easily searched, rated, and commented on. They use Facebook, Instagram, Foursquare, Twitter, and other apps as a normal, everyday part of their lives.

The other group of people, who were older, were not comfortable with the concept of handing over personal information. Some of them thought that no one in their right minds would be interested in such information. Most couldn't see the point.

Although the generation gap might be closing and there is a steady relaxation of what people are willing to put on the Internet, developers have a responsibility to the suppliers of the information. You have to consider whether the results you generate will cause a concern to them or enhance their lives.

The Anonymity of User Data

You can learn from data, but users get touchy when their names are attached to it. Creating hashes of important data is a starting point, but it's certainly not

the end game. Consider my name as an MD5 hash. Using the Linux `md5sum` command I can find it out very easily, as shown here:

```
$ printf '%s' "Jason Bell" | md5sum
a7b19ed2ca59f8e94121b54f9f26333c  -
```

Now, I have a hash value, which is a good start, but it's still not really protecting my identity. You now know it and what it would possibly relate to if it were used as a user key in a machine learning process. It wouldn't take much time for a decent programmer with a list of first and last names to generate all the `md5` values for all the combinations.

Using a salt value is a better solution. A *salt value* is random data that's used with the piece of data to make it more secure and harder to crack.

Let's assume the salt value is the number of nanoseconds from the 1st January 1970. You take that and the string you're looking to hash:

```
$ printf '%s' "Jason Bell $(date +%sN)" | md5sum
40e46b48a873c30c80469dbbefaa5e16  -
```

There are different ways of handling the input string. You might want to remove spaces but the concept remains the same. The security of these hashes has to be maintained by you, so when the time comes to interpret the answers, you'll know which customers are doing the actions you are seeking. Hashes aren't just restricted to usernames or customer names; they can be applied to any data. Anything that you consider private information (known as personally identifiable information or PII)—something that you don't want any third party to see—must be hashed.

Don't Cross "The Creepy Line"

Be careful not to make the customer freak out by crossing the line in the sand that I call "the creepy line." It's the point where the horrified customer would shriek, "How did they know that?" For an example of a company and what they know about you visit the settings pages of your Google account (`https://www.google.com/settings/dashboard`) and have a look your web search history or your location history.

One near-legendary example in data science, Big Data, and machine learning circles is the story of Target and pregnant mothers, which was widely cited on the Internet because of Charles Duhigg's book *The Power of Habit* (Random House, 2011). What readers of the Internet forgot to realize was that Target had been using the same practice for years; the concept was originally run in 2002 as an exercise to see if there was a correlation between two things.

Good mathematics and item matching isolated a number of items that mothers-to-be started to buy. Target has enough data to predict what trimester of the pregnancy the mother is in. With an opt-in to the baby club this might have all passed without problem. But when an angry father rolls up to the store to enquire why his teenage daughter is receiving baby promotions and coupons, well, that's a different matter.

What does this example highlight? Well, apart from freaking out the customer, it causes undue pressure on the in-store staff. Everyone in the organization needs to be aware of the work that's going on. Also, the data team needs to be acutely aware of the social effect of their learning.

The UK supermarket chain Tesco started the Clubcard loyalty scheme in 1995; it holds more data than some governments on customer purchasing behavior, social classes, and income bracket. The store's data processing power is controlled by a marketing company, Dunn Humby, which runs the Clubcard and analyzes the data. What is the upside for the customer? Four times a year Clubcard members receive coupons for money off and incentives to buy items they normally purchase. The offers resemble the customers' typical shopping patterns, but other items are thrown in so it doesn't look like they've been stalked.

Mining the baskets is hardly a new idea (you'll be reading about other techniques in later chapters), but when the supermarket becomes large and the volumes of data are huge, the insight that can be gained becomes an enormous commercial advantage. The cost of this advantage is appearing to know the intimate shopping details of the customer even when they've not overtly given permission for you to send offers.

Data Quality and Cleaning

In an ideal world, you'd receive data and put it straight into the system for processing. Then your favorite actor or actress would hand you your favorite drink and pat you on the back for a job well done.

In the real world, data is messy, usually unclean, and error prone. The following sections offer some basic checks you should do, and I've included some sample data so you can see clearly what to look for.

The example data is a simple address book with a first name, last name, e-mail address, and age.

Presence Checks

First things first, check that data has been entered at all. Within web-based businesses, registration usually involves at least an e-mail address, first name, and last name. It's amazing how many times users will try to avoid putting in their names.

The presence check is simple enough. If the field length is empty or null, and that piece of data is important in the analysis, then you can't use records from which the data is missing.

	FIRSTNAME	LASTNAME	E-MAIL	AGE
Correct	Jason	Bell	me@domain.com	42
Incorrect		Bell		42

The first name and e-mail are missing from the example, so the record should really be fixed or rejected. In theory, the data could be used if knowing the customer was not important.

Type Checks

With relational databases you have schemas created, so there's already an expectation of what type of data is going where. If incorrect data is written to a field of a different data type, then the database engine will throw an error and complain at you.

In text data, such as CSV files, that's not the case, so it's worth looking at each field and ensuring that what you're expecting to see is valid.

```
#firstname, lastname, email, age
Jason,Bell,me@domain.com,42
42,Bell,me@domain.com,Jason
```

From the example, you can see that the first row of data is correct, but the second is wrong because the `firstname` field has a number in it and not a string type. There are a couple of things you could do here. The first option is to ignore the record, as it doesn't fit the data-quality check. The other option is to see if any other records have the same e-mail address and check the name against those records.

Length Checks

Field lengths must be checked, too; once again, relational databases exercise a certain amount of control, but textual data can be error-prone if people don't go with the general rules of the schema.

FIELD	LENGTH	GOOD	BAD
Firstname	10	Jason	Mr Jason Bell
Email	20	me@domain.com	jason.bell@thing.domain.com

Range Checks

Range or reasonableness checks are used with numeric or date ranges. Age ranges are the main talking point here. Until there are advances in scientific medicine to prolong life, you can make a fairly good assumption that the upper lifespan of someone is about 120. You can even play it safe and extend the upper range to 150; anyone who is older than that is lying or just trying to put a false value in to trip up the system.

FIELD	LOWER RANGE	UPPER RANGE
Age	0	120
Month	1	12

Format Checks

When you know that certain data must follow a given format then it's always good to check it. Regular expression knowledge is a big advantage here if you know it. E-mail addresses can be used and abused in web forms and database tables, so it's always a good idea to validate what you can at source.

There's much discussion in the developer world about what a correct e-mail regular expression actually is. The official standard for the e-mail address specification is RFC 5322. Correctly matching the e-mail address as a regular expression is a huge pattern. What you're looking for is something that will catch the majority of e-mail addresses:

```
[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-
z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?
```

The main thing to do is create a run of test cases with all the eventualities of an e-mail address you think you will come across. Don't just test it once; keep retesting it over time.

Postcodes and Zip codes are another source of formatting woe—especially UK postcodes. Regular expressions also help in this case, but sometimes an odd one slips through the testing. At the end of the day, this sort of thing is better left to specialized software or expert services.

The Britney Dilemma

Users being users will input all sorts of things, and it's really up to us to make sure that our software catches what it can. Although search strings aren't specific to machine learning, it is, however, a very interesting case of how different names can really mess up the results.

For instance, take the variations of the search term "Britney Spears" in a well-known search engine. In an ideal and slightly utopian vision, everyone would type her name perfectly into a text field box:

```
britney spears
```

Life rarely goes as planned, and users type what they think is right, such as the following:

```
brittany spears
brittney spears
britany spears
britny spears
briteny spears
britteny spears
briney spears
brittny spears
brintey spears
britanny spears
britiny spears
britnet spears
britiney spears
britaney spears
britnay spears
brithney spears
brtiney spears
birtney spears
brintney spears
briteney spears
bitney spears
brinty spears
brittaney spears
brittnay spears
britey spears
brittiny spears
```

If you were to put that through a Hadoop cluster looking for unique singer search terms, you'd be in a bit of a mess, as each of these would register a new result count.

What you want is something to weigh each term and see what it resembles. The simplest approach is to use a classifier to weigh each search term as it comes in. You know the correct term, so it's a case of running the incoming terms against the correct one and seeing what the confidence scoring is.

```java
package chapter2;

import java.util.ArrayList;
import java.util.List;

import net.sf.classifier4J.ClassifierException;
```

```java
import net.sf.classifier4J.vector.HashMapTermVectorStorage;
import net.sf.classifier4J.vector.TermVectorStorage;
import net.sf.classifier4J.vector.VectorClassifier;

public class BritneyDilemma {

    public BritneyDilemma() {
        List<String> terms = new ArrayList<String>();
        terms.add("brittany spears");
        terms.add("brittney spears");
        terms.add("britany spears");
        terms.add("britny spears");
        terms.add("briteny spears");
        terms.add("britteny spears");
        terms.add("briney spears");
        terms.add("brittny spears");
        terms.add("brintey spears");
        terms.add("britanny spears");
        terms.add("britiny spears");
        terms.add("britnet spears");
        terms.add("britiney spears");
        terms.add("christina aguilera");

        TermVectorStorage storage = new HashMapTermVectorStorage();
        VectorClassifier vc = new VectorClassifier(storage);
        String correctString = "britney spears";

        for (String term : terms) {
          try {
            vc.teachMatch("sterm", correctString);
            double result = vc.classify("sterm", term);
            System.out.println(term + " = " + result);
          } catch (ClassifierException e) {
            e.printStackTrace();
          }
        }
    }

    public static void main(String[] args) {
        BritneyDilemma bd = new BritneyDilemma();
    }
}
```

This code sample uses the Classifer4J library to run a basic vector space search on the incoming spellings of Britney; it then ranks them against the correct string. When this code is run, you get the following output:

```
brittany spears = 0.7071067811865475
brittney spears = 0.7071067811865475
britany spears = 0.7071067811865475
britny spears = 0.7071067811865475
briteny spears = 0.7071067811865475
```

```
britteny spears = 0.7071067811865475
briney spears = 0.7071067811865475
brittny spears = 0.7071067811865475
brintey spears = 0.7071067811865475
britanny spears = 0.7071067811865475
britiny spears = 0.7071067811865475
britnet spears = 0.7071067811865475
britiney spears = 0.7071067811865475
britaney spears = 0.7071067811865475
britnay spears = 0.7071067811865475
brithney spears = 0.7071067811865475
brtiney spears = 0.7071067811865475
birtney spears = 0.7071067811865475
brintney spears = 0.7071067811865475
briteney spears = 0.7071067811865475
bitney spears = 0.7071067811865475
brinty spears = 0.7071067811865475
brittaney spears = 0.7071067811865475
brittnay spears = 0.7071067811865475
britey spears = 0.7071067811865475
brittiny spears = 0.7071067811865475
christina aguilera = 0.0
```

The confidence is always a number between 0 and 0.9999. Just to prove that, putting the correct spelling in the list and running the program again would generate a positive score.

```
britney spears = 0.9999999999999998
```

Obviously, there's some preparation required, as you need to know the correct spellings of the search terms before you can run the classifier. This example just proves the point.

What's in a Country Name?

Data cleaning needs to be done in a variety of circumstances, but the most common reason is too many options were given in the first place.

A few years ago I was looking at a database for a hotel. Its data was gathered via a web-based enquiry form, but instead of offering a selection of countries from a drop-down list of countries, there was just an open text field. (Always remember that freedom of input, where it can be avoided, should be avoided.)

Let's consider this for a moment. If you take a country like Ireland then you might have the following entries for country name:

- Ireland
- Republic of Ireland
- Eire

- EIR
- Rep. of Ireland

All these are essentially the same place; the only exception would be Northern Ireland, which is still part of the United Kingdom.

What you have is a huge job to clean up the country field of a database. To fix this, you would have to find all the distinct names in the country field and associate them with a two-letter country code. So, Ireland and all the other names that were associated with Ireland become IE. You would have to do this for all the countries. Where possible, it's better to have tight control of the input data, as this will make things a lot easier when it comes to processing.

In programming terms, you could make each of the distinct countries a key in a HashMap and add a method to get the value of the corresponding input name.

```
package chapter2;

import java.util.HashMap;
import java.util.Map;

public class CountryHashMap {

    private Map<String, String> countries = new HashMap<String,
String>();

    public CountryHashMap() {
        countries.put("Ireland", "IE");
        countries.put("Eire", "IE");
        countries.put("Republic of Ireland", "IE");
        countries.put("Northern Ireland", "UK");
        countries.put("England", "UK");
        // you could add more or generate from a database.
    }

    public String getCountryCode(String country) {
        return countries.get(country);
    }

    public static void main(String[] args) {
        CountryHashMap chm = new CountryHashMap();
        System.out.println(chm.getCountryCode("Ireland"));
        System.out.println(chm.getCountryCode("Northern Ireland"));
    }
}
```

The preceding example is a basic piece of code that would automate the cleaning process in a short amount of time. However, you are strongly advised to look

at the source of the problem and refactor the input. If no change is made, then the same cost to the business will occur, as you'll have to clean the data again.

Ideally, to avoid having to do this sort of cleaning, you would employ verification strategies at the input stage. So, for example, if you're using web forms you should use JavaScript to validate the input before it's saved to the database. Other times you inherit data and occasionally have to employ such methods.

Dates and Times

For time series processing, you must ensure that you have a consistent set of dates to read. The format you choose is really up to you. International Standard ISO 8601 lays out the specification for date and time representations in a numerical format. The issue with the ISO 8601 standard is that it's not immune to the Y10K bug when timestamps will be incorrect after 19th January 2038. The Temps Atomique International (TAI) standard takes into account these issues.

Regardless of the language you are using, make yourself aware of how the date formatting and parsing routines work. For Java, have a look at the SimpleDateFormat API, which gives you a rundown on all the settings along with some useful examples. Use caution when running code on distributed systems and also with different time zones.

Table 2-1 shows some of the commonly used date/time formats.

Table 2-1: Commonly Used Date/Time Formats

DATE/TIME FORMAT	SIMPLEDATEFORMAT REPRESENTATION
2014-01-01	Yyyy-MM-dd
2014-01-01 11:59:00	Yyyy-MM-dd hh:mm:ss
1388577540	(Unix timestamps are like long variable types but with nano seconds added.)

I've seen many a database table with different date formats that have been saved as string types. Things have gotten better, but it's still something I keep in mind.

Final Thoughts on Data Cleaning

Data cleaning is a big deal, because it increases the chances of getting better results. For some Big Data projects, 80 percent of the project time is spent on

data cleaning before the actual analysis starts. It's important to keep this step high up in the project plan and manage time accordingly.

Thinking about Input Data

With any machine learning project, you need to think about the incoming data, what format it's in, and how it will be accessed by the code that's being built.

Data comes in all sorts of forms, so it's a good idea to know what you're dealing with before you start crafting any code. The following sections describe some of the more common data formats.

Raw Text

Basic raw text files are used in many publications. If you look at the likes of the Guttenberg Project, you'll see that you can download works in a raw text file. The data is unstructured, so it rarely has a proper form with which you can work.

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse
eget metus quis erat tempor hendrerit. Vestibulum turpis ante, bibendum
vitae nisi non, euismod blandit dui. Maecenas tristique consectetur est
nec elementum. Maecenas porttitor, arcu sed gravida tempus, purus tellus
lacinia erat, dapibus euismod felis enim eget nisl. Nunc mollis volutpat
ligula. Etiam interdum porttitor nulla non lobortis.
```

Common formats for text files are Unicode, ASCII, or UTF-8. If there's any international encoding required, UTF-8 or Unicode are most common. Note that PDF documents, Rich Text Format files, and Word documents are not raw text files. Microsoft Office documents (such as Word files) are particularly troublesome because of "smart quotes" and other non-text extraneous characters that wreak havoc in Java programs.

Comma Separated Variables

The CSV format is widely used across the data landscape. The comma character is used between each field of data. You might find that other delimiters are used, such as tabulation (TSV) and the pipe (|) symbol (PSV). Delimiters are not limited to one character either. If you look at something like the USDA Food Database you'll see ~^~ used as a delimiter. The following CSV file is generated from a fake name generator site. (It's always good to use fake data when you're testing things.)

```
1,male,Mr.,Joe,L,Perry,50 Park Row,EDERN,,LL53 2SQ,GB,United
Kingdom,JoePerry@einrot.com,Annever,eiThahph9Ah,077 6473 7650,Fry,
7/4/1991,Visa,4539148712302735,342,2/2018,YB 20 98 60 A,1Z 23F 389
```

```
61 4167 727 1,Blue,Nephrology nurse,Friendly Advice,1999 Alfa Romeo
145,BadProtection.co.uk,O+,169.4,77.0,5' 10",177,a617f840-6e42-4146-
b743-090ee59c2c9f,52.806493,-4.72918

2,male,Mr.,Daniel,J,Carpenter,51 Guildford Rd,EAST
DRAYTON,,DN22 3GT,GB,United Kingdom,DanielCarpenter@teleworm.
us,Reste1990,Eich1Kiegie,079 2890 2948,Harris,3/26/1990,MasterCard
,5353722386063326,717,7/2018,KL 50 03 59 C,1Z 895 362 50 0377 620
2,Blue,Corporate administrative assistant,Hit or Miss,2000 Jeep Grand
Cherokee,BiologyConvention.co.uk,AB+,175.3,79.7,5' 7",169,ac907a59-a091-
4ba2-9b0f-a1276b3b5ada,52.801024,-0.719021

3,male,Mr.,Harvey,A,Hawkins,37 Shore Street,STOKE TALMAGE,,OX9
4FY,GB,United Kingdom,HarveyHawkins@armyspy.com,Spicionly,UcheeGh9xoh,077
7965 0825,Rees,3/1/1974,MasterCard,5131613608666799,523,7/2017,SS 81 32
33 C,1Z Y11 884 19 7792 722 8,Black,Education planner,Monsource,1999 BMW
740,LightingShadows.co.uk,A-,224.8,102.2,6' 1",185,6cf865fb-81ae-42af-
9a9d-5b86d5da7ce9,51.573674,-1.179834

4,male,Mr.,Kyle,E,Patel,97 Cloch Rd,ST MARTIN,,TR12 6LT,GB,United
Kingdom,KylePatel@superrito.com,Wilvear,de2EeJew,079 2879 6351,Hancoc
k,6/7/1978,Visa,4916480323599950,960,4/2016,MH 93 02 76 D,1Z 590 692
15 4564 674 8,Blue,Interior decorator,Grade A Investment,2002 Proton
Juara,ConsumerMenu.co.uk,AB+,189.2,86.0,5' 10",179,e977c58e-ba61-406e-
a1d1-2904807be365,49.957435,-5.258628

5,male,Mr.,Dylan,A,Willis,66 Temple Way,WINWICK,,WA2 5HE,GB,United
Kingdom,DylanWillis@cuvox.de,Hishound,shael7Foo,077 1105 4178,Kelly,
8/16/1948,Visa,4485311140499796,423,11/2016,WG 24 10 62 D,1Z 538 4E0
39 8247 102 7,Black,Community health educator,Mr. Steak,2002 Nissan
X-Trail,FakeRomance.co.uk,A+,170.1,77.3,5' 9",175,335c2508-71be-43ad-
9760-4f5c186ec029,53.443749,-2.631634

6,female,Mrs.,Courtney,R,Jordan,42 Kendell Street,SHARLSTON,,WF4
1PZ,GB,United Kingdom,CourtneyJordan@fleckens.
hu,Ponforsittle,Hi2oteel1,070 3469 5710,Payne,2/23/1982,MasterCa
rd,5570815007804057,456,12/2019,CJ 87 95 98 D,1Z 853 489 84 8609
859 3,Blue,Mechanical inspector,Olson Electronics,2000 Chrysler
LHS,LandscapeCovers.co.uk,B+,143.9,65.4,5' 3",161,27d229b0-6106-4700-
8533-5edc2661a0bf,53.645118,-1.563952
```

People might refer to files as CSV files even though they are not comma separated. The best way to find out if something is really a CSV file is to open up the data and have a look.

JSON

JavaScript Object Notation (JSON) is a commonly used data format that utilizes key/value pairs to communicate data between machines and the web. It

was designed as an alternative to XML. Don't be fooled by the use of the word JavaScript; you don't need JavaScript in order to use this data format. There are JSON parsers for various languages. The earlier CSV example used fake name data; here's the first entry of the CSV in JSON notation:

```
[
  {
    "Number":1,
    "Gender":"male",
    "Title":"Mr.",
    "GivenName":"Joe",
    "MiddleInitial":"L",
    "Surname":"Perry",
    "StreetAddress":"50 Park Row",
    "City":"EDERN",
    "State":"",
    "ZipCode":"LL53 2SQ",
    "Country":"GB",
    "CountryFull":"United Kingdom",
    "EmailAddress":"JoePerry@einrot.com",
    "Username":"Annever",
    "Password":"eiThahph9Ah",
    "TelephoneNumber":"077 6473 7650",
    "MothersMaiden":"Fry",
    "Birthday":"7/4/1991",
    "CCType":"Visa",
    "CCNumber":4539148712302735,
    "CVV2":342,
    "CCExpires":"2/2018",
    "NationalID":"YB 20 98 60 A",
    "UPS":"1Z 23F 389 61 4167 727 1",
    "Color":"Blue",
    "Occupation":"Nephrology nurse",
    "Company":"Friendly Advice",
    "Vehicle":"1999 Alfa Romeo 145",
    "Domain":"BadProtection.co.uk",
    "BloodType":"O+",
    "Pounds":169.4,
    "Kilograms":77.0,
    "FeetInches":"5' 10\"",
    "Centimeters":177,
    "GUID":"a617f840-6e42-4146-b743-090ee59c2c9f",
    "Latitude":52.806493,
    "Longitude":-4.72918
  }
]
```

Many application-programming interfaces (APIs) use JSON to send response data back to the requesting program. Some parsers might take the JSON data

and represent it as an object. Others might be able to create a hash map of the data for you to access.

YAML

Whereas JSON is a document markup format, YAML (meaning "YAML Ain't Markup Language") is most certainly a data format. It's not as widely used as JSON but from a distance looks very similar.

```
date   : 2014-01-02
bill-to: &id001
    given  : Jason
    family : Bell
    address:
        lines: |
            458 Some Street Somewhere
            In Some Suburb
        city   : MyCity
        state  : CA
        postal : 55555
```

XML

The extensible markup language (XML) followed on from the popular use of Standard Generalized Markup Language (SGML) for document markup. The idea was for XML to be easily read by humans and also by machines. On first inspection, XML is like Hypertext Markup Language (HTML); later versions of HTML use strict XML formatting types.

XML gets criticism for its complexity, especially when reading large structures. That's one reason it's popular for web-based APIs to use JSON data as its response. There are a large number of APIs delivering XML response data, so it's worthwhile to look at how it works:

```
<?xml version="1.0" encoding="UTF-8" ?>
    <Customer>
        <Number>1</Number>
        <Gender>male</Gender>
        <Title>Mr.</Title>
        <GivenName>Joe</GivenName>
        <MiddleInitial>L</MiddleInitial>
        <Surname>Perry</Surname>
        <StreetAddress>50 Park Row</StreetAddress>
        <City>EDERN</City>
        <State></State>
        <ZipCode>LL53 2SQ</ZipCode>
        <Country>GB</Country>
        <CountryFull>United Kingdom</CountryFull>
```

```
                <EmailAddress>JoePerry@einrot.com</EmailAddress>
                <Username>Annever</Username>
                <Password>eiThahph9Ah</Password>
                <TelephoneNumber>077 6473 7650</TelephoneNumber>
                <MothersMaiden>Fry</MothersMaiden>
                <Birthday>7/4/1991</Birthday>
                <CCType>Visa</CCType>
                <CCNumber>4539148712302735</CCNumber>
                <CVV2>342</CVV2>
                <CCExpires>2/2018</CCExpires>
                <NationalID>YB 20 98 60 A</NationalID>
                <UPS>1Z 23F 389 61 4167 727 1</UPS>
                <Color>Blue</Color>
                <Occupation>Nephrology nurse</Occupation>
                <Company>Friendly Advice</Company>
                <Vehicle>1999 Alfa Romeo 145</Vehicle>
                <Domain>BadProtection.co.uk</Domain>
                <BloodType>O+</BloodType>
                <Pounds>169.4</Pounds>
                <Kilograms>77</Kilograms>
                <FeetInches>5' 10&quot;</FeetInches>
                <Centimeters>177</Centimeters>
                <GUID>a617f840-6e42-4146-b743-090ee59c2c9f</GUID>
                <Latitude>52.806493</Latitude>
                <Longitude>-4.72918</Longitude>
        </Customer>
```

Most of the common languages have XML parsers available using either a document object model (DOM) parser or the Simple API for XML (SAX) parser. Both types come with advantages and disadvantages depending on the size and complexity of the XML document with which you are working.

Spreadsheets

Talk to any finance person in your organization, and you'll discover that their entire world revolves around spreadsheets. Programmers have a tendency to shun spreadsheets in favor of data formats that make their lives easier. You can't totally ignore them, though. Spreadsheets are the lifeblood of an organization, and they probably hold most of the organization's data.

There are lots of different spreadsheet programs, but the most commonly used applications are Microsoft Excel, Google Docs Spreadsheet, and LibreOffice.

Fortunately there are programming APIs that you can use to extract the data from spreadsheets directly, which saves a lot of work in converting

the spreadsheet to the likes of CSV files. It's worth studying the formulas in the spreadsheets, because there might be some algorithms lurking there that are worth their weight in gold.

If you want your finance person to be supportive of the project, tell that person that the results will be in a spreadsheet and you'll have a friend for a long time after.

The Java programming language has a few APIs to choose from that will enable you to read and write spreadsheets. The Apache POI project and JExcel API are the two most popular.

Databases

If you've been brought up with web programming, then you might have had some exposure to databases and database tables. Common ones are MySQL, Postgres, Microsoft SQL Server, and Oracle.

Recently, there's been an explosion of NoSQL (meaning Not Only SQL), such as MongoDB, CouchDB, Cassandra, Redis, and HBase, which all bring their own flavors to data storage. These document and key/value stores move away from the rigid table-like structures of traditional databases.

In addition, there are graph databases such as Apache Giraph and Neo4J and in-memory systems such as Spark, memcached, and Storm. Chapter 11 is an introduction to Spark.

In my opinion, all databases have their place and are worth investigating. There's nothing wrong with having a relational, document and graph database running concurrently for the project. Each has its advantages to the project that you might not have considered. As with all these things, there might be a learning curve that you need to factor in to your project time.

Images

The common data formats previously mentioned mainly deal with text or numbers in different shades, but you can't discount images. There are a number of things you can learn from images. Whether you're trying to use facial recognition or emotion tracking or you're trying to determine whether an image is a cat or dog (yes, it has been done), there are several APIs that will help.

The most popular formats are the portable network graphics (PNG) and JPEG images; these are regularly used on the web. If processing power is freely available then TIFF or BMP are much larger files, but they contain more image information.

Thinking about Output Data

Now it's time to turn your attention to the output data. This is where the stakeholders might have a say in how things are going to be done, because ultimately it will be those people who deal with the results.

The primary question about the output of machine learning data is, "Who is the intended audience?" Depending on the answer to that question, your output will vary. You might need a spreadsheet for the financial folks to see the results. If the audience is comprised of website users, then it makes sense to put the data back into a database table. The machine learning results could be merged with other data to define more learning. It really comes down to what was defined in the project.

There are a number of paid and free reporting tools available. Some are full-blown systems, such as Jasper Reports, BIRT, and Tableau. If you are reporting to a web-based audience, then the likes of D3 and Processing might be of help to you.

Don't Be Afraid to Experiment

It's safe to say that there is no "one solution fits all." There are many components, formats, tools, and considerations to ponder on any project. In effect, every machine learning project starts with a clean sheet, and communication among all involved, from stakeholders all the way through to visualization. Tools and scripts can be reused, but every case is going to be different, so things need minor adjustments as you go along. Don't be afraid to play around with data as you acquire it; see if there's anything you can glean from it.

It's also worth taking time to grab some open data and make your own scenarios and ask your own questions. It's like a musician practicing an instrument; it's worth putting in the hours, so you are ready for the day when the big gig arrives.

The machine learning community is large, and there are plenty of blog posts, articles, videos, and books produced by the community. Forums are the perfect place to swap stories and experiences, too. As with most things, the more you put in, the more you will get out of it.

Over the years, I've found that people are more than willing to help contribute to a solution if you're stuck on a problem . If you've not looked at the likes of `http://stackoverflow.com`, a collaborative question and answer platform for software developers, then have a search around. Chances are that someone will have encountered the same problem as you.

Summary

As with any project, planning is a key and essential part of machine learning and shouldn't be taken lightly. This chapter covered many aspects of planning, including processing, storage, privacy, and data cleaning. You were also introduced to some useful tools and commands that will help in the cleaning phases and some validation checks.

The planning phase is a constantly evolving process, and the more machine learning projects you and the team perform, the more you will learn from previous mistakes.

The key is to start small. Take a snapshot of the data and take a random sample with a size of 10 percent of the total. Get the team to inspect the data. Can you work with it? Do you anticipate any problems with the processing of this data?

Cleaning the data might take the most time of the project; the actual processing might consume only a fraction of the overall project time. If you can supply clean data, then your results will be refined.

Regardless of whether you are working on a ten-man team or on your own, be aware of your network of contacts; some might have domain knowledge that will be useful. Ask lots of questions, too. You'd be surprised how many folks are willing to answer questions in order to see you succeed.

The next few chapters examine some different machine learning techniques and put some sample code together, so you can start to apply them to your own projects.

Working with Decision Trees

Do not be deceived by the decision tree; at first glance it might look like a simple concept, but within the simplicity lies the power. This chapter shows you how decision trees work. The examples use Weka to create a working decision tree that will also create the Java code for you.

The Basics of Decision Trees

The aim with any decision tree is to create a workable model that will predict the value of a target variable based on the set of input variables. This section explains where decision trees are used along with some of the advantages and limitations of decision trees. In this section you also find out how a decision tree is calculated manually so you can see the math involved.

Uses for Decision Trees

Think about how you select different options within an automated telephone call. The options are essentially decisions that are being made for you to get to the desired department. These decision trees are used effectively in many industry areas.

Financial institutions use decision trees. One of the fundamental use cases is in option pricing, where a binary-like decision tree is used to predict the price of an option in either a bull or bear market.

Marketers use decision trees to establish customers by type and predict whether a customer will buy a specific type of product.

In the medical field, decision tree models have been designed to diagnose blood infections or even predict heart attack outcomes in chest pain patients. Variables in the decision tree include diagnosis, treatment, and patient data.

The gaming industry now uses multiple decision trees in movement recognition and facial recognition. The Microsoft Kinect platform uses this method to track body movement. The Kinect team used one million images and trained three trees. Within one day, and using a 1,000-core cluster, the decision trees were classifying specific body parts across the screen.

Advantages of Decision Trees

There are some good reasons to use decision trees. For one thing, they are easy to read. After a model is generated, it's easy to report back to others regarding how the tree works. Also, with decision trees you can handle numerical or categorized information. Later, this chapter demonstrates how to manually work through an algorithm with category values; the example walkthrough uses numerical data.

In terms of data preparation, there's little to do. As long as the data is formalized in something like comma separated variables, then you can create a working model. This also makes it easy to validate the model using various tests. With decision trees you use white-box testing—meaning the internal workings can be observed but not changed; you can view the steps that are being used when the tree is being modeled.

Decision trees perform well with reasonable amounts of computing power. If you have a large set of data, then decision tree learning will handle it well.

Limitations of Decision Trees

With every set of advantages there's usually a set of disadvantages sitting in the background. One of the main issues of decision trees is that they can create overly complex models, depending on the data presented in the training set. To avoid the machine learning algorithm's over-fitting the data, it's sometimes worth reviewing the training data and pruning the values to categories, which will produce a more refined and better-tuned model.

Some of the decision tree concepts can be hard to learn because the model cannot express them easily. This shortcoming sometimes results in a larger-than-normal

model. You might be required to change the model or look at different methods of machine learning.

Different Algorithm Types

Over the years, there have been various algorithms developed for decision tree analysis. Some of the more common ones are listed here.

ID3

The *ID3* (Iterative Dichotomiser 3) algorithm was invented by Ross Quinlan to create trees from datasets. By calculating the entropy for every attribute in the dataset, this could be split into subsets based on the minimum entropy value. After the set had a decision tree node created, all that was required was to recursively go through the remaining attributes in the set.

ID3 uses the method of information gain—the measure of difference in entropy before and after an attribute is split—to decide on the root node (the node with the highest information gain).

ID3 suffered from over-fitting on training data, and the algorithm was better suited to smaller trees than large ones. The ID3 algorithm is used less these days in favor of the C4.5 algorithm, which is outlined next.

C4.5

Quinlan came back for an encore with the C4.5 algorithm. It's also based on the information gain method, but it enables the trees to be used for classification. This is a widely used algorithm in that many users run in Weka with the open source Java version of C4.5, the J48 algorithm.

There are notable improvements in C4.5 over the original ID3 algorithm. With the ability to work on continuous attributes, the C4.5 method will calculate a threshold point for the split to occur. For example, with a list of values like the following:

85,80,83,70,68,65,64,72,69,75,75,72,81,71

C4.5 will work out a split point for the attribute (a) and give a simple decision criterion of:

a <= 80 or a > 80

C4.5 has the ability to work despite missing attribute values. The missing values are marked with a question mark (?). The gain and entropy calculations are simply skipped when there is no data available.

Trees created with C4.5 are pruned after creation; the algorithm will revisit the nodes and decide if a node is contributing to the result in the tree. If it isn't, then it's replaced with a leaf node.

CHAID

The *CHAID* (*Chi-squared Automatic Interaction Detection*) technique was developed by Gordon V. Kass in 1980. The main use of it was within marketing, but it was also used within medical and psychiatric research.

MARS

For numerical data, it might be worth investigating the *MARS* (*multivariate adaptive regression splines*) algorithm. You might see this as an open source alternative called "Earth," as MARS is trademarked by Salford Systems.

How Decision Trees Work

Every tree is comprised of nodes. Each node is associated with one of the input variables. The edges coming from that node are the total possible values of that node. A leaf represents the value based on the values given from the input variable in the path running from the root node to the leaf. Because a picture paints a thousand words, see Figure 3-1 for an example.
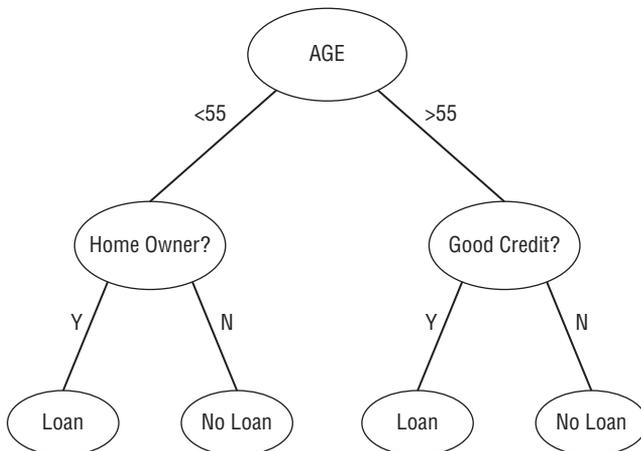
Figure 3-1: A decision tree

Decision trees always start with a root node and end on a leaf. Notice that the trees don't converge at any point; they split their way out as the nodes are processed.

Figure 3-1 shows a decision tree that classifies a loan decision. The root node is "Age" and has two branches that come from it, whether the customer is younger than 55 years old or older.

The age of the client determines what happens next. If the person is younger than 55, then the tree prompts you to find out if he or she is a student. If the client is older than 55 then you are prompted to check his or her credit rating.

With this type of machine learning, you are using supervised learning to deduce the optimal method to make a prediction; what I mean by "supervised learning" is that you give the classifier data with the outcomes. The real question is, "What's the best node to start with as the root node?" The next section examines how that calculation is done.

Building a Decision Tree

Decision trees are built around the basic concept of this algorithm.

- Check the model for the base cases.
- Iterate through all the attributes (`attr`).
- Get the normalized information gain from splitting on `attr`.
- Let `best_attr` be the attribute with the highest information gain.
- Create a decision node that splits on the `best_attr` attribute.
- Work on the sublists that are obtained by splitting on `best_attr` and add those nodes as child nodes.

That's the basic outline of what happens when you build a decision tree. Depending on the algorithm type, like the ones previously mentioned, there might be subtle differences in the way things are done.

Manually Walking Through an Example

If you are interested in the basic mechanics of how the algorithm works and want to follow along, this section walks through the basics of calculating entropy and information gain. If you want to get to the hands-on part of the chapter, then you can skip this section.

The method of using information gain based on pre- and post-attribute entropy is the key method used within the ID3 and C4.5 algorithms. As these are the commonly used algorithms, this section concentrates on that basic method of finding out how the decision tree is built.

With machine learning–based decision trees, you can get the algorithm to do all the work for you. It will figure out which is the best node to use as the root node. This requires finding out the purity of each node. Consider Table

3-1, which includes only true/false values, of some user purchases through an e-commerce store.

Table 3-1: Users' Purchase History

	HAS CREDIT ACCOUNT?	READ REVIEWS	PREVIOUS CUSTOMER?	DID PURCHASE?
User A	N	Y	Y	Y
User B	Y	Y	Y	Y
User C	N	N	Y	N
User D	Y	N	N	Y
User E	Y	Y	Y	Y

There are four nodes in the table:

- Does the customer have an account?
- Did the customer read previous product reviews?
- Is the customer a returning customer?
- Did the customer purchase the product?

At the start of calculating the decision tree there is no awareness of the node that will give the best result. You're looking for the node that can best predict the outcome. This requires some calculation. Enter entropy.

Calculating Entropy

Entropy is a measure of uncertainty and is measured in bits and comes as a number between zero and 1 (entropy bits are not the same bits as used in computing terminology). Basically, you are looking for the unpredictability in a random variable.

You need to calculate the gain for the positive and negative cases. I've written a quick Java program to do the calculating:

```java
package chapter3;

public class InformationGain {

    private double calcLog2(double value) {
        if(value <= 0.) {
            return 0.;
        }
        return Math.log10(value) / Math.log10(2.);
```

```
    }

    public double calcGain(double positive, double negative) {
        double sum = positive + negative;
        double gain = positive * calcLog2(positive/sum)/sum + negative *
calcLog2(negative/sum)/sum;
        return -gain;
    }

    public static void main(String[] args) {
        InformationGain ig = new InformationGain();
        System.out.println(ig.calcGain(2, 3));
    }

}
```

Looking back at the table of customers with credit accounts there are three with and two without. So calculating the gain with these variables you get the following result:

$$\text{Gain}(3,2) \qquad = (3/5)*\log2(3/5) + (2/5)*\log2(2/5)$$
$$= 0.97$$

log2() refers to the calculation in the `calcLog2()` method in the code snippet. If you don't want to type or compile the code listing, then try copying and pasting the gain equation into `www.wolframalpha.com` and you'll see the answer there. The outcomes of the variables in the `reads reviews` attribute linking back to the `accounts` attribute are the following:

Reads reviews = [Y, Y, N]
Does not read reviews = [N, Y]

You can now calculate the entropy with the split based on the first attribute:

$$\text{Gain}(2,1) \qquad = (2/3)*\log2(2/3) + (1/3)*\log2(1/3)$$
$$= 0.91$$
$$\text{Gain}(1,1) \qquad = (1/2)*\log2(1/2) + (1/2)*\log2(1/2)$$
$$= 1$$

The net gain is finally calculated:

Net gain(attribute = has credit account)
$$= (2/5) * 0.91 + (3/5) * 1$$
$$= 0.96$$

So, you have two gains: one before the split (0.97) and one after the split (0.96).

You're nearly done on this attribute. You just have to calculate the information gain.

Information Gain

When you know the gain before and after the split in the attribute, you can calculate the information gain. With the attribute to see if the customer has a credit account, your calculation will be the following:

$$InformationGain = Gain(\text{before the split}) - Gain(\text{after the split})$$
$$= 0.97 - 0.96$$
$$= 0.01$$

So, the information gain on the `has credit account` attribute is 0.01.

Rinse and Repeat

The previous two sections covered the calculation of information gain for one attribute, `Has Credit Account`. You need to work on the other two attributes to find their information gain.

Reads Reviews:
Gain(3,2) = 0.97
Net Gain = 0.4
Information Gain = 0.57

Previous Customer:
Gain(4,1) = 0.72
Net Gain = 0.486
Information Gain = 0.234

With the values of information gain for all the attributes, you can now make a decision on which node to start with in the tree.

ATTRIBUTE	INFORMATION GAIN
Has Credit Account	0.01
Reads Reviews	0.57
Is Previous Customer	0.234

Now things are becoming clearer; the `Reads Reviews` attribute has the highest information gain and therefore should be the root node in the tree, then comes the Is Previous Customer node followed by Has Credit Account.

The order of information gain determines where the node will appear in the decision tree model. The node with the highest gain becomes the root node.

That's enough of the basic theory of how decision trees work. The best way to learn is to get something working, which is described in the next section.

Decision Trees in Weka

In this section, you'll use the Weka data-mining tool to work through some training data of the optimum sales of Lady Gaga's CDs depending on specific factors within the store. I explain the factors in question as you walk though that data.

The Requirement

The requirement is to create a model that will be able to predict a customer sale on Lady Gaga CDs depending on the CDs' placement within the store. You've been given some data by the record store about where the product was placed, whether it was at eye level or not, and whether the customer actually purchased the CD or put it back on the shelf.

The client wants to be able to run other sets of data through the model to determine how sales of a product will fare.

Working through this methodically, you need to do the following:

1. Run through the training data supplied and turn it into a definition file for Weka.
2. Use the Weka workbench to build the decision tree for you and plot an output graph.
3. Export some generated Java code with the new decision tree classifier.
4. Test the code against some test data.
5. Think about future iterations of the classifier.

It feels like there's a lot to do, but after you get into the routine, it's quite simple to do with the tools at hand. First look at the training data.

Training Data

Before anything else happens, you need some training data. The client has given you some in a `.csv` file, but it would be nice to formalize this. This is what you received:

```
Placement,prominence, pricing, eye_level, customer_purchase
end_rack,85,85,FALSE,yes
end_rack,80,90,TRUE,yes
cd_spec,83,86,FALSE,no
std_rack,70,96,FALSE,no
std_rack,68,80,FALSE,no
```

```
std_rack,65,70,TRUE,yes
cd_spec,64,65,TRUE,yes
end_rack,72,95,FALSE,yes
end_rack,69,70,FALSE,yes
std_rack,75,80,FALSE,no
end_rack,75,70,TRUE,no
cd_spec,72,90,TRUE,no
cd_spec,81,75,FALSE,yes
std_rack,71,91,TRUE,yes
```

Weka saves the file as a `.arff` file to set up the attributes and let you give it some data from which to train. The `.arff` file is a text file that outlines the data model you are going to use:

```
@relation ladygaga

@attribute placement {end_rack, cd_spec, std_rack}
@attribute prominence numeric
@attribute pricing numeric
@attribute eye_level {TRUE, FALSE}
@attribute customer_purchase {yes, no}

@data
end_rack,85,85,FALSE,yes
end_rack,80,90,TRUE,yes
cd_spec,83,86,FALSE,no
std_rack,70,96,FALSE,no
std_rack,68,80,FALSE,no
std_rack,65,70,TRUE,yes
cd_spec,64,65,TRUE,yes
end_rack,72,95,FALSE,yes
end_rack,69,70,FALSE,no
std_rack,75,80,FALSE,no
end_rack,75,70,TRUE,no
cd_spec,72,90,TRUE,no
cd_spec,81,75,FALSE,yes
std_rack,71,91,TRUE,yes
```

The data file has a few elements to it, so let's look through it one section at a time.

Relation

The `@relation` tag is the name of the dataset you are using. In this instance it's Lady Gaga's CDs, so I've called it `ladygaga`.

Attributes

Next, you have the attributes that are used within your data model. There are five attributes in this set that are the top line of raw CSV data that you received from the client.

- **Placement:** What type of stand the CD is displayed on: an end rack, special offer bucket, or a standard rack?
- **Prominence:** What percentage of the CDs on display are Lady Gaga CDs?
- **Pricing:** What percentage of the full price was the CD at the time of purchase? Very rarely is a CD sold at full price, unless it is an old, back catalog title.
- **Eye Level:** Was the product displayed at eye level position? The majority of sales will happen when a product is displayed at eye level.
- **Customer Purchase:** What was the outcome? Did the customer purchase?

The Prominence and Pricing attributes are both numeric values. The other three are given the nominal values that are to be expected when the algorithm is being run. Placement has three: `end_rack`, `cd_spec`, or `std_rack`. The Eye Level attribute is either true or false, and the Customer Purchase attribute has two nominal values of either yes or no to show that the customer bought the product.

Data

Finally, you have the data. It's comma separated in the order of the attributes (Placement, Prominence, Pricing, Eye Level, and Customer Purchase). In this sample, you know the outcomes—whether a customer purchased or not; this model is about using regression to get your predictions in tune for new data coming in.

You can find all the code for this chapter on the book's companion website at `www.wiley.com/go/machinelearning`.

Using Weka to Create a Decision Tree

Now that you have your data model in place, you can get started. When you open the Weka program you are presented with a small opening screen (see Figure 3-2) with four buttons: Explorer, Experimenter, KnowledgeFlow, and Simple CLI. Click the Explorer button.

Figure 3-2: The Weka GUI Chooser

When the Explorer opens, you will be confronted with another window with a number of sections and an array of buttons (see Figure 3-3). Don't worry if it all looks confusing right now; this walkthrough takes you through it step by step.
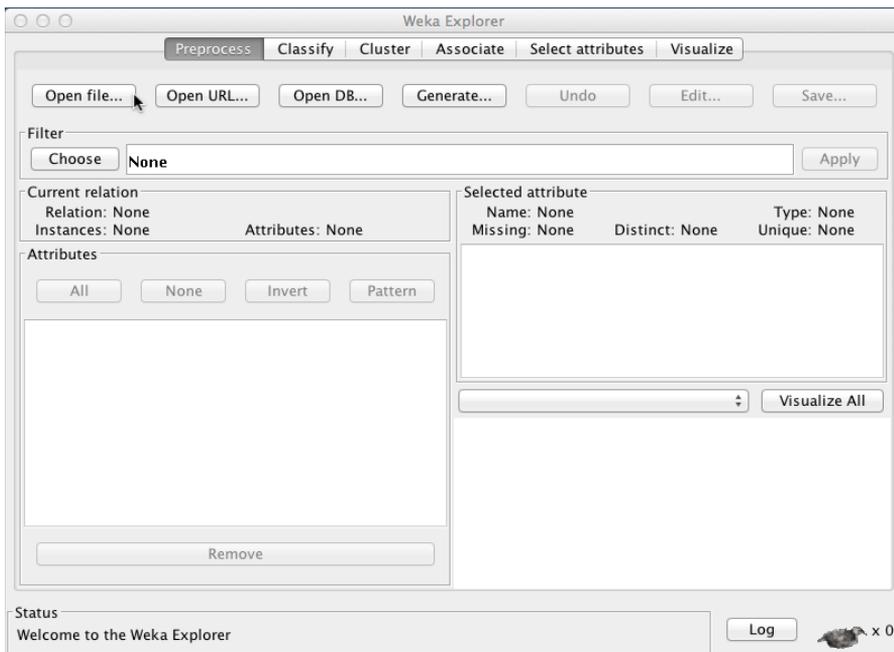
Figure 3-3: The basic Explorer window

Click the Open File button and select the data file called `ladygaga.arff`. Weka parses the data model and preprocesses the data. Within no time you're already getting information based on the preprocessing of the data model and the data.

The Select Attribute pane on the right side of the Explorer window in Figure 3-4 shows the three distinct nominal values of the `customer_purchase` attribute. Weka has also noticed that you have 14 instance rows and the five attributes.

After preprocessing comes classification. Click the Classify button in the top row of buttons. You're going to use the C4.5 classification algorithm; within Weka this is called the J48 algorithm. In the Classifier pane (see Figure 3-5), click the Choose button and select the J48 option under the Trees menu heading. The selection pane closes automatically, and you see that the name of the classifier has changed from the default `ZeroR` to `J48 -C 0.25 -M 2`. (See Figure 3-5.)
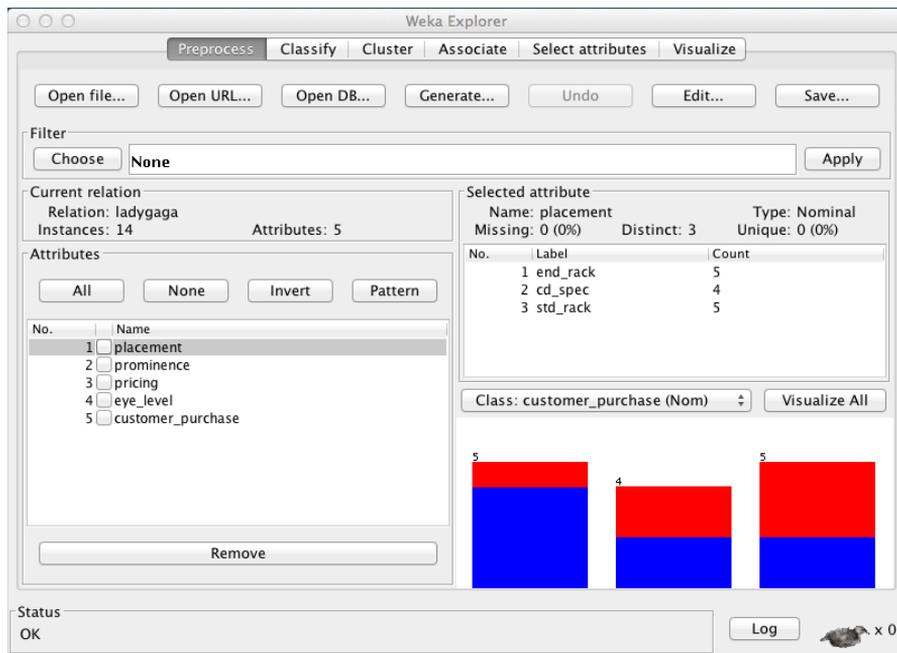
Figure 3-4: The preprocess pane with data
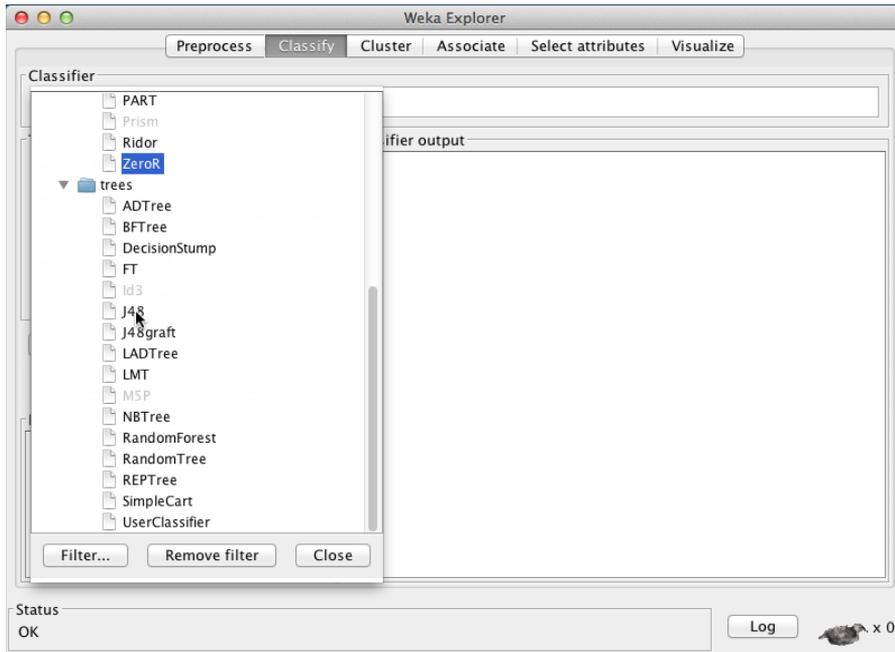
Figure 3-5: Selecting the classifier

The option flags used in the default J48 classifier are setting the pruning confidence (the –C flag) and the minimum number of instances (-M).

To run the classifier, click the Start button and watch the Classifier output window (see Figure 3-6). You see the information on the run appear. The run information tells you about the scheme used and gives a run-down on the model on which Weka has worked.

Interesting data starts to emerge. The J48 pruned tree gives results on, in this case, the placement, as it has the highest information gain:

```
J48 pruned tree
------------------

placement = end_rack: yes (5.0/1.0)
placement = cd_spec
|   pricing <= 80: yes (2.0)
|   pricing > 80: no (2.0)
placement = std_rack
|   eye_level = TRUE: yes (2.0)
```

```
|    eye_level = FALSE: no (3.0)

Number of Leaves  :        5

Size of the tree :   8


Time taken to build model: 0 seconds
```
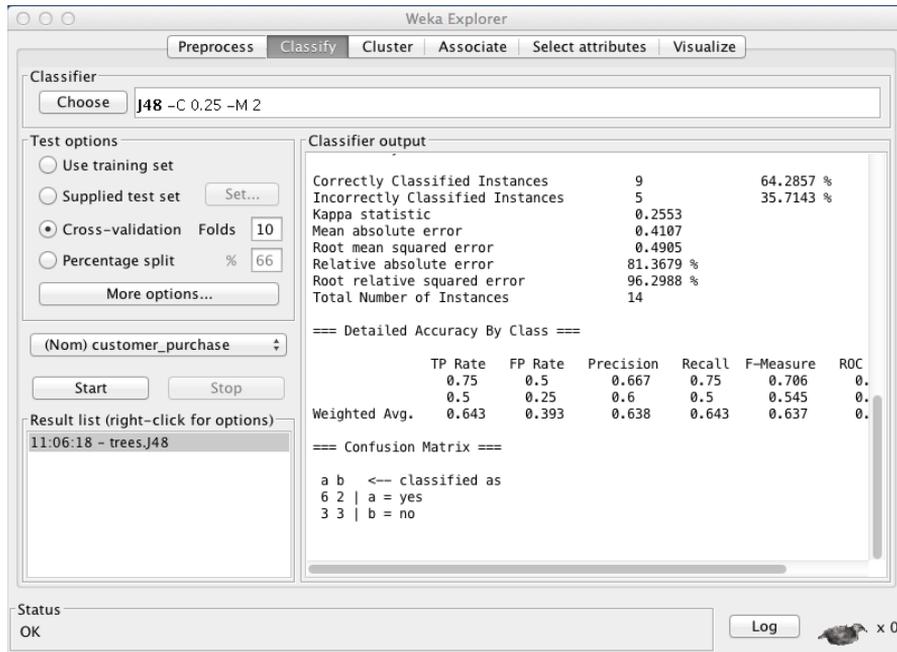

Figure 3-6: Classifier with output

It appears that placing product on the end rack is good for sales. For the special offer rack, it seems that pricing plays a part; if the product is too cheap customers walk away. On the standard racks, the placement of the product is a factor for sales; it sells if it's at eye level.

Finally you want to plot the visualization of the tree for the management team to look at because pictures speak louder than words. On the Results List pane on the bottom left of the Explorer window you can see the time and algorithm that was run. Right-click (use Alt + click if you are using an OSX machine) and select the Visualize Tree option to see the tree in its visual representation, as shown in Figure 3-7.
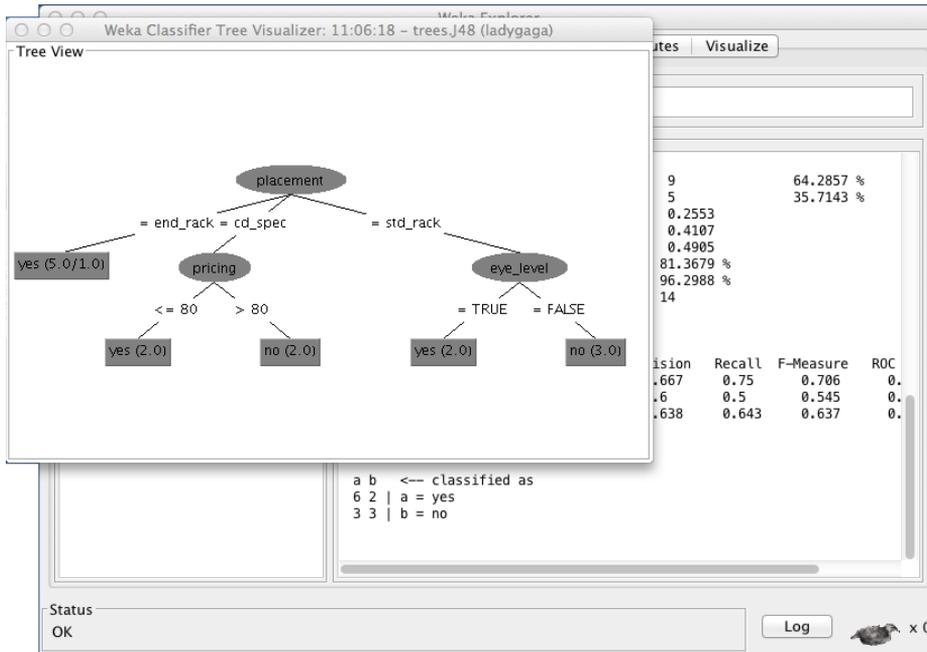
Figure 3-7: J48 visualization

It's usually at this point where everyone pats each other on the back and says, "Job well done," but you're not finished yet. You don't want to have to run the Weka Explorer every time you have data to run. What you want is some code that you can reuse.

Creating Java Code from the Classification

As mentioned in Chapter 2, there is no one tool that really fits all. Weka is excellent, but you want code that you can safely run in an existing codebase. Perhaps you want to hook your newly created classification to a Hadoop job, if the incoming volume of data was sufficient to do so.

With the existing classifier, click the More Options button and a new window opens with the options for the current evaluator. (See Figure 3-8.)

The last option is to output to source code. By default, the class name will be WekaClassifier. It won't save your Java code, but it will output in the Classifier output window.
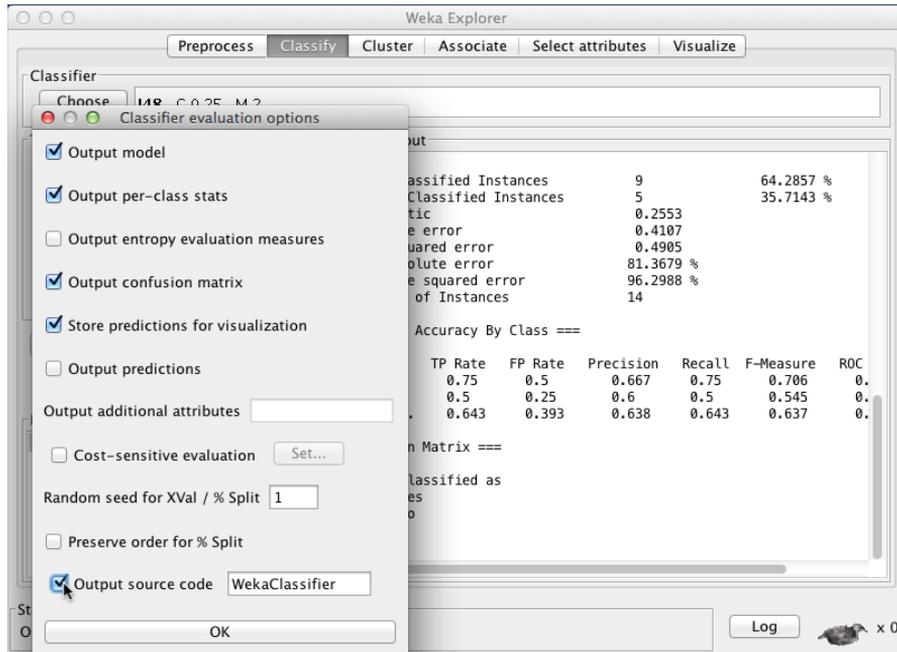
Figure 3-8: Evaluation options pane

Start the classifier again, and in the output window you see the Java code at the end of the output information:

```
package weka.classifiers;

import weka.core.Attribute;
import weka.core.Capabilities;
import weka.core.Capabilities.Capability;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.RevisionUtils;
import weka.classifiers.Classifier;

public class WekaWrapper
  extends Classifier {

  /**
   * Returns only the toString() method.
   *
   * @return a string describing the classifier
   */
  public String globalInfo() {
    return toString();
```

```
    }

    /**
     * Returns the capabilities of this classifier.
     *
     * @return the capabilities
     */
    public Capabilities getCapabilities() {
      weka.core.Capabilities result = new weka.core.Capabilities(this);

      result.enable(weka.core.Capabilities.Capability.NOMINAL_ATTRIBUTES);
      result.enable(weka.core.Capabilities.Capability.NUMERIC_ATTRIBUTES);
      result.enable(weka.core.Capabilities.Capability.DATE_ATTRIBUTES);
      result.enable(weka.core.Capabilities.Capability.MISSING_VALUES);
      result.enable(weka.core.Capabilities.Capability.NOMINAL_CLASS);
      result.enable(weka.core.Capabilities.Capability.MISSING_CLASS_
VALUES);

      result.setMinimumNumberInstances(0);

      return result;
    }

    /**
     * only checks the data against its capabilities.
     *
     * @param i the training data
     */
    public void buildClassifier(Instances i) throws Exception {
      // can classifier handle the data?
      getCapabilities().testWithFail(i);
    }

    /**
     * Classifies the given instance.
     *
     * @param i the instance to classify
     * @return the classification result
     */
    public double classifyInstance(Instance i) throws Exception {
      Object[] s = new Object[i.numAttributes()];

      for (int j = 0; j < s.length; j++) {
        if (!i.isMissing(j)) {
          if (i.attribute(j).isNominal())
            s[j] = new String(i.stringValue(j));
          else if (i.attribute(j).isNumeric())
            s[j] = new Double(i.value(j));
        }
      }
```

```java
    // set class value to missing
    s[i.classIndex()] = null;

    return WekaClassifier.classify(s);
  }

  /**
   * Returns the revision string.
   *
   * @return        the revision
   */
  public String getRevision() {
    return RevisionUtils.extract("1.0");
  }

  /**
   * Returns only the classnames and what classifier it is based on.
   *
   * @return a short description
   */
  public String toString() {
    return "Auto-generated classifier wrapper, based on weka.
classifiers.trees.J48 (generated with Weka 3.6.10).\n" + this.
getClass().getName() + "/WekaClassifier";
  }

  /**
   * Runs the classfier from commandline.
   *
   * @param args the commandline arguments
   */
  public static void main(String args[]) {
    runClassifier(new WekaWrapper(), args);
  }
}

class WekaClassifier {

  public static double classify(Object[] i)
    throws Exception {

    double p = Double.NaN;
    p = WekaClassifier.N32ec89882(i);
    return p;
  }
  static double N32ec89882(Object []i) {
    double p = Double.NaN;
    if (i[0] == null) {
      p = 0;
```

```
    } else if (i[0].equals("end_rack")) {
      p = 0;
    } else if (i[0].equals("cd_spec")) {
    p = WekaClassifier.N473959d63(i);
    } else if (i[0].equals("std_rack")) {
    p = WekaClassifier.N63915224(i);
    }
    return p;
  }
  static double N473959d63(Object []i) {
    double p = Double.NaN;
    if (i[2] == null) {
      p = 0;
    } else if (((Double) i[2]).doubleValue() <= 80.0) {
      p = 0;
    } else if (((Double) i[2]).doubleValue() > 80.0) {
      p = 1;
    }
    return p;
  }
  static double N63915224(Object []i) {
    double p = Double.NaN;
    if (i[3] == null) {
      p = 0;
    } else if (i[3].equals("TRUE")) {
      p = 0;
    } else if (i[3].equals("FALSE")) {
      p = 1;
    }
    return p;
  }
}
```

Open your text editor of choice and then copy and paste the Java code. Save the file as `WekaClassifier.java` (or the name of the class you specified in the options pane).

In the source code, there are actually two classes. A wrapper class that Weka generates and a main method from which to run. The core of the classifier is in the second class, `WekaClassifier`. This is basically a set of if/then statements based on the classified tree.

Testing the Classifier Code

Make a copy of the `.arff` file to test your coded classifier. Where the outcomes are yes or no, replace them with question marks (?). This means you want the classifier to work out the answer for you:

```
end_rack,85,85,FALSE,?
end_rack,80,90,TRUE,?
cd_spec,83,86,FALSE,?
```

```
std_rack,70,96,FALSE,?
std_rack,68,80,FALSE,?
std_rack,65,70,TRUE,?
cd_spec,64,65,TRUE,?
end_rack,72,95,FALSE,?
end_rack,69,70,FALSE,?
std_rack,75,80,FALSE,?
end_rack,75,70,TRUE,?
cd_spec,72,90,TRUE,?
cd_spec,81,75,FALSE,?
std_rack,71,91,TRUE,?
```

You need to write a new class to load in your test data and run each instance against the coded classifier:

```java
package chapter3;

import java.io.BufferedReader;
import java.io.FileReader;

import weka.core.Instances;

public class TestClassifier {
    public static void main(String[] args) {
        WekaWrapper ww = new WekaWrapper();
        try {
            Instances unlabeled = new Instances(new
BufferedReader(
                    new FileReader("lg2.arff")));

            unlabeled.setClassIndex(unlabeled.numAttributes() -
1);

            for (int i = 0; i < unlabeled.numInstances(); i++) {
                double clsLabel =
ww.classifyInstance(unlabeled.instance(i));
                System.out.println(clsLabel + " -> " +
unlabeled.classAttribute().value((int) clsLabel));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

}
```

The instances are loaded in and then the `for` loop iterates and uses the generated `classifyInstance()` method to get the scoring from the classifier. In this example, you're looking for the decision of whether a sale will happen or not.

Because the `classifyInstance()` returns the value as a double data type, you reference that against the class attribute array position. In this case, the

`customer_purchase` attribute has only two elements "yes" and "no." The first element in the array (0) points to "yes," and the second element (1) points to "no."

Running this example generates the following output:

```
0.0 -> yes
0.0 -> yes
1.0 -> no
1.0 -> no
1.0 -> no
0.0 -> yes
0.0 -> yes
0.0 -> yes
0.0 -> yes
1.0 -> no
0.0 -> yes
1.0 -> no
0.0 -> yes
0.0 -> yes
```

You could develop this basic code further to pull the required information from a database via Java Database Connectivity (JDBC) and then store the results again. You could even dump the results into a text file by making a copy of the instances first and updating them in the `for` loop.

```
Instances unlabeled = new Instances(new BufferedReader(
                    new FileReader("lg2.arff")));
unlabeled.setClassIndex(unlabeled.numAttributes() - 1);
Instances trained = new Instances(unlabeled);

for (int i = 0; i < unlabeled.numInstances(); i++) {
    double clsLabel = ww.classifyInstance(unlabeled.instance(i));
    trained.instance(i).setClassValue(clsLabel);
    System.out.println(clsLabel + " -> " +     unlabeled.
classAttribute().value((int) clsLabel));
}
```

The changes required are labeled in bold. This would be useful if you were to output the changes of the instances to a text file, for example.

In terms of the actual work, you're done. You can deliver some solid code.

Thinking about Future Iterations

This chapter covers a lot of ground in a short space of time: putting an `.arff` file together to creating a classifier, and generating the Java code with Weka and testing it with more unclassified data.

The test data you had was small, which is fine for getting everything working. In the real world, though, you'd be processing much more data. The question

is this: How much data should you retain for training? As a guide, I use 10 percent of the total data as a starting point and work from there. It's also worth thinking about the seasonality of data, especially if you are working in retail. Creating models for certain seasonal periods can boost the information gain in your training sets.

Time waits for no one and the same applies here. Data changes; trends change; and so do management decisions and so on. It's important to keep the classifier up to date by means of running new test data and seeing if the model can improve.

Summary

You've seen how decision trees work and the different algorithm types that are available. At a hands-on level, you've worked on a full project to create a working classifier based on the C4.5 (J48, which is the Java open source implementation as used in Weka) algorithm to predict customer purchasing behavior on products determined by placement, prominence, and pricing. Although many people perceive decision trees as simple, do not underestimate their uses. They are easy to understand and don't need a huge amount of preparation. They are often useful regardless of whether you have category or numerical data.

Bayesian Networks

You might hear the Bayesian Network referred to by a few different names: probabilistic directed acyclic graphical model, Bayes Network, Belief Network, or Bayesian Model. Based on a set of variables or parameters, it's possible to predict outcomes based on probabilities. These variables are connected in such a way that the resulting value of one variable will influence the output probability of another, hence the use of networked nodes. A Bayesian Network manages to combine probability theory with graph theory and provides a very handy method for dealing with complexity and uncertainty.

This chapter covers simple Bayesian Networks and how they are used in industry. After you have mastered the simple concepts, then you can expand your study in this area.

Pilots to Paperclips

Bayesian Networks are found all over the place where uncertainty is in play, which turns out to be a lot of places. Where there is uncertainty, there is probability.

Weather forecasting and stock option predictions are examples. The financial industry uses Bayesian Networks a lot to make reasonable predictions even when the data is not complete. Bayesian Networks are the perfect tool for the likes of the insurance, banking, and investment industries. The following are a couple of specific examples of places that Bayesian Networks are being used:

- The College of Civil Aviation at Nanjing University in China has a Bayesian Network for measuring safety risk as a result of delayed flights based on a large set of nodes.
- The Lumiere Project was born out of a Microsoft research project in 1993 with the goal of developing a platform to help users as they worked. It was later used in a pilot information system in commercial aviation, data displays to flight engineers at NASA, and that paperclip that comes up in Microsoft's Office Assistant. (Yes, Clippy is a Bayesian Network.)

NOTE The name "Bayesian Network" was initially coined by Judea Pearl to emphasize the aspects of a network that could rely on Bayes' Theorem for updating information in causal networks—things that are directly related to each other.

To get started with implementing your own Bayesian Networks, you first need to come to grips with graphs, probability, and Thomas Bayes. Then you can put it all together.

A Little Graph Theory

Graph theory seems to invoke the fear of spiritual beings when it's mentioned, but after you get the basics it's not that difficult. I think a lot of confusion is borne from the definitions.

Math folk call what's shown in Figure 4-1 a vertex, but I prefer to call it a node.

Node 1
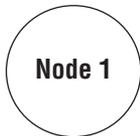
Figure 4-1: A node (or vertex)

The nodes can signify anything you want. It could be rock bands, actors, films, or proteins.

You can have as many nodes as you want, but you need to find a way of connecting them together; that's where the edge (a line that connects the nodes, not the guitarist from U2) comes in, as shown in Figure 4-2.
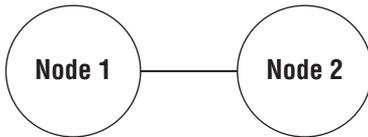
Figure 4-2: Two nodes (or vertices) and an edge

You now know there's a relationship between two nodes.

When you need to manipulate and traverse a lot of edges in a graph quickly, it's worth looking at using large-scale graph databases such as Neo4J, Apache Giraph, or Spark. If you are interested in a language to query the nodes, then have a look at any reference to the Cypher language, which is very similar to the following example graph.

```
MATCH (actress)-[:acted_in]->(film)
```

Conceptual graphs are used in computer science and support a relationship between the nodes. The relationship is defined in the edge like the previous example in Figure 4-2.

When you add direct arrows to the edges, you have a directed graph. These can sometimes be called arcs or directed edges. The relationship is defined by the connection and the direction of the arrow. Figure 4-3 shows an example.
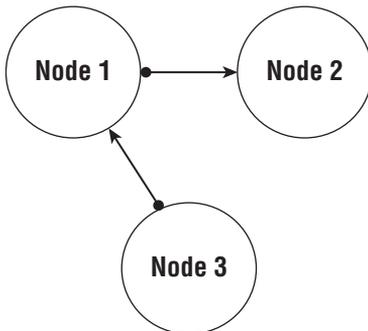
Figure 4-3: Directed and undirected graphs

For the purpose of explaining Bayesian Networks, this simple notation will suffice for now in terms of the theory. Next you need to turn your attention to some probability.

A Little Probability Theory

Okay, hands up if you're having flashbacks to learning probability theory at school. Do you feel a little panicked? There is no need to fret over such things. Probability is, quite simply, the measure of the likeliness that an event will occur.

The question of interest will be something along the lines of "What will the result of the next coin toss be?" You use a value between 0 and 1 as your unit of measurement; the higher the probability that the event will happen, the higher the value of the number given.

Coin Flips

For a coin toss you know there are two possible outcomes—either heads or tails—with a 50 percent chance of either event occurring. There is a notation for describing probability. For the coin toss outcomes you would write the following:

p(Coin toss will be heads) = 0.5 (or ½ or 50%)
p(Coin toss will be tails) = 0.5 (or ½ or 50%)

Now that's just for one coin toss. The answer to the question "What are the chances of the next coin toss being heads?" is ½. There can only be two possible outcomes.

Conditional Probability

So far, this chapter has concentrated on fairly mundane and highly improbable (but not impossible) day-to-day events. What about completely unconnected events? Well, you can apply conditional probability to events that have just occurred.

Take a look at the coin flips again. You've already covered one coin flip has two possible outcomes: heads or tails. What are the chances that two heads show up in two coin flips?

You have a series of events:

Coin flip 1. (*A*)
Coin flip 2. (*B*)

The probability is shown like so:

p(A | B)

You know that with one coin there is ½ chance of turning up heads. The possible outcomes for two coins are ½ x ½, which is ¼. You're simply multiplying the number of coin flips (2) by the probability of heads showing per coin flip (½). It's time to move on to a more realistic, everyday example.

Winning the Lottery

Coin flips are helpful and in some forms of gambling the game with two coin flips is still played. But these days the lottery is a more common game of probability. Everyone wants to win the lottery, right?

In the UK, the National Lottery has 49 balls. To win the jackpot prize you have to correctly guess all six numbers. With six predictions and 49 balls, what are the chances that you will win the lottery?

When the first ball is picked, there are six chances to pick one of the chosen numbers with 49 balls from which to choose. Next time there are 48 balls and five chances, and so on until all six numbers are picked. This results in the following mathematical equation:

$$\frac{6}{49} \times \frac{5}{48} \times \frac{4}{47} \times \frac{3}{46} \times \frac{2}{45} \times \frac{1}{44} = \frac{720}{10,068,347,520}$$

The number on the bottom line is basically just more than 10 billion. So, dear reader, there is a 720 in 10,068,347,520 chance of winning the lottery jackpot, or 1 in 13,983,816.

Depending on what country you live in, there are various lotteries with different possible outcomes. The Euromillons, for example, is played in a number of European countries, so the jackpot is usually higher. The downside is that, because you are looking for seven numbers (five from the main set of numbers 1–50 and two "lucky stars" numbered 1–11), the probability to win the jackpot is 1 in 116,531,800. Other lotteries play six numbers from 44 balls, some from 39 balls. Regardless of how you look at it, the chances of winning are slim but not impossible.

I'm going to park probability there; I only wanted to cover what you need to know in terms of Bayesian Networks. You've marveled at how graphs work, rolled a six-sided die some number of times to calculate probability, and even played the lottery once or twice in the name of research. Now you can delve into a little bit of background about a gentleman named Thomas and a theorem named after him: Bayes' Theorem.

Bayes' Theorem

If there's one aspect of machine learning you'll hear talked about, it's the application of Bayes' Theorem. You might also hear the terms Bayes' Law or Bayes' Rule used, but all three are essentially the same thing.

Thomas Bayes was born in 1701 and died in 1761. As well as being a Presbyterian minister, he was a philosopher and statistician. His theorem wasn't named after

him until after his death in 1763 when his essay, "An Essay Towards Solving a Problem in the Doctrine of Chances," was cited by Richard Price to help prove the existence of God.

The important part of Bayes' Theorem is the observation of previous events, or your degree of belief that something will occur. If you have a degree of belief of one event happening, you can apply it to new data and make an informed calculation to its probability. There are plenty of examples of the application of Bayes' Theorem to determine the result of more coin flips, the spread of disease, or the potential gender of offspring. I'm going to avoid all the clichéd examples and demonstrate with my own about country music.

DIAGNOSING COUNTRY MUSIC

For the record, I do like country music—well, certain types of it. Please don't judge me, but for the purpose of explaining Bayes' Theorem, I'm going to use country music to get the point across.

The test for diagnosing whether or not a person likes country music is conducted by playing a mixture of Nanci Griffith, Mary Chapin Carpenter, Garth Brooks, and Lyle Lovett to the listener. Based on various brain responses, you can deduce a negative or positive response from the test.

We know a positive test is 95 percent accurate; the listener likes country music. The test is 99 percent successful in diagnosing those who do not like country music. People in the know will call this 98 percent sensitivity and 99 percent specificity. If you call the test T and whether a person likes Country music C, you have the following probability outcomes:

> C likes country
>
> ~C does not like country
>
> T tests positive for liking country
>
> ~T tests negative for liking country

The probabilities are listed for sensitivity as

$$p(T \mid C) = 0.95$$

For specificity you get

$$p(\sim T \mid \sim C) = 0.99$$

And finally, you assume that 2 percent of the population will like country music:

$$p(D) = 0.02$$

Although it's all well and good to run these experiments, you also know that false positives can creep in. There is a chance that someone will test positive for liking country music but doesn't actually like it. There are also false negatives to take into account—someone who in fact does like country music but tests negative.

With Bayes' Theorem, you can calculate the probability with all the previously defined information. In scary math books, it looks like this:

$$p(C \mid T) = \frac{p(T \mid C)p(C)}{p(T \mid C)p(C) + p(T \mid \sim C)p(\sim C)}$$

It looks complicated, doesn't it? It reads easier after you get the values in the equation. What you are really saying is this:

$$p(C \mid T) = \frac{0.95 \times 0.02}{(0.95 \times 0.02) + (0.02 \times 0.98)} = \frac{0.019}{0.019 \times 0.0184} = 0.51$$

There's still a 51 percent chance that a test will return a false positive result. All that excellent music will be wasted on those subjects, as they don't really like it that much.

Those are the three building blocks of graph theory, probability, and Bayes Theorem explained in the simplest way possible. Now you can return to the actual Bayesian Network to see how it is put together.

How Bayesian Networks Work

Now that you have a basic grasp of graphs, probability, and Bayes' Theorem, look at how the network is put together.

Consider the graph shown in Figure 4-4.
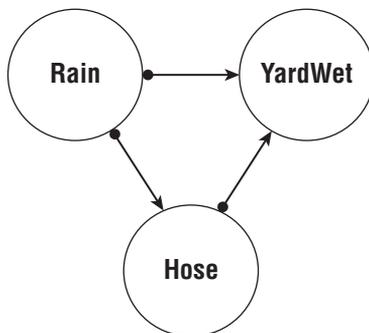
Figure 4-4: A basic Bayesian Network

In the classic "Is my backyard wet?" graph (or "Is the grass wet" graph) shown in Figure 4-4, you can see there are three nodes: Yard Wet, Rain, and Hose. There are two events that would cause the yard to be wet; either the owner had

hosed it or it has been raining. In any normal circumstance, you wouldn't hose the yard while it was raining.

For each node you can assign true/false values:

Y = Yard wet (True or False)

R = Raining (True or False)

H = Someone using the hose (True or False)

The joint probability would be written as

$$p(Y,R,H) = p(Y \mid R,H) \, p(R \mid H) \, p(R)$$

With these variables in hand, you can start asking some questions—for example, what's the probability that it's raining when the yard is wet?

Assigning Probabilities

As mentioned earlier in the chapter, probability values are between 0 and 1. As the nodes are all either true or false, you can start to assign some basic outcomes to the nodes. First is the node Rain. Because it doesn't have any parent nodes, it is easy to assign a probability to it.

RAIN	
True	**False**
0.2	0.8

Next, look at the Hose node; it uses the Rain node as a parent node, so you need to assign probabilities for each of the outcomes of the parent node. The outcomes table for Hose looks like the following:

HOSE		
Value of Rain Node	**True**	**False**
False	0.4	0.6
True	0.01	0.99

The main thing to note is that the values always add up to 1. If you've gone over that amount, then you need to correct it.

The last node is Yard, which has two parents: Rain and Hose. You need to ensure that all the outcomes are taken into account.

YARD			
Hose	**Rain**	**True**	**False**
False	False	0.0	1.0
False	True	0.8	0.2

YARD			
True	False	0.9	0.1
True	True	0.99	0.01

The Yard node includes all the outcomes that are related to it. Once again, all the probabilities add up to 1.

You have three variables, each with two possible values and all the probabilities assigned. Now you can calculate some results.

Calculating Results

As with all good data analysis, you should start with a question. In this case, the question is "Given that the yard is wet, what's the probability that it's raining?"

Start with what you know. The clues are in the question: the yard is wet (True), and we want to know if it's raining (True). The only variable you're not certain about is the state of the hose; it could be true or false as it stands.

The joint probability function written at the start gives you the relationship of the nodes and their parents from a probability point of view. What you're basically saying is

$$p(Y=True, H=True, R=True)$$
$$p(Y=True \,|\, H=True, R=True) \times p(H=True \,|\, R=True) \times$$
$$p(R=True)$$

I'll break that down a little, as it might be confusing if you're not used to reading it. What is being said is this:

Multiply the values of the wet Yard probability, where the values for Hose and Rain are also true, by the probability of Hose being true with the value of Rain being true, by the probability of Rain being true.

So, the values you need are

$$0.99 \ (Y=T, H=T, R=T) \times 0.01 \ (H=T, R=T) \times 0.2 \ (R=T) = 0.00198$$

Next, you work out the value of the probability if the hose was false. This is the variable you don't know, so it's important to work out the probability for it.

$$p(Y=True \,|\, H=False, R=True) \times p(H=False \,|\, R=True) \times$$
$$p(R=True)$$
$$0.8 \times 0.99 \times 0.2 = 0.1584$$

This part of the calculation is for the upper part of the equation (the numerator); now you can plug in values for the lower part (the denominator).

All that happens is a repeat of the earlier equation but for the other outcomes of the yard being wet.

T,T,T = we know is 0.00198
T,F,T = we know is 0.1584

T,T,F = $0.9 \times 0.4 \times 0.8 = 0.288$
T,F,F = $0.0 \times 0.6 \times 0.8 = 0.0$

So the final equation looks like this:

$$\frac{0.00198 + 0.1584}{0.00198 + 0.288 + 0.1584 + 0.0}$$

The top divided by the bottom results in 0.3576876, but just round it up to 0.3577. As a percentage, that's 35.77 percent.

Node Counts

Bayesian Networks depend on a lot of counting. The more nodes you have, the more counting the network has to do. The earlier example has three nodes. Each node has only two probability variables. So you can compute the counts easily.

$$2 \times 2 \times 2 = 8 \text{ counts}$$

If you had a network with 12 nodes, 7 of which had three variables and the other 5 had six variables, you would have a calculation of:

$3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 3 \times 6 \times 6 \times 6 \times 6 \times 6 = 17,006,112$ counts

You have to be aware that as you add nodes with n number of variables, the performance needs of your computing will rise dramatically. It will take longer to gain results in the first instance when the computing of the network is done.

Bayesian Networks can run into memory problems as well, so keep that in mind while you are developing your programs.

Using Domain Experts

One thing you might be wondering reading this chapter and the "How Bayesian Networks Work" section is "Who's dictating the initial probabilities?" Well, in this instance I decided on the probabilities by way of illustrating how the network is put together. In a real-world context it will be, more than likely, a domain expert.

Software developers are great at developing software; that's what they do. However, they're not always good at knowing the rest of the business domain. If you ask them the question "Why do our customers always buy this product?" chances are they won't have the answers. Someone who knows the retail domain and how customers think will have a better understanding than they do.

It's important to sit down with the domain experts from the start and gather as much information as you can to get the network in a good working order to give reasonable answers. You will refine these things later on; as the algorithm gets used more and more, the results will get better after you share the output with the domain expert and the rest of the team and they update the values.

A Bayesian Network Walkthrough

There are a number of libraries you can use for creating and running Bayesian Networks. Weka has its own support for them and incorporates the K2 algorithm for learning. The emphasis on many tools, such as OpenMarkov and Weka, is the use of a graphical user interface (GUI) to enable you (and the domain experts) to create the graphs and assign the probabilities. For this walkthrough I will use the JavaBayes API.

Java APIs for Bayesian Networks

Because this book uses Java for the core of the work, it would be nice to use a Java API that creates nodes, their edges, and their probabilities. Netica (`https://www.norsys.com/netica.html`) is one, but it is a commercial application (meaning you have to pay for it) and it relies on native libraries of the operating system on which you are working.

Another library is Jayes (`https://github.com/kutschkem/Jayes`) and it's used within Eclipse for code completion algorithms. It's open source; its build process is tied to Maven; and it is not the easiest to get going easily. It's certainly worth a look, though; so put it on your to-do list for when you have time.

Lastly, there's JavaBayes. To give you an idea how old it is, it was originally written for Java 1.0.2, so it goes back quite a while. On the plus side, Joe Schweitzer at Dataworks has updated it on Github, so you can download it and use it fairly quickly.

For the walkthrough, I've added the helper classes from Joe's original code samples so they are in one complete library. If you want to complete the tutorial, then you need to download the jar file from `https://github.com/jasebell/JavaBayesAPI.`

Planning the Network

Before you start any coding, you have to think about what needs to be done. You're creating a simple Bayesian Network in Java, and this section describes the plan.

The best place to plan these sorts of things is on paper; it's also handy to grab your nearest domain expert who's well versed in the domain you are working in. You'll use a real-world example that would require an expert, as it's fair to say most people already know what causes our yards and lawns to be wet.

Cervical Spondylotic Myelopathy (CSM) is a spine disease and is one of the most common spinal cord dysfunctions in patients over 55 years old. Symptoms include, but are not limited to, impaired gait leading to issues walking, numbness of hands, and weakness in general. Surgical procedures are available and performed when the CSM progresses to mild or severe symptoms.

You're going to design a Bayesian Network with some given information and calculate the surgical outcome based on what you know. First, you walkthrough what you need to prepare and then transfer it to code.

Determining Nodes

Every node needs to be created. This example requires a bit more involved version of a Bayesian Network that has a few more nodes than the explanation previously covered. For CSM we're going to look at the following nodes:

- Age of patient (A)
- Does the patient smoke? (S)
- Duration of symptoms (D)
- Surgical outcome success (SS)

This walkthrough keeps it simple; for the aim of this tutorial there are a range of other things that you could have measured and made nodes from; they would potentially give you a more refined network. For the present, this is a good starting point.

Assigning Probabilities

You must assign all probabilities for all nodes including the probabilities in the parent nodes. If this sounds a little confusing, try rereading the Yard/Hose/Rain example earlier in the chapter.

For each of the nodes, you have the following probabilities:

AGE (A)	
<55	**>55**
0.8	0.2

SMOKER (S)		
Value of (A)	Smokes	Does Not Smoke
<55	0.4	0.6
>55	0.8	0.2

DURATION OF SYMPTOMS (D)	
< 2 Years	> 2 Years
0.9	0.1

SURGICAL OUTCOME SUCCESS (SS)			
(S)	(D)	Positive	Negative
Smoker	<2Y	0.1	0.9
Smoker	>2Y	0.01	0.99
Non smoker	<2Y	0.8	0.2
Non smoker	>2Y	0.58	0.42

The tables are similar to the ones earlier in the chapter: an exhaustive list of probabilities and conditional probabilities wherever they are required, which give the basis of the outcomes of the network when it's run.

With this information from the expert you can now assign the probabilities in code.

Coding Up the Network

You have down on paper (sort of) what you are looking to do. The next step is to create some code. You're going to work through a complete project from start to finish. I'm using Eclipse for this project, but you can easily substitute your own integrated development environment (IDE).

Creating the Project

First, create a clean project with which to work. Click File ➪ New ➪ Java Project. Call this project BayesNetDemo, as shown in Figure 4-5.
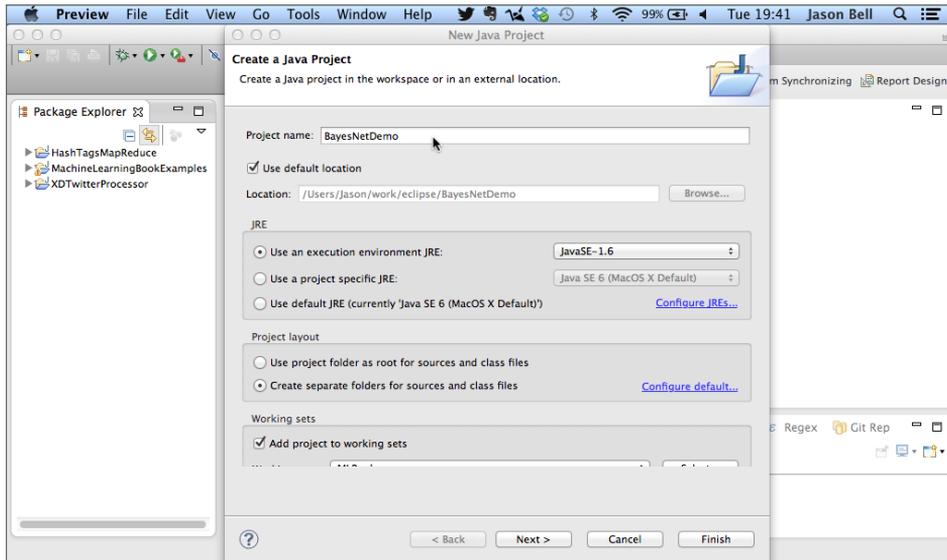
Figure 4-5: Creating a new project

Adding the JavaBayes Library

You need the JavaBayes library to complete the demo, so if you haven't done so already, get the precompiled jar file from the Github repository at `https://github.com/jasebell/JavaBayesAPI`.

Create a new directory on your filesystem and via the command line type the following (see Figure 4-6):

```
git clone https://github.com/jasebell/JavaBayesAPI
```

Back in your IDE, you need to add the jar file to your build properties so your code can see the library. In Eclipse you do this by clicking the name of the project in the project chooser window. Right-click and you see the Properties option.

Click Java Build Path and then click Add External JARs, as shown in Figure 4-7. Find the file location of the jar file and click Open.
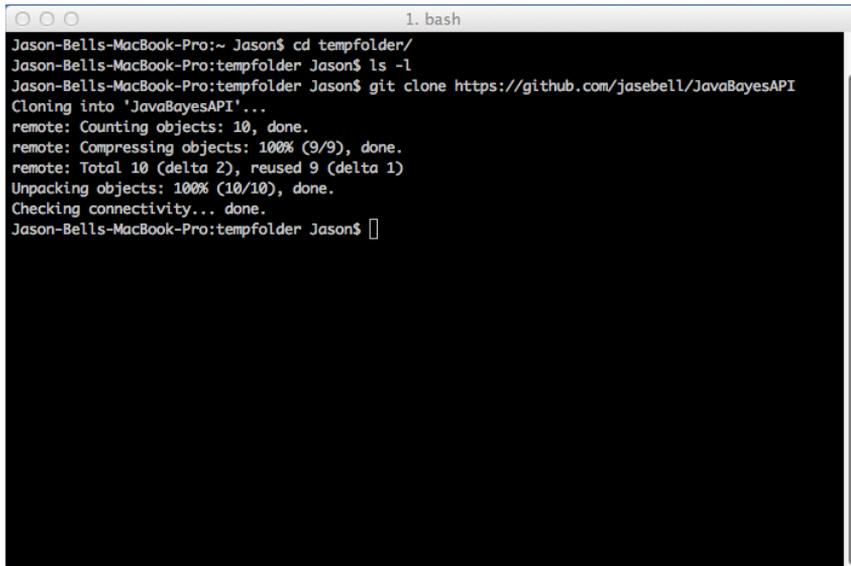
Figure 4-6: Cloning the Git repository
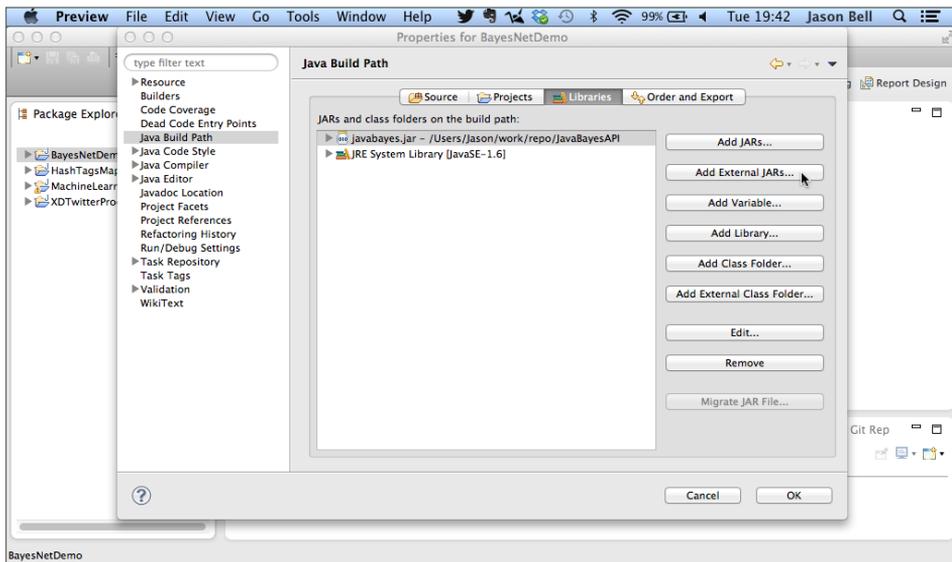
Figure 4-7: Project properties

Creating the Base Graph

With the library added, you can start to put some code in place. Create a new Java class called `BayesNetExample.java`. In Eclipse, select File ⇨ New ⇨ Class, as shown in Figure 4-8. Make sure you have a main method to run the code as well.
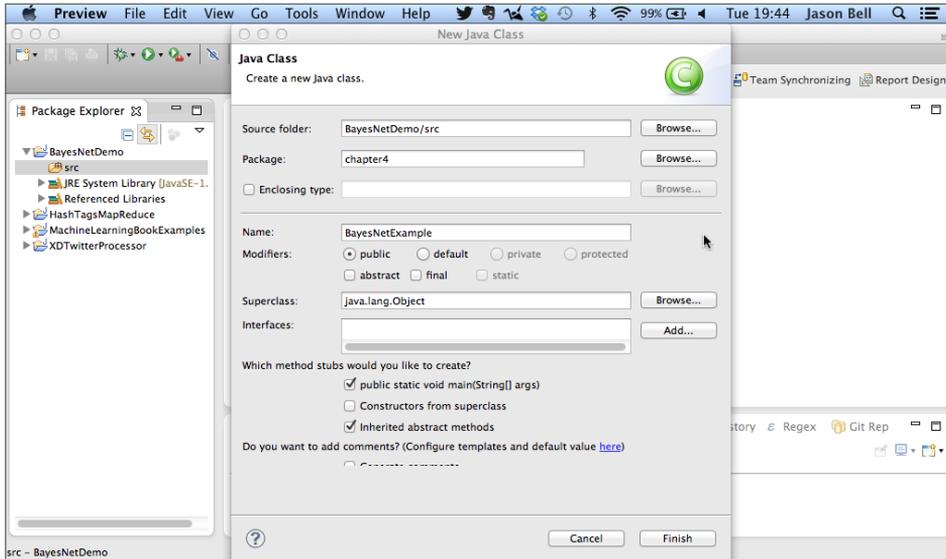
Figure 4-8: Creating a new class

Create the constructor and add an InferenceGraph class. This is your graph network; you'll use this class to do the belief calculations at the end.

```java
import javabayes.Helpers.BayesNetHelper;
import javabayes.InferenceGraphs.InferenceGraph;
import javabayes.InferenceGraphs.InferenceGraphNode;
public class BayesNetExample {
    public BayesNetExample() {
        InferenceGraph inferenceGraph = new InferenceGraph();


    }

    public static void main(String[] args) {
        BayesNetExample bne = new BayesNetExample();
    }
}
```

I've added the line in the main method to ensure that the program runs.

Adding the Nodes

You have four nodes: the age of the patient, the patient's smoking status, the duration of the condition, and the surgical outcome. What's required code-wise is the creation of each node connected to the graph. First, here is a Java code representation of the age node:

```
InferenceGraphNode age = BayesNetHelper.createNode(inferenceGraph,"under
55", "<55", ">55");
```

The `InferenceGraphNode` assigns the node to the graph class you created; you also give it a name `"under55"` and then the names of the two outcomes, `"<55"` and `">55"`. These are the true/false states of the node. You'll assign values to these shortly. To create the nodes you're using a helper class called `BayesnNetHelper`, which has a `createNode()` method that does the work for you.

One node down, three to go; here is the Java code to represent the other three nodes:

```
InferenceGraphNode smoker = BayesNetHelper.createNode(inferenceGraph,
"smoker", "smokes", "doesnotsmoke");
InferenceGraphNode duration = BayesNetHelper.createNode(inferenceGraph,
"duration", "<2Y", ">2Y");
InferenceGraphNode surgical = BayesNetHelper.createNode(inferenceGraph,
"surgicalOutcome", "positive", "negative");
```

Right now the nodes don't know each other; there are no edges set to connect the nodes together. That's covered in the next section.

Connecting the Nodes

Now that you have nodes, you can connect them. Remember from the yard example earlier in this chapter that certain nodes were parents of others. For this CSM model, the age node is the parent of the smoker node, which in turn is the parent of the surgical outcome node. Also, the duration node is the parent of the surgical outcome node.

In the code, you connect the nodes (called arcs in the syntax of the Java code, but still graph edges to me) with the `create_arc()` function. It takes the following syntax.

```
graph.create_arc(parent_node, child_node);
```

The following are your newly created `InferenceGraphNodes`; you can now create the arcs that connect them.

```
inferenceGraph.create_arc(age, smoker);
inferenceGraph.create_arc(smoker, surgical);
inferenceGraph.create_arc(duration, surgical);
```

If you were to draw the graph, you'd have something along the lines of what's shown in Figure 4-9.

Age (A)

<55	>55
0.80	0.20

Smoker (S)

A	Smokes	Does Not Smoke
<55	0.40	0.60
>55	0.80	0.20

Duration (D) of Symptoms

<2YR	>2YR
0.90	0.10

Surgical Outcomes (SS)

S	D	Positive	Negative
Smoker	<2YR	0.10	0.90
Smoker	>2YR	0.01	0.99
Non Smoker	<2YR	0.80	0.20
Non Smoker	>2YR	0.52	0.42
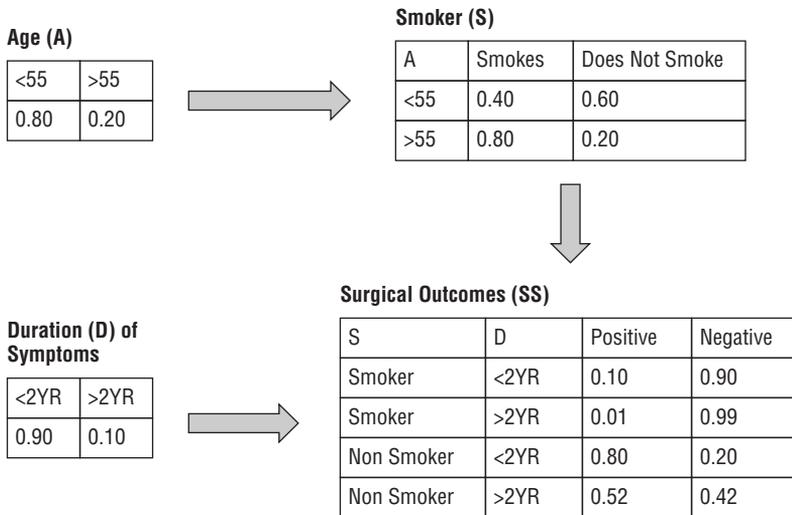
Figure 4-9: The CSM graph

With the nodes connected, it's time to turn your attention to the probabilities.

Assigning Probabilities

The `BayesNetHelper` class enables you to set the probability values while taking away the complexity. Start with the conditional probabilities. The smoker node is conditional with the age of the patient. In pseudocode, it would look like this:

if age < 55 then smoker(smokes) = 0.4, smoker(doesnotsmoke) = 0.6
if age > 55 then smoker(smokes) = 0.8, smoker(doesnotsmoke) = 0.2

In the Java class, you add the values with the helper class:

```
BayesNetHelper.setProbabilityValues(smoker, "<55", 0.4, 0.6);
BayesNetHelper.setProbabilityValues(smoker, ">55", 0.8, 0.2);
```

Next you set the probability values for the surgical outcome node; this one has two conditional probabilities connected to it, so you have to look at all the possible conditions and assign the probabilities accordingly:

if smokes and duration < 2Y then surgical(positive) = 0.1, surgical(negative) = 0.9
if smokes and duration > 2Y then surgical(positive) = 0.01, surgical(negative) = 0.99
if does not smoke and duration < 2Y then surgical(positive) = 0.8, surgical(negative) = 0.2
if does not smoke and duration > 2Y then surgical(positive) = 0.58, surgical(negative) = 0.42

In the code, you again add the probabilities with the helper class. Notice this time you use parent and the secondary parent conditions.

```
BayesNetHelper.setProbabilityValues(surgical, "smokes", "<2Y",
0.1, 0.9);
BayesNetHelper.setProbabilityValues(surgical, "smokes", ">2Y",
0.01, 0.99);
BayesNetHelper.setProbabilityValues(surgical, "doesnotsmoke",
"<2Y",0.8 , 0.2);
BayesNetHelper.setProbabilityValues(surgical, "doesnotsmoke", ">2Y",
0.58, 0.42);
```

The last two nodes are called leaf nodes because they don't have any parents connected to them. All you have to do with these is add the true/false values via the helper. You set one for the duration node and another for the age node.

```
BayesNetHelper.setProbabilityValues(duration, 0.9, 0.1);
BayesNetHelper.setProbabilityValues(age, 0.8, 0.2);
```

With nodes created, edges defined, and probabilities set, you can now start getting some results from the code. Make sure you have saved your work before running it.

Testing the Belief Network

To get the percentage belief of the network, you have to use the helper class again. It has a method called `getBelief()` and it takes the graph and the node for which you want to get the belief value. First off, look at the predicted probability of a surgery's success when you have no other conditions assigned to it:

```
double belief = BayesNetHelper.getBelief(inferenceGraph, surgical);
System.out.println("The probability of surgery being positive: " +
belief);
```

You will see the output in the console window:

```
The probability of surgery being positive: 0.44823999999999997
```

The model's predicted probability of a surgery's being successful is 44.8 percent; remember this calculation is made without any other conditions. When you know more information about the patient, you'll run the program again and see how the model's predicted probability for a successful surgery changes for the patient. Remember that the model is based on the predicted probabilities based on the expert's recommendation. There's no assurance that the surgery is going to be successful; only the model's predicted result. The more times the surgery is performed should validate the model's accuracy in the long term.

Testing the Belief Network with a Condition

You now have some information about the patient so you can add this information to the program and see how the chances for a successful surgery look.

The age of the patient is less than 55. You can set the observation within the age node itself and then run the belief calculation again.

```
age.set_observation_value("<55");
belief = BayesNetHelper.getBelief(inferenceGraph, surgical);
System.out.println("The probability of surgery being positive and
patient is younger than 55 : " + belief);
```

Run the program again and the model's prediction of the patient's chances improves.

```
The probability of surgery being positive and patient is younger than 55
: 0.5032
```

If the patient smokes, it would have an impact on the result from the model. Adding another observation to the node you now have the following to compute:

```
smoker.set_observation_value("smokes");
belief = BayesNetHelper.getBelief(inferenceGraph, surgical);
System.out.println("The probability of surgery being positive for a
smoker, younger than 55: " + belief);
```

Adding this last datum changes things drastically for the patient! While being younger than 55 is a good indicator for successful surgery, the fact he or she smokes causes the predicted probability from the network to decrease drastically.

```
The probability of surgery being positive for a smoker, younger than 55:
0.09100000000000001
```

The experts know that because of the location of the issue—the neck—surgery will affect the throat area, so there's a high probability the surgery will fail; many patients who desire surgery are refused on that basis alone.

Lastly add the duration the patient has had the symptoms. In this patient's case it's been more than two years.

```
duration.set_observation_value(">2Y");
belief = BayesNetHelper.getBelief(inferenceGraph, surgical);
System.out.println("The probability of surgery being positive for a
smoker, younger than 55 with symptoms over 2 years: " + belief);
```

Once again this decreases the chance of successful surgery.

```
The probability of surgery being positive for a smoker, younger than 55
with symptoms for over 2 years: 0.01
```

You've created a Bayesian Network to calculate the probability of successful surgery of CSM. With the conditional parameters, you see that the probability varies enormously, especially when the age and smoking factors are taken into account.

The following is the full code listing for the Bayes Network class created in this chapter.

Listing 4-1: Bayes Network class

```java
import javabayes.Helpers.BayesNetHelper;
import javabayes.InferenceGraphs.InferenceGraph;
import javabayes.InferenceGraphs.InferenceGraphNode;

public class BayesNetExample {
    public BayesNetExample() {
        InferenceGraph inferenceGraph = new InferenceGraph();

        InferenceGraphNode age = BayesNetHelper.createNode(inferenceGraph,"
under55", "<55", ">55");
        InferenceGraphNode smoker = BayesNetHelper.
createNode(inferenceGraph, "smoker", "smokes", "doesnotsmoke");
        InferenceGraphNode duration = BayesNetHelper.
createNode(inferenceGraph, "duration", "<2Y", ">2Y");
        InferenceGraphNode surgical = BayesNetHelper.
createNode(inferenceGraph, "surgicalOutcome", "positive", "negative");

        inferenceGraph.create_arc(age, smoker);
        inferenceGraph.create_arc(smoker, surgical);
        inferenceGraph.create_arc(duration, surgical);


        BayesNetHelper.setProbabilityValues(smoker, "<55", 0.4, 0.6);
        BayesNetHelper.setProbabilityValues(smoker, ">55", 0.8, 0.2);

        BayesNetHelper.setProbabilityValues(surgical, "smokes", "<2Y", 0.1,
0.9);
        BayesNetHelper.setProbabilityValues(surgical, "smokes", ">2Y",
0.01, 0.99);
        BayesNetHelper.setProbabilityValues(surgical, "doesnotsmoke",
"<2Y",0.8 , 0.2);
        BayesNetHelper.setProbabilityValues(surgical, "doesnotsmoke",
">2Y", 0.58, 0.42);

        BayesNetHelper.setProbabilityValues(duration, 0.9, 0.1);

        BayesNetHelper.setProbabilityValues(age, 0.8, 0.2);

        double belief = BayesNetHelper.getBelief(inferenceGraph, surgical);
```

```
        System.out.println("The probabilty of surgery being postive: " +
belief);

        age.set_observation_value("<55");
        belief = BayesNetHelper.getBelief(inferenceGraph, surgical);
        System.out.println("The probability of surgery being postive and
patient is younger than 55 : " + belief);


        smoker.set_observation_value("smokes");
        belief = BayesNetHelper.getBelief(inferenceGraph, surgical);
        System.out.println("The probability of surgery being postive for a
smoker, younger than 55: " + belief);

        duration.set_observation_value(">2Y");
        belief = BayesNetHelper.getBelief(inferenceGraph, surgical);
        System.out.println("The probability of surgery being postive for a
smoker, younger than 55 with symptoms over 2 years: " + belief);




    }

    public static void main(String[] args) {
        BayesNetExample bne = new BayesNetExample();
    }
}
```

NOTE The probabilities in the walkthrough are just base values used for demonstration; they are not connected to any study or informed by any expert. They are merely there to illustrate how Bayesian Networks can be formulated.

Summary

This chapter covered a lot of ground, including simple graph theory, probability, and Bayes' Theorem. The Bayesian Network examples show that it's straightforward to create a network, create the nodes and connect them, and then assign probabilities and conditional probabilities. After you apply existing observations, your overall results change due to the nature of the graph.

Coding these things can be difficult and require some proper planning on paper. Wherever you can, try to enlist a domain expert to help with the initial values of the probabilities, because this will make your final prediction output more accurate.

Artificial Neural Networks

There's something about gathering knowledge about the human brain that makes people tick. Many people think that if we can mimic how the brain works, we'll be able to make better decisions.

In this chapter, you look at how artificial neural networks work and how they are applied in the machine learning arena.

What Is a Neural Network?

Artificial neural networks are essentially modeled on the parallel architecture of animal brains, not necessarily human ones. The network is based on a simple form of inputs and outputs.

> **...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.**
>
> *Dr. Robert Hecht-Nielson as quoted in*
> *"Neural Network Primer: Part I" by*
> *Maureen Caudill, AI Expert, Feb. 1989*

In biology terms, a *neuron* is a cell that can transmit and process chemical or electrical signals. The neuron is connected with other neurons to create a

network; picture the notion of graph theory with nodes and edges, and then you're picturing a neural network.

Within humans, there are a huge number of neurons interconnected with each other—tens of billions of interconnected structures. Every neuron has an input (called the *dendrite*), a cell body, and an output (called the *axon*), as shown in Figure 5-1.
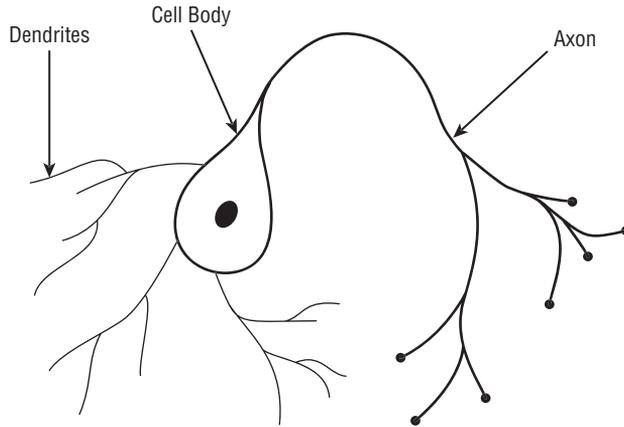
Figure 5-1: The neuron structure

Outputs connect to inputs of other neurons and the network develops. Biologically, neurons can have 10,000 different inputs, but their complexity is much greater than the artificial ones I'm talking about here.

Neurons are activated when the electrochemical signal is sent through the axon. The cell body determines the weight of the signal, and, if a threshold is passed, the firing continues through the output, along the dendrite.

Artificial Neural Network Uses

Artificial neural networks thrive on data volume and speed, so they are used within real-time or very near real-time scenarios. The following sections describe some typical use cases where artificial neural networks are used.

High-Frequency Trading

With the way artificial neural networks mimic the brain but with a much increased speed factor, they are perfect for high-speed trading. Because HFT can make decisions far faster than a human can—thousands of transactions can be done

in the same time it takes a human to make one—it's obvious why the majority of stock market systems have gone to the automated trading side.

High-frequency trading is usually done on a supervised learning method; there is a lot of training data available from which to learn. The artificial neural network is looking for entropy from the incoming data.

Credit Applications

Although many examples of credit applications are performed with decision trees, they are often run with artificial neural networks. With the variety of application data available, it's a fairly straightforward task to train the model to spot good and bad credit factors.

Data Center Management

Google uses neural networks for data center management. With incoming data on loads, operating temperatures, network equipment usage, and outside air temperatures, Google can calculate efficiency of the data center and be able to adjust the settings on monitoring and cooling equipment.

Jim Gao started this exercise as a Google 20 percent project (a program in which Google employees are encouraged to use 20% of their work time on their own projects) and, over time, has trained the model to be 99.6 percent accurate. If you are interested in reading more on this check out Google's blog post on the subject at `http://googleblog.blogspot.ca/2014/05/better-data-centers-through-machine.html`.

Robotics

Artificial intelligence has been used in robotics for several years. Some artificial intelligence requires pattern recognition, and some requires huge amounts of sensor data to be fed into a neural network to determine what movement or action to take.

Training models in robotics takes an awful long time to create, mainly because there are potentially so many different inputs and output variables to process and learn from. For example, developers of autonomous driving vehicles need hundreds of hours of previous driving data to make a model that can handle many road conditions. (Personally, I still prefer my hands on the wheel.)

Medical Monitoring

Medical machinery can be monitored via artificial neural networks, which involves the constant updating of many variables, such as heart rate, blood pressure, and so on.

Conditions that have multiple variations and trigger symptoms can be calculated and monitored, and staff can be alerted when the variables go over certain thresholds.

Breaking Down the Artificial Neural Network

One of the keys to understanding the artificial neural network is knowing that the application of the model implies you're not exactly sure of the relationship of the input and output nodes. You might have a hunch, but you don't know for sure. The simple fact of the matter is, if you did know this, then you'd be using another machine learning algorithm.

Before you jump into data and examples, have a look at the components in a neural network.

Perceptrons

The basis for a neural network is the *perceptron*. Its role is quite simple. It receives an input signal and then passes the value through some form of function. It outputs the result of the function. (See Figure 5-2.)

Input   Output

2.5   1.5
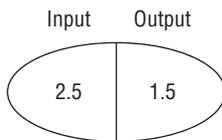
Figure 5-2: A simple perceptron

Perceptrons deal with numbers when a number or vector of numbers is passed to the input. It is then passed to a function that calculates the outgoing value; this is called the *activation function*. The node can handle any number of inputs—Figure 5-3 shows two inputs passing in to the function—and it takes the weighted sum of all the inputs.

Assuming the input is a vector Z, you'd end up with something like this:

$Z1 = 2$

$Z2 = 5$

$Z3 = 1$

Or (2,5,1)

The weighted sum of all the inputs is calculated as follows:

$$\sum_i wiZi$$

In other words, "add it all up." So for the likes of me, who is not used to too much math notation, it looks like the following:

$$2_{w1} + 5_{w2} + 1_{w3}$$

The outgoing part of the node has a set threshold. If the summed value is over the threshold, then the output, denoted by the y variable, is 1, and if it's below the threshold, then y is 0 (zero).

You end up with the following equation:

$$if \sum_i wiZi \geq t \; then \; y = 1$$
$$else \; y = 0$$
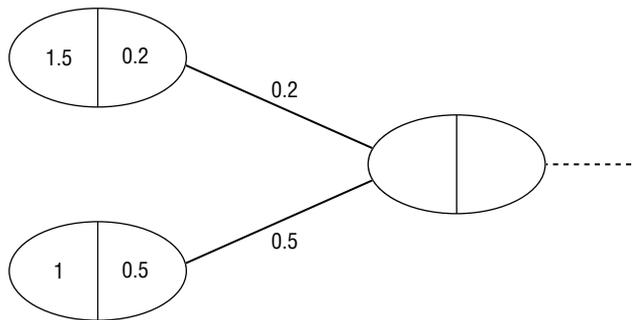
Figure 5-3: Perceptron with two inputs

The weight of the perceptron can be zero or any other value. If the weight value is zero, then it does not alter the input node value coming into the perceptron. Likewise, inputs can be positive or negative numbers. The key to the output is based on the weighted sum against the threshold.

That's the basis of a single-node perceptron. When you strip the components apart, it's quite basic in composition.

Activation Functions

The *activation function* is the processing that happens after the input is passed into the neuron. The result of this function determines whether the value is passed to the output axon and onto the next neuron in the network.

Commonly, the Sigmoid function (see Figure 5-4) and the hyperbolic tangent are used as activation functions to calculate the output.

The Sigmoid function only outputs one of two values: 0 and 1. For the programmers, the function is written as

```
return 1.0 / (1.0 + Math.exp(-x));
```
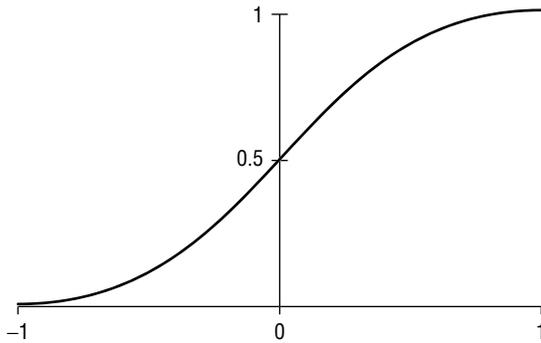
Figure 5-4: Sigmoid function

The sharpness of the curve could also be altered if required, but for most applications a straight function is fine.

Multilayer Perceptrons

The problem with single-layer perceptrons is that they are linearly separable. The output is either one value or another.

If you think of an AND gate in logic theory, there is only one outcome if you have two inputs, as shown in Table 5-1.

Table 5-1: AND Gate Output Table

INPUT	OUTPUT
Off and On	Off
On and Off	Off
Off and Off	Off
On and On	On

The perceptron would be fashioned as shown in Figure 5-5.
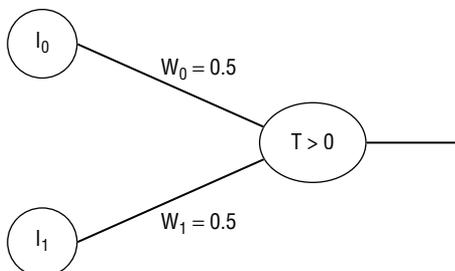
Figure 5-5: AND gate perceptron

The network output equation would be the following:

$$Output = \begin{cases} 1 \; if \; (W00 \times I0) + (W01 \times I1) > 0 \\ 0 \; \text{Otherwise....} \end{cases}$$

So far, I've covered the processing of one perceptron. Artificial neural networks have many interconnected neurons, each with its own input, output, and activation function.

For most machine learning functions, artificial neural networks are used for solving problems of a nonlinear fashion. Many problems cannot be solved in a purely linear fashion, so using a single-layer perceptron for this kind of problem solving was never worth considering. If you think of an XOR gate (Exclusive OR) with the input types shown in Table 5-2, you could easily think of the network shown in Figure 5-6.

Table 5-2: Exclusive OR Output Table

INPUT	OUTPUT
Off and On	On
On and Off	On
Off and Off	Off
On and On	Off
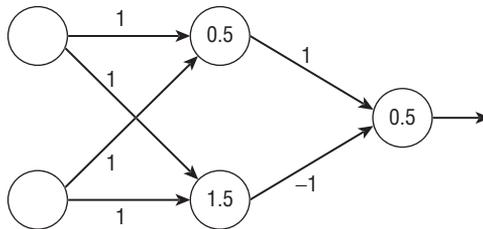
Figure 5-6: XOR gate network

Multilayer perceptrons have one or more layers between the input nodes and the eventual output nodes. The XOR example has a middle layer, called a hidden layer, between the input and the output (see Figure 5-7). Although you and I know what the outputs of an XOR gate would be (I've just outlined them in the table), and we could define the middle layer ourselves, a truly automated learning platform would take some time.
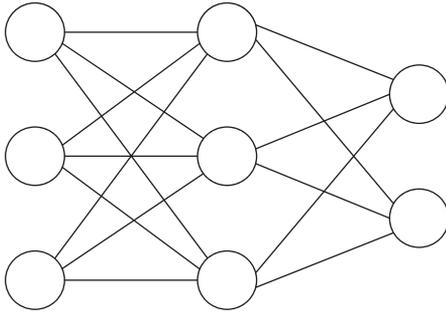
Figure 5-7: Multilayer perceptron with one hidden layer

The question is, what happens in the hidden layer? Going back to the XOR example for a moment, you can see the two input nodes with their values. These would then be fed to the hidden layer, and the input is dependent on the output of the input layer.

This is where the neural network becomes useful. You can train the network for classification and pattern recognition, but it does require training. You can train an artificial neural network by unsupervised or supervised means.

The issue is that you don't know what the weight values should be for the hidden layer. By changing the bias in the Sigmoid function, you can vary the output layer, an error function can be applied, and the aim is to get the value of the error function to a minimum value.

I described the threshold function within the perceptron previously in the chapter, but this isn't suitable for your needs. You need something that is continuous and differentiable. With the bias option implemented in the Sigmoid function, each run of the network refines the output and the error function. This leads to a better-trained network and more reliable answers.

Back Propagation

Within the multilayer perceptron is the concept of *back propagation*, short for the "backward propagation of errors." Back propagation calculates the gradients and maps the correct inputs to the correct outputs.

There are two steps to back propagation: the propagation phase and the updating of the weight. This would occur for all the neurons in the network.

If you were to look at this as pseudocode—assuming an input layer, single hidden layer, and an output layer—it would look like this:

```
initialize weights in network (random values)

while(examples to process)
  for each example x
    prediction = neural_output(network, x)
```

```
    actual = trained-output(x)
    error is (prediction – actual) on output nodes

backwardpass:
  compute weights from hidden layer to the output layer
  compute weights from input layer to hidden layer
  update network weights
  until all classified correctly against training data
  return finalized network
```

Propagation happens, and the training is input through the network and generates the activations of the output. It then backward propagates the output activations and generates deltas of all the output and hidden layers of the network based on the target of the training pattern.

In the second phase, the weight update is calculated by multiplying the output delta and input activation. This gives you the gradient weight. The percentage ratio is then subtracted from the weight. The second part is done for all the weight axons in the network.

The percentage ratio is called the *learning rate*. The higher the ratio, the faster the learning. With a lower ratio you know the accuracy of the learning is good.

NOTE I appreciate that it's difficult to grasp mathematical concepts on neural networks in a book that focuses on the practical aspects of getting machine learning up and running quickly. This overview gives a very general idea of how they work. The main concepts of input and output layers, perceptrons, and the notion of forward and backward propagation provide a good, although simple, grounding in the thought process.

Data Preparation for Artificial Neural Networks

For creating an artificial neural network, it's worth using a supervised learning method. However, this requires some thought about the data that you are going to use to train the network.

Artificial neural networks work only with numerical data values. So, if there are normalized things with text values, they need to be converted. This isn't so much an issue with the likes of gender, where the common output would be Male = 0 and Female = 1, for example. Raw text wouldn't be suitable, so it will either need to be tidied up, hashed to numeric values, or removed from the test data.

As with all data strategies, it's a case of thinking about what's important and what data you can live without.

As more variables increase in your data for classification, you will come across the phenomenon called "the curse of dimensionality." This is when added variables increase the total volume of training data required to get reasonable

results and insight. So, when you are thinking of adding another variable, make sure that you have enough training data to cover eventualities across all the other variables.

Although neural networks are pretty tolerant to noisy data, it's worth trying to ensure that there aren't large outliers that could potentially cause issue with the results. Either find and remove the wayward digits or turn them into missing values.

Artificial Neural Networks with Weka

The Weka framework supports a multilayer perceptron and trains it with the back propagation technique I just described. In this walkthrough, you create some data and then generate a neural network.

Generating a Dataset

My dataset is going to contain classifications for different types of vehicles. I'm first going to create a Java program that generates some random, but weighted, data to give us four types of vehicles: bike, car, bus, and truck.

Here's the code listing:

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;

public class MLPData {

    private String[] classtype = new String[] { "Bike", "Car", "Bus",
"Truck" };

    public MLPData() {

        Random rand = new Random(System.nanoTime());

        try {
            BufferedWriter out = new BufferedWriter(new FileWriter(
                    "vehicledata.csv"));
    out.write("wheels,chassis,pax,vtype\n");
            for (int i = 0; i < 100; i++) {
                StringBuilder sb = new StringBuilder();
                switch (rand.nextInt(3)) {
                case 0:
                    sb.append((rand.nextInt(1) + 1) + ","); // num of
wheels
```

```
                        sb.append((rand.nextInt(1) + 1) + ","); // chassis
length
                        sb.append((rand.nextInt(1) + 1) + ","); // passenger
number
                        sb.append(classtype[0] + "\n");
                        break;
                    case 1:
                        sb.append((rand.nextInt(2) + 4) + ","); // num of
wheels
                        sb.append((rand.nextInt(4) + 1) + ","); // chassis
length
                        sb.append((rand.nextInt(4) + 1) + ","); // passenger
number
                        sb.append(classtype[1] + "\n");
                        break;
                    case 2:
                        sb.append((rand.nextInt(6) + 4) + ","); // num of
wheels
                        sb.append((rand.nextInt(12) + 12) + ","); // chassis
length
sb.append((rand.nextInt(30) + 10) + ","); // passenger number

                        sb.append(classtype[2] + "\n");
                        break;
                    case 3:
                        sb.append("18,"); // num of wheels
                        sb.append((rand.nextInt(10) + 20) + ","); // chassis
length
                        sb.append((rand.nextInt(2) + 1) + ","); // passenger
number
                        sb.append(classtype[3] + "\n");
                        break;
                    default:
                        break;
                    }
                    out.write(sb.toString());
                }
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public static void main(String[] args) {
            MLPData mlp = new MLPData();

        }
    }
```

When run, the preceding code creates a CSV file called `vehicledata.csv`. Start by creating 100 rows of output:

```
4,2,4,Car
9,20,25,Bus
5,14,18,Bus
5,2,1,Car
9,17,25,Bus
1,1,1,Bike
4,4,2,Car
9,15,36,Bus
1,1,1,Bike
5,1,4,Car
4,2,1,Car
```

As discussed previously, you need to perform a fair amount of training to make the neural network accurate in its predictions.

Loading the Data into Weka

Open the Weka toolkit and select the Explorer function to display the Explorer shown in Figure 5-8.
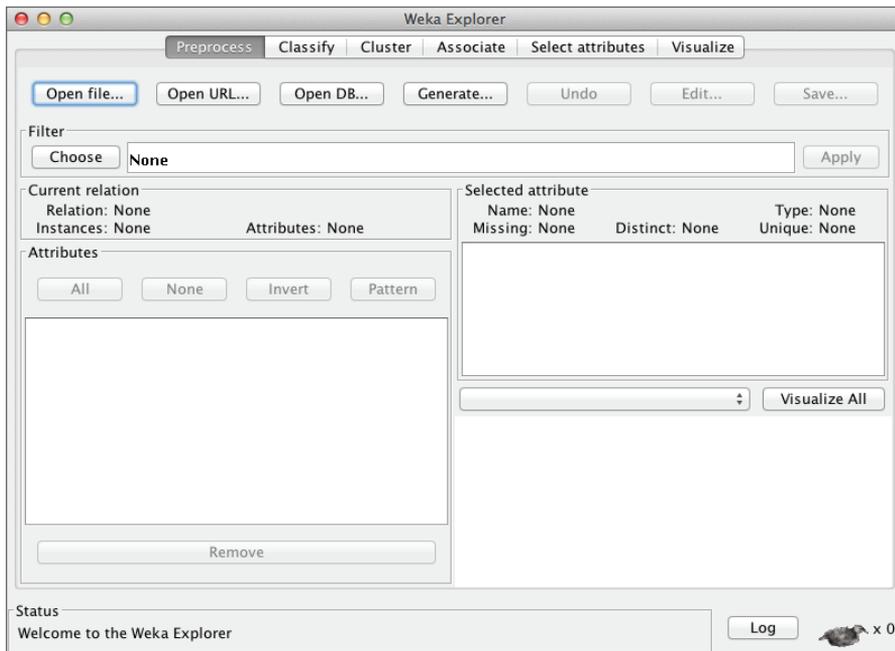
Figure 5-8: Weka Explorer

You're going to import the CSV file that's been created. Make sure that the Preprocess window is selected and then click the Open File button and select the `vehicledata.csv` file. Don't forget to change the File Format drop-down menu from `.arff` to `.csv`, as shown in Figure 5-9.
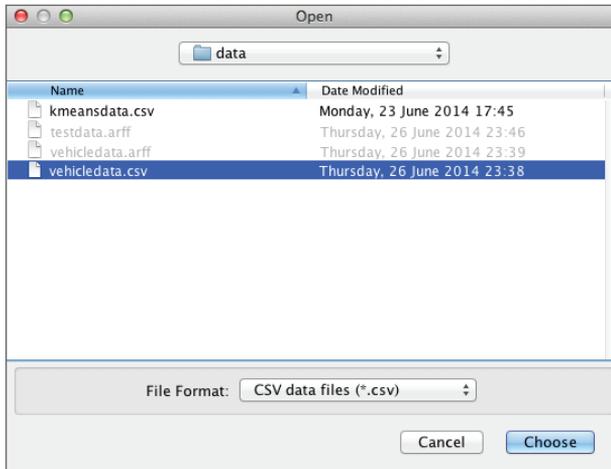
Figure 5-9: Weka File dialog box

You see the data loaded with the basic representation of the relation and attribute information.

Configuring the Multilayer Perceptron

The neural network function of Weka comes with its own graphic user interface. When run, you can see the graphical representation of the neural network.

Click the Classify panel. Where the default classifier is ZeroR, click Choose and change it to MultilayerPerceptron (see Figure 5-10), which is in the Functions branch of the tree listing.

You see the classifier change to MultilayerPerceptron with a lot of options next to it. If you click that line, a window of options opens, as shown in Figure 5-11.
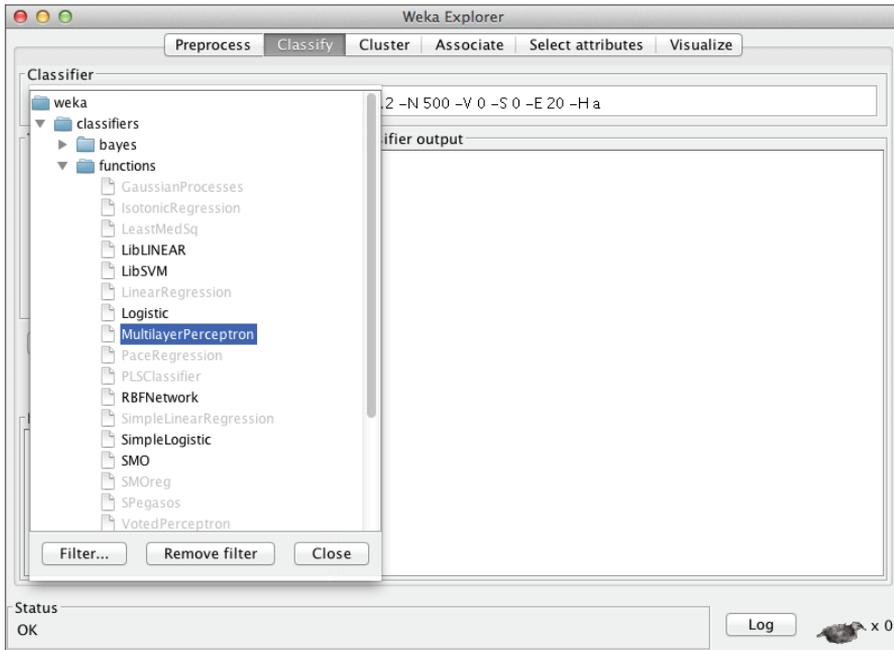
Figure 5-10: Changing the classifier
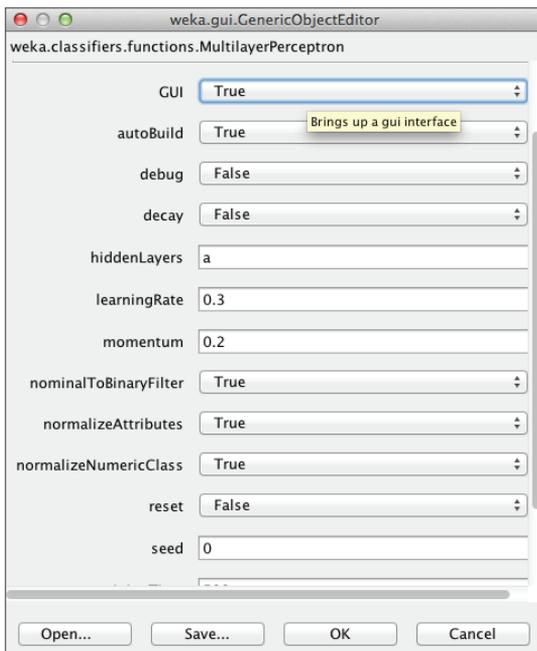
Figure 5-11: Options dialog box for MultilayerPerceptron

Change the GUI setting to True. This setting makes the neural network display in a graphic form; the display is also interactive, and you can change the network. If the GUI setting is set to False, then Weka generates the network for you without your intervention.

Although this version of the MultilayerPerceptron converts and handles your nominal values for you, it's still prudent to take the time to ensure that your data is prepared properly. The network autobuilds by default. If you want to create your own, then you can turn this off and craft the network by hand.

There are a few values that are worth keeping an eye on before you let the network do its training.

Learning Rate

The amount the weights are updated is defaulted at 0.3. If that seems a little heavy or too light, then you can adjust as desired.

Hidden Layers

You can define how many hidden layers the neural network will have. By default, Weka builds four (attributes and classes/2) (set to "a"), but you can also have just the attributes ("i"), the classes ("o") and the attributes and classes complete ("t").

Training Time

The number of epochs through which Weka iterates during training is set to 500. The higher the number, the lower the error rate will be. As you'll see in a moment, this can give varying results in the output.

When you are happy with the options, you can click OK and go back to the Classify window.

Training the Network

You have to do a few runs of neural networks to find the sweet spot where the network is coming up with good classifications. With 100 rows of data, you're not going to be solving much of any worth; regardless, it gives you an idea of how it works.

Make sure the test options are set to use the whole training set. The cross-validation is fine, but it ends up running the training through all ten folds, and that can get time consuming when you just want to test. Click Start, and the neural network window shown in Figure 5-12 displays.
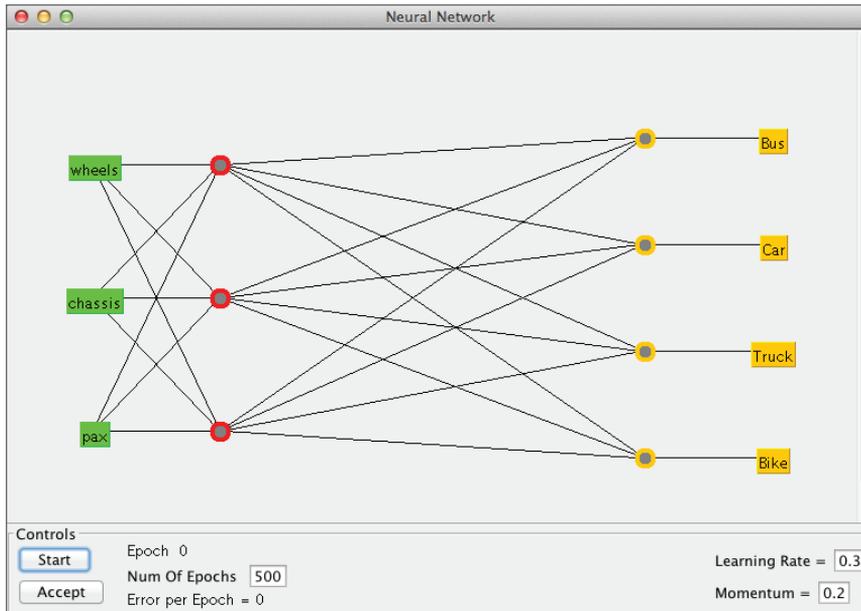
Figure 5-12: Neural network GUI window

Click Start, and you see the epoch count rise and the error rate decrease. If you click Accept by accident, then no data will have been classified and the results will be wrong.

After the neural network has run, click the Accept button and you will be returned to the classification output screen.

The full classifier output gives the output for the hidden layer nodes. Nodes 0, 1, 2, and 3 and the four nodes on the right side of Figure 5-12 are the output connections are the output connections. The class attributes for classification are shown as bike, car, bus, or truck on the right hand side of the neural network output (refer to Figure 5-12).

```
Sigmoid Node 0
    Inputs    Weights
    Threshold    0.018993883149676594
    Node 4    -0.04038638643499096
    Node 5    0.0065483634965212145
    Node 6    -0.03873854654480489
Sigmoid Node 1
    Inputs    Weights
    Threshold    -0.0451840582741909
    Node 4    -0.002851224687941599
    Node 5    -0.012455737520358182
    Node 6    -0.0491382673800735
```

```
Sigmoid Node 2
    Inputs    Weights
    Threshold    -0.010479295335213488
    Node 4    0.02129170595398988
    Node 5    0.02877248387280648
    Node 6    -0.001813155428890656
Sigmoid Node 3
    Inputs    Weights
    Threshold    0.02680212410425596
    Node 4    0.006810392393573984
    Node 5    -0.04968676115705444
    Node 6    -0.015015642691489917
```

Nodes 4, 5, and 6 comprise the hidden layer that takes the input from the input attributes for wheels, chassis, and passenger count.

```
Sigmoid Node 4
    Inputs    Weights
    Threshold    0.011850776365702677
    Attrib wheels    0.0429940506718635
    Attrib chassis    -0.035625493582980464
    Attrib pax    -0.021284810000068835
Sigmoid Node 5
    Inputs    Weights
    Threshold    0.011165074786232076
    Attrib wheels    -0.018370069737576836
    Attrib chassis    -0.030938315802372954
    Attrib pax    0.01567513412449774
Sigmoid Node 6
    Inputs    Weights
    Threshold    -0.04753959806853169
    Attrib wheels    -0.00211881373779247
    Attrib chassis    0.040431974347463484
    Attrib pax    -0.017943250444400316
```

Each node has the input type and the weight values of the corresponding input node.

The summary shows how many instances have been correctly classified, along with other values for the error data if it has occurred.

In the last section, you can see how the classification counts added up in the Confusion Matrix.

```
=== Confusion Matrix ===

  a  b  c  d   <-- classified as
 33  0  0  0 |  a = Bus
  0 27  0  0 |  b = Car
  0  0 20  0 |  c = Bike
  0  0  0 20 |  d = Truck
```

Altering the Network

With the GUI option set to True, you can add nodes and also remove input paths to parts of the hidden layer. If you make any changes you need to retrain the neural network; the updated network will display in the GUI.

Which Bit Is Which?

Working from left to right on the GUI, you see the raw input nodes as labels in the yellow boxes. Red dots are the hidden layer nodes, and the orange dots are the output nodes. The orange labels are the classes with which the orange dot nodes are associated.

Adding Nodes

You can add a new node by clicking the GUI. The red dot appears to signify a hidden layer node. It won't be connected to anything, unless you have already selected nodes in the GUI.

Connecting Nodes

With the node selected, you can click on another node to see the connection being made.

Removing Connections

To remove a connection, select one of the connected nodes and then right-click the other connected node. The connecting line disappears.

Removing Nodes

Right-clicking a node removes it and all the connections to it. Be careful to make sure that there aren't any other selected nodes, otherwise they, and their connections, will be removed, too.

Increasing the Test Data Size

Within the `for` loop of the `MLPData.java` program you created earlier in the chapter, change the loop count from 100 rows to 100,000 rows. Go back to the Preprocess window and load the new CSV file. It might take some time to load.

Now, go back to the Classify window and rerun the neural network. When the GUI window opens, you see the network looks the same as before in terms of the hidden layers. Where you had 500 epochs running against the 100 rows of data, you now have the same epoch number against all 100,000 rows of training data.

Click Start and the training begins. You'll notice a difference in response time from the GUI as it trains all 100,000 rows. The main thing to look at is the errors per epoch; the number keeps reducing to the point where you get minute changes per 100 to 200 epochs. By the time the training has finished, you will have a very accurate training model.

All this comes at a price of memory, though. My training set took more than two minutes:

```
Time taken to build model: 124.52 seconds
```

Two minutes isn't a huge amount of time in the grand scheme of things, but as I previously mentioned in regard to gathering data for neural networks, adding more variables gives the curse of dimensionality.

The more rows you can use for training, the better the prediction results will be. There is a point in time to figure out when there's too much training data against the errors per epoch. It takes some practice (and everyone's data is different, so there's no hard or fast rule), and it's a case of experiment, measure, and try again.

Implementing a Neural Network in Java

With the Weka API, you can build a neural network with the same multilayer perceptron that Weka uses within the GUI.

Create the Project

Select File ➪ New ➪ Java Project and call it MLPProcessor, as shown in Figure 5-13.

You need to tell Eclipse where the Weka API is; it's called weka.jar. On Mac OS X machines, Weka is usually installed within the Applications directory. The location on Windows machines varies depending on the specific operating system and Weka installation. In most cases it will be /Program Files (x86)/ Weka-3-6/weka.jar.

With the WekaCluster project selected, select File ➪ Properties and look for the Java Build Path. Then click the Libraries tab. Add the external jar file by clicking Add External JARs, then in the file dialog box find the weka.jar file, as shown in Figure 5-14.
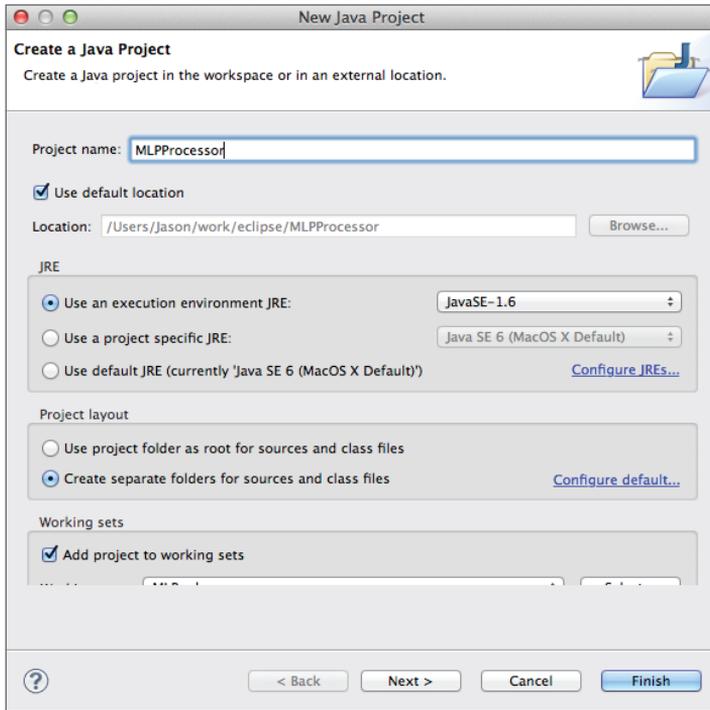
Figure 5-13: Eclipse New Project dialog box
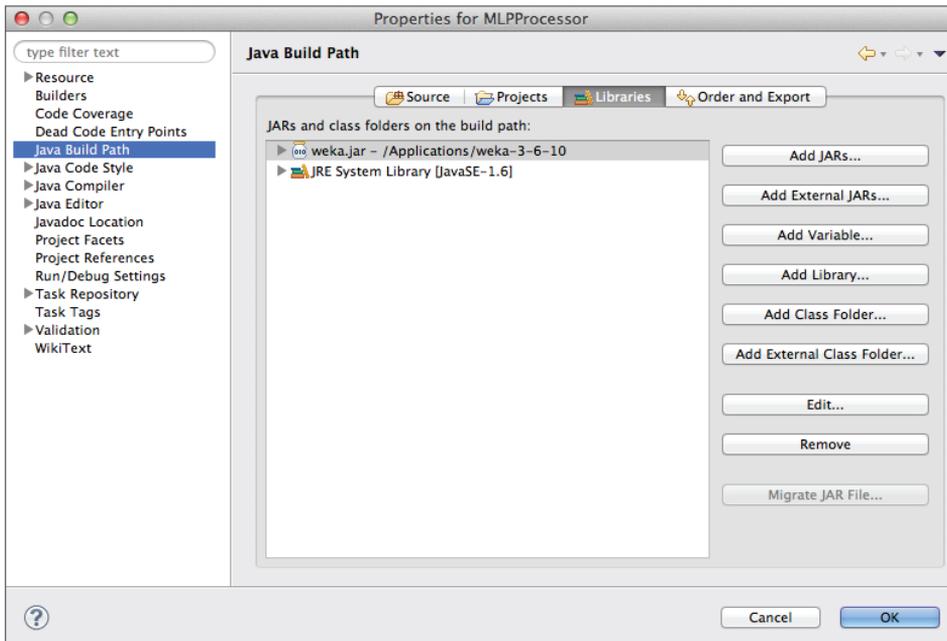
Figure 5-14: Adding external jars

The last thing to do is create a new class called `MLPProcessor.java` (using File ⇨ New ⇨ Class), as shown in Figure 5-15.
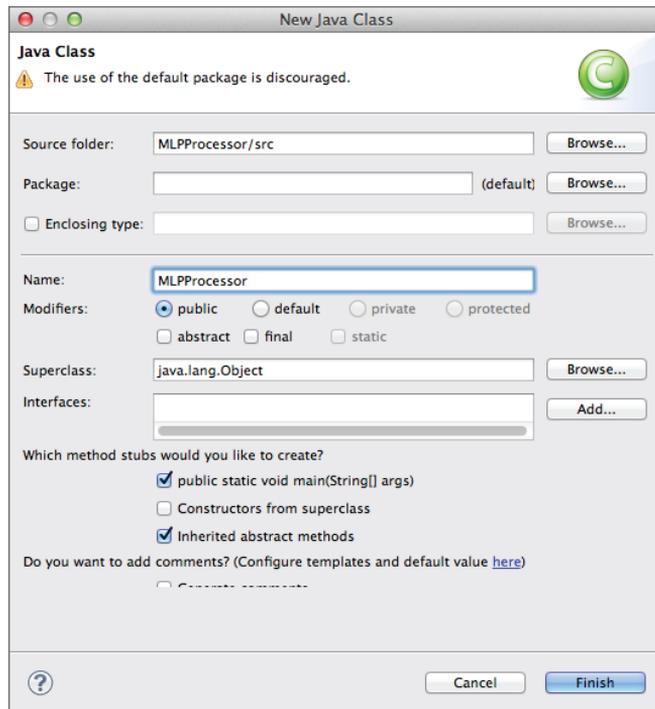
Figure 5-15: Creating a new class file

The Code

The actual Java is straightforward. You're going to do the following:

- Open the training data `.arff` file.
- Create a MultilayerPerceptron and set the same options as the Weka GUI example.
- Build the classifier.
- Load in some test data.
- Run an evaluation test with the test data against the trained data.

You need to create a small test data file to test against the model. In a text file called `testdata.arff` enter the following:

```
@relation vehicledata

@attribute wheels numeric
@attribute chassis numeric
```

```
@attribute pax numeric
@attribute vtype {Bus,Car,Truck,Bike}

@data
18,25,2,Truck
8,21,24,Bus
18,27,2,Truck
1,1,1,Bike
7,23,21,Bus
18,20,1,Truck
8,16,30,Bus
18,28,2,Truck
7,18,36,Bus
8,21,27,Bus
5,2,4,Car
18,28,1,Truck
5,1,1,Car
1,1,1,Bike
18,27,1,Truck
5,1,1,Car
6,15,38,Bus
7,21,38,Bus
18,20,2,Truck
1,1,1,Bike
18,28,2,Truck
18,24,2,Truck
18,20,1,Truck
1,1,1,Bike
5,17,18,Bus
18,27,1,Truck
4,4,3,Car
18,21,1,Truck
5,2,3,Car
4,3,3,Car
18,23,1,Truck
5,20,30,Bus
5,3,3,Car
18,28,1,Truck
5,3,1,Car
9,13,19,Bus
1,1,1,Bike
18,26,2,Truck
```

After you've created the test file, use the following code:

```java
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import weka.classifiers.Evaluation;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.core.Instances;
```

```java
import weka.core.Utils;

public class MLPProcessor {

    public MLPProcessor() {
        try {
            FileReader fr = new FileReader("vehicledata.arff");

            Instances training = new Instances(fr);

            training.setClassIndex(training.numAttributes() -1);

            MultilayerPerceptron mlp = new MultilayerPerceptron();
            mlp.setOptions(Utils.splitOptions("-L 0.3 -M 0.2 -N 500 -V 0
-S 0 -E 20 -H 4"));


            mlp.buildClassifier(training);

            FileReader tr = new FileReader("testdata.arff");
            Instances testdata = new Instances(tr);
            testdata.setClassIndex(testdata.numAttributes() -1);

            Evaluation eval = new Evaluation(training);
            eval.evaluateModel(mlp, testdata);

System.out.println(eval.toSummaryString("\nResults\n*******\n", false));

            tr.close();
            fr.close();

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        MLPProcessor mlp = new MLPProcessor();
    }

}
```

The actual neural network is taken care of within three lines of code. Create the MultilayerPerceptron, set which class you want to determine, and then build the classifier. The rest of the code is loading the training and test data in.

Converting from CSV to Arff

CSV files don't contain the data that Weka needs. You could implement the `CSVLoader` class, but I prefer to know that the `.arff` data is ready for use. It also makes it easier for others to decode the data model if they need to.

From the command line, you can convert the data from a `.csv` file to `.arff` in one command.

```
java -cp /Applications/weka-3-6-10/weka.jar weka.core.converters.
CSVLoader vehicledata.csv > vehicledata.arff
```

If you inspect the `.arff` file, you see the attribute information set up for you.

```
@relation vehicledata

@attribute wheels numeric
@attribute chassis numeric
@attribute pax numeric
@attribute vtype {Bus,Car,Truck,Bike}

@data
6,20,39,Bus
8,23,11,Bus
5,3,1,Car
4,3,4,Car
5,3,1,Car
4,18,37,Bus
18,23,2,Truck
```

Running the Neural Network

The code listing doesn't include any output messages while it's running, with the exception of the output of the evaluation. I say this because the training data could have 100,000 rows in it, and it's going take a few minutes to run.

Run the class with Run ⇨ Run from Eclipse, and it starts to generate the model. After a while, you see the output from the evaluation.

```
Results
======

Correctly Classified Instances          38                100      %
Incorrectly Classified Instances         0                  0      %
Kappa statistic                          1
Mean absolute error                      0.0003
Root mean squared error                  0.0004
Relative absolute error                  0.0795 %
Root relative squared error              0.0949 %
Total Number of Instances               38
```

Instances can be easily classified by using the multilayer perceptron `classifyInstance()` method, which takes in a single `Instance` class and outputs a numeric representation of the result. This result corresponds to your output class in the `.arff` training file.

Summary

Perhaps it's my English nature, but I find it slightly ironic that I could spend so few pages on a subject so massive. It's about the brain, neurons, and all that kind of stuff!

Seriously though, this chapter should have given you a basic grounding on how neural networks work, including a couple of examples in Weka and Java.

The key to a successful neural network project comes down to the data preparation. Too little preparation and the network won't predict right; too much and you hit memory issues. It's about finding the right set of data, the right quantity, and the right training method.

Association Rules Learning

Among the machine learning methods available, association rules learning is probably the most used. From point-of-sale systems to web page usage mining, this method is employed frequently to examine transactions. It finds out the interesting connections among elements of the data and the sequence (behaviors) that led up to some correlated result.

This chapter describes how association rules learning methods work and also goes through an example using Apache Mahout for mining baskets of purchases. This chapter also touches on the myth, the reality, and the legend of using this type of machine learning.

Where Is Association Rules Learning Used?

The retail industry is tripping over itself to give you, the customer, offers on merchandise it *thinks* you will buy. In order to do that, though, it needs to know what you've bought previously and what other customers, similar to you, have bought. Brands such as Tesco and Target thrive on basket analysis to see what you've purchased previously. If you think the amount of content that Twitter produces is big, then just think about point-of-sale data; it's another world. Some

supermarkets fail to adopt this technology and never look into baskets, much to their competitive disadvantage. If you can analyze baskets and act on the results, then you can see how to increase bottom-line revenue.

Association rules learning isn't only for retail and supermarkets, though. In the field of web analytics, association rules learning is used to track, learn, and predict user behavior on websites.

There are huge amounts of biological data that are being mined to gain knowledge. Bioinformatics uses association rules learning for protein and gene sequencing. It's on a smaller scale compared to something like computational biology, as it homes in on specifics compared to something like DNA. So, studies on mutations of genomes are part of a branch of bioinformatics that's probably working with it.

Web Usage Mining

Knowing which pages a user is looking at and then suggesting which pages might be of interest to the user is commonplace to keep a website more compelling and "sticky." For this type of mining, you require a mechanism for knowing which user is looking at which pages; the user could be identified by a user session, a cookie ID, or a previous user log in where sites require users to log in to see the information.

If you have access to your website log files then there is opportunity for you to mine the information. Many companies use the likes of Google Analytics as it saves them mining logs themselves, but it's worthwhile doing your own analysis if you can.

The basic log file, for example, has information against which you could run some basic association rules learning. Looking at the Apache Common Log Format (CLF) you can see the IP address of the request and the file it was trying to access:

```
86.78.88.189 - thisuserid [10/May/2014:13:55:59 -0700] "GET /
myinterestingarticle.html HTTP/1.0" 200 2326
```

By extracting the URL and the IP address, the association rules could eventually suggest related content on your site that would be of interest to the user.

Beer and Diapers

It is written on parchment dating back many years, the parable of the beer and the diapers (or nappies, as I will always call them).

Tis written on this day that the American male of the species would frequent the larger markets of super the day prior to the Sabbath. Newly attired with sleeping eyes and new child, said American male would buy device of child's dropping catching of cloth and safety pin, when, lo, he spotteth the beer of delights full appreciating he shall not make it to the inn after evensong, such be his newly acquired fatherly role. And Mart of Wal did look upon this repeated behavior and move the aisles according to the scriptures of the product of placement, thus increasing the bottom line.

This story has been preached by marketing departments the world over (possibly not in the style presented here), and it's been used in everything from keynotes to short talks, from hackathons to late night codejams. However, it's a case of fact mixed with myth.

When he was CEO of a company called Mindmeld, Thomas Blischok was also on the panel of a webcast on the past, present, and future of data mining and had managed the study on data that spawned the beer and nappies story. The study went back to the early 1990s when his team was looking at the basket data for Osco Drug. They did see a correlation on basket purchases between 5:00 and 7:00 p.m. and presented the findings to their client.

After that point, there's some confusion about where the story actually goes. Many versions are basically myth and legend, they've generated great chat and debate around the water cooler for years and will continue to do so.

The myth has now been superseded by the privacy-fearing consumer story known as the "Target can predict whether I'm pregnant or not" scenario. I, for one, have two reasons why Target could never predict my outcome: I've never shopped there, and it's biologically impossible. You don't need a two-node decision tree to figure that out. (Read Chapter 3, "Working with Decision Trees" for more information on that subject.)

> **NOTE** For the full story on the Beer and Diapers legend, have a look at D.J. Power's article from November 2002 at `http://www.dssresources.com/newsletters/66.php`. The myth will live on forever, I'm sure (especially if you're in marketing), and it makes for good reading.

How Association Rules Learning Works

The basket analysis scenario is a good example to explain with, so I'll continue with it. Consider the following table of transactions:

TRANSACTIONID	PRODUCT1	PRODUCT2	PRODUCT3	PRODUCT4
1	True	True	False	False
2	False	False	True	False
3	False	False	False	True
4	True	True	True	False
5	False	True	False	False

This is essentially an item set of transactions with a transaction ID and the products (could be milk, nappies, beer, and beans, for example).

Ultimately you're looking for associations in the products. For example if a customer buys products 1 and 2, he is likely to buy product 4.

So a set of items:

$$I=\{product1,product2,product3,product4\}$$

And a set of transactions:

$$T=\{product1,product2,product4\}$$

Each transaction must have a unique ID for the rule to glean any information. Also, it's worth noting that this sort of rule needs hundreds of transactions before it starts to generate anything of any value to you. The larger the transaction set, the better the statistical output will be and the better the predictions will be.

The rule is defined as an implication, what you're looking at is the following:

$$X,Y \subseteq I \text{, where } X \cap Y = (/)$$

In plain English, what you're saying is X and Y are a subset of the item set in the intersection of X and Y.

They take on the form of a set as items denoted as X and Y. In scary math books, it will look like this:

$$X \Rightarrow Y$$

The X denotes the items set before (or left of) the rule, called the *antecedent*, and the Y is the item set after (or right of) the rule, called the *consequent*.

Getting back to the products in the basic item set:

$$I = \{product1,product2,product3,product4\}$$

The true/false statements show whether the item is in that basket transaction or not.

To get the true picture of how the rules work, you need to investigate a little further into the concepts of support, confidence, lift, and conviction.

Support

Support is defined as the proportion of items in the data that contain the item set. It's written like so:

$$\text{Supp}(X) = \frac{transactions\_containing\_X}{total\_number\_of\_transactions}$$

If you were to take transaction number 1, as an example, you'd have the following equation:

$$\text{supp}(X) = \frac{\{product1, product2\}}{5} = 0.2$$

The item set appears only once in the transaction log, and there are five transactions, so the support is 1/5, which is 0.2.

Confidence

Confidence in the rule is measured as

$$conf(X \Rightarrow Y) = \text{supp}(X \cup Y) / \text{supp}(X)$$

What you're defining here is the proportion of transactions containing set X, which also contain Y. This can be interpreted as the probability of finding the right-hand side of transactions under the condition of finding them on the left-hand side.

To use an analogy, think of parimutuel betting. All bets are placed together in a pool, and after the race has finished the payout is calculated based on the total pool (minus the commission to the agent). For example, assume there are five horses racing and bets have been placed against each one:

HORSE NUMBER	BET
1	$40.00
2	$150.00
3	$25.00
4	$40.00
5	$30.00

The total pool comes to \$285, and after the event is run and the winner is confirmed the payout can be calculated. Assuming that horse number 4 was the winner the calculation would be

Pool size after commission = \$285 x (1 – 0.15) = \$242.25.
Payout per \$1 on outcome 4 = \$6.05 per \$1 wagered.

Lift

Lift is defined as the ratio of the observed if the X and Y item sets were independent. It's written as

$$Lift(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X) \times \text{supp}(Y)}$$

Conviction

Finally there's *conviction*, which is defined as the ratio of the expected frequency that X occurs without Y:

$$conv(X \Rightarrow Y) = \frac{1 - \text{supp}(Y)}{1 - conf(X \Rightarrow Y)}$$

Defining the Process

Association rules are defined to satisfy two user-defined criteria, a minimum support value and a minimum confidence. The rules generation is done in two parts.

Firstly the minimum support is applied to all the frequent item sets in the database (or file or data source). The frequent item sets along with the minimum confidence are used to form the rules.

Finding frequent item sets can be hard; it involves trawling through all the possible item combinations in the item sets. The number of possible item sets is the "power set" over the item set.

For example, if you have the following:

$$I = \{p1, p2, p3\}$$

then the power set of *I* would be this:

$$\{\{p1\}, \{p2\}, \{p3\}, \{p1, p2\}, \{p1, p3\}, \{p2, p3\}, \{p1, p2, p3\}\}$$

Notice that the empty set ({}) is omitted in the power set; this formulation gives you a size of $2^n - 1$, where n is the number of items. A small increase in the number of items causes the size of the power set to increase enormously;

therefore this method is quite hungry in memory when using something like the Apriori algorithm. Obviously, the power set of all combinations of baskets does not occur, and the calculation will be based only on those basket combinations that do. Nonetheless, it is still very expensive in time and memory to run calculations based on this method.

Algorithms

There are several algorithms used in association rule learning that you'll come across; the two described in this section are the most prevalent.

Apriori

Using a bottom-up approach, the Apriori algorithm works through item sets one at a time. Candidate groups are tested against the data; when no extensions to the set are found, the algorithm will stop. The support threshold for the example is 3.

If you consider the following item set:

{1,2,3,4}

{1,3,4}

{1,2}

{2,3,4}

{3,4}

{2,4}

First, it counts the support of each item:

{1} = 3

{2} = 5

{3} = 4

{4} = 5

The next step is to look at the pairs:

{1,2} = 2

{1,3} = 2

{1,4} = 2

{2,3} = 2

{2,4} = 3

{3,4} = 4

As {1,2}, {1,3},{1,4}, and {2,3} are under the chosen support threshold you can reject them from the triples that are in the database. In the example, there is only one triple:

{2,3,4} = 1 (we've discounted one from the {1,2} group).

From that deduction, you have the frequent item sets.

FP-Growth

The Frequent Pattern Growth algorithm (FP-Growth) works as a tree structure (called an FP-Tree). It creates the tree by counting the occurrences of the items in the database and storing them in a header table.

In a second pass, the tree is built by inserting the instances it sees in the data as it goes along the header table. Items that don't meet the minimum support threshold are discarded; otherwise they are listed in descending order.

You can think of the FP-Growth algorithm like a graph, as is covered in the chapter about Bayesian Networks (Chapter 4). With a reduced dataset in a tree formation, the FP-Growth algorithm starts at the bottom—the place with the longest branches—and finds all instances of the given condition. When no more single items match the attribute's support threshold, the growth ends and then it works on the next part of the FP Tree.

Mining the Baskets—A Walkthrough

This walkthrough uses Mahout to work on the data using the Apriori algorithm and to get a set of results based on the historical basket contents.

The nice thing about this sort of project is that it can be easily ported to your own e-commerce platform if you're operating one. The keys are to manage how to get the data out of the database for processing and then craft a way of getting the insight back in.

Downloading the Raw Data

Instead of using precious time to create data, I've used a dataset from Dr. Tariq Mahmood's website, which contains a test data file for basket analysis.

You can download the CSV file from his site: `https://sites.google.com/a/nu.edu.pk/tariq-mahmood/teaching-1/fall-12---dm/marketbasket.csv?attredirects=0&d=1`.

Each column represents a product type and each row represents a basket transaction from the store.

ASPIRIN	SWEET POTATOES	CANNED TOMATOES	CHUNKY PEANUT BUTTER	WOOD POLISH
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	false	false	false	false
false	true	false	false	false
false	true	false	false	true
false	false	false	false	false
false	false	false	false	false

The data comprises 1,361 transactions against 302 products, which equals more than 411,000 transactions to mine through. Notice there are no quantities to deal with—just a notification of whether the item was purchased or not (true or false).

Setting Up the Project in Eclipse

There are a few classes you need to create in order to get association rules learning to work from a code point of view. It's not so much a case of creating MapReduce functions or anything specific for Mahout to work from; it is more a matter of data preparation and being able to read the output after the mining has taken place.

Create a new project called `ARLearning` in Eclipse using the File ➪ New ➪ Java Project option as shown in Figure 6-1.
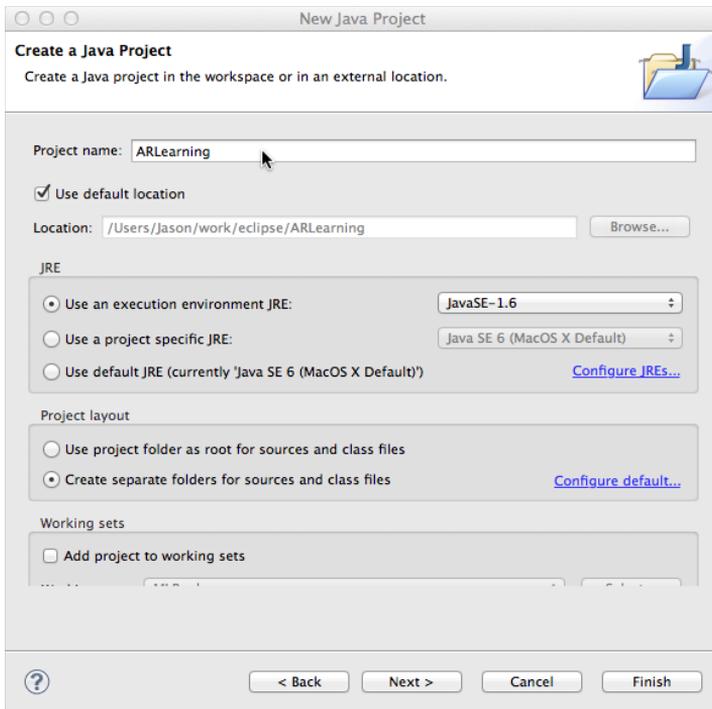
Figure 6-1: A new Java project

Within the project properties you need to add the locations for the core Hadoop library, the MySQL JDBC driver, and the core Mahout library, as shown in Figure 6-2.

NOTE **If you don't have the MySQL JDBC driver, you can download it from** `http://dev.mysql.com/downloads/connector/j/.`

Setting Up the Items Data File

I'm going to store the product data (the column names) in a database table; you'll need to refer to these later when you have some results from Mahout.

From the command line, type in the following MySQL commands:

```
mysqladmin -u root create apriori
mysql -u root apriori
```

The table will only have two fields: one for the id and another for the product name:

```
mysql> create table products(
    -> id int(11) not null default -1,
    -> productname varchar(100) not null default "");
```
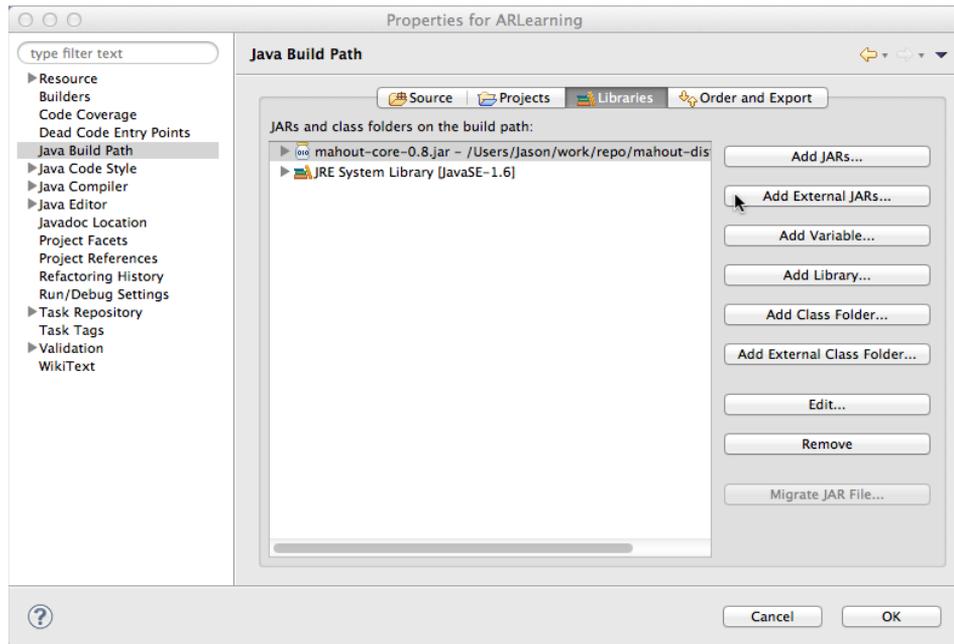

Figure 6-2: Adding the required JAR files

The next task is to import the product titles into the database. I've written a small Java program to read in the `.csv` file and extract the first line only, iterating each column and adding the product name into the database table. It's a very straightforward piece of code, and it works for this example. Use the following code, changing the file path to the place you downloaded or prepared the `rawdata.csv` file:

```java
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
```

```java
import java.sql.SQLException;

public class ExtractProductNames {

    static {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    public ExtractProductNames() {
        try {
            Connection con = DriverManager.getConnection("jdbc:mysql://
localhost/apriori","root","");
            PreparedStatement pstmt = con.prepareStatement("INSERT INTO
products (id, productname) VALUES (?,?)");

            BufferedReader csvfile = new BufferedReader(new
FileReader("/path/to/your/data/rawdata.csv"));
            String productsLine = csvfile.readLine();
            String[] products = productsLine.split(",");

            for(int i = 0; i < products.length; i++) {
                pstmt.clearParameters();
                pstmt.setInt(1, i);
                pstmt.setString(2, products[i].trim());
                pstmt.execute();
                System.out.println("Added: " + products[i] + " into db");
            }
            pstmt.close();
            con.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch(SQLException e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        ExtractProductNames epn = new ExtractProductNames();
    }

}
```

When you run the program, you see the output of the lines as they are added into the database table:

```
Added:   Salt into db
Added:   Green Beans into db
Added:   Flavored Ice into db
Added:   Imported Beer into db
Added:   Grits into db
Added:   Apple Jelly into db
Added:   Beef Jerky into db
Added:   Potatoes into db
Added:   Small Eggs into db
Added:   Silver Cleaner into db
```

Now it's time to look at the actual data.

Setting Up the Data

The raw `.csv` file has the values for a customer purchase as true or false, but you need to convert it to a format that Mahout is happy to process. Mahout prefers the item ID in a comma-separated format. Now that you have item IDs set up in the database, you can use those in a data file so Mahout has the item IDs for each basket.

Create a new Java class called `DataConverter` (File ➪ New ➪ Class). The following Java code reads in the remaining lines of the `.csv` file and converts the true fields into the respective ID of the product that you saved in the database:

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;

public class DataConverter {

    public DataConverter() {
        try {
            FileWriter outputWriter = new FileWriter(
                    "/path/to/your/data/output.dat");
            int txcount = 0;

            BufferedReader csvReader = new BufferedReader(new FileReader(
                    "/path/to/your/data/rawdata.csv"));

            // read the first line in but do nothing with it.
```

```
                String thisLine = csvReader.readLine();
                String[] tokens = thisLine.split(",");
                //
                int i;
                while (true) {
                    thisLine = csvReader.readLine();
                    if (thisLine == null) {
                        break;
                    }

                    tokens = thisLine.split(",");
                    i = 0;
                    boolean firstElementInRow = true;
                    for (String token : tokens) {
                        if (token.trim().equals("true")) {
                            if (firstElementInRow) {
                                firstElementInRow = false;
                            } else {
                                outputWriter.append(",");
                            }
                            outputWriter.append(Integer.toString(i));
                        }
                        i++;
                    }
                    outputWriter.append("\n");
                    txcount++;
                }
                outputWriter.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        public static void main(String[] args) {
            DataConverter dc = new DataConverter();


        }
    }
```

After you run the program, you see that a new file called `output.dat` has been created. It now contains the product IDs in each basket.

```
6,7,12,53,64,89,93,109,123,151,163,182,202,207,210,243,259,287
5,7,18,23,111,127,142,155,163,202,218,244,276,279,287,290
17
1,179,213
225
286
```

```
33
89,224,271
228
93
```

That's all you need to do in preparation. Now, you can move on to Mahout and mining the data with association rules learning.

Running Mahout

You have a couple of options here: You can run Mahout standalone without Hadoop, or you can run Mahout as a map reduce job in Hadoop. Note that you will require Mahout version 0.7, which you can download from `http://archive .apache.org/dist/mahout/0.7/` and install to a directory.

Mahout in Standalone Mode

In the directory with the raw data file, you run the following command:

```
/your/path/to/mahout/bin/mahout fpg -i output.dat -o patterns -k 10 -s 2
```

Take a look at those command-line options:

- `fpg`: You're using the Frequent Pattern Growth algorithm.
- `-i`: This is the input file you're using.
- `-k`: For every item you're mining, you want to find 10 associated items. The highest number of item associations in the basket is what you're looking for.
- `-s`: You only consider items that appear in more than two of the transactions you're working with.

Depending on the machine, it can take Mahout a few seconds to show anything onscreen from the terminal. If nothing seems to be happening, be patient. The output will look like the following:

```
14/04/29 20:15:15 INFO fpgrowth.FPGrowth: Found 3 Patterns with Least
Support 149
14/04/29 20:15:15 INFO fpgrowth.FPGrowth: Mining FTree Tree for all
patterns with 1
14/04/29 20:15:15 INFO fpgrowth.FPGrowth: Found 5 Patterns with Least
Support 162
14/04/29 20:15:15 INFO fpgrowth.FPGrowth: Mining FTree Tree for all
patterns with 0
```

```
14/04/29 20:15:15 INFO fpgrowth.FPGrowth: Found 4 Patterns with Least
Support 167
14/04/29 20:15:15 INFO fpgrowth.FPGrowth: Tree Cache: First Level: Cache
hits=1180 Cache Misses=21972
```

When Mahout has finished, it dumps the results into a file called `patterns` with no file extension. If you attempt to read the file as-is, then it won't make much sense. You need to use Mahout again to read the sequence file. You use the sequence dumper function that's supplied with Mahout:

```
/your/path/to/mahout/bin/mahout seqdumper -i patterns
```

Then you see the actual mined output:

```
Key class: class org.apache.hadoop.io.Text Value Class: class org.
apache.mahout.fpm.pfpgrowth.convertors.string.TopKStringPatterns
Key: 91: Value: ([91],7), ([133, 233, 286, 91],5), ([111, 270, 91],5),
([177, 91],5), ([179, 91],5), ([133, 142, 91],4), ([133, 17, 91],4),
([125, 91],4), ([133, 142, 17, 91],3), ([125, 133, 142, 91],3), ([142,
5, 91],3)
Key: 136: Value: ([136],7), ([136, 142, 176, 253, 286],5), ([125,
136],5), ([136, 231],5), ([125, 133, 136, 142],4), ([136, 142, 17],4),
([125, 133, 136, 142, 17],3), ([136, 142, 17, 5],3), ([136, 301, 5],3),
([125, 133, 136, 142, 17, 5],2)
Key: 57: Value: ([57],8), ([125, 57],6), ([125, 301, 57],4), ([125, 142,
301, 57],3), ([125, 133, 301, 57],3), ([125, 17, 5, 57],3), ([239, 301,
57],3), ([111, 57],3)
Key: 30: Value: ([30],8), ([125, 30],6), ([133, 142, 30],5), ([17,
30],5), ([133, 142, 17, 30],4), ([125, 133, 142, 30],4), ([125, 17,
30],4), ([125, 133, 142, 17, 30],3), ([125, 17, 30, 5],3), ([111, 125,
30, 5],3)
Key: 275: Value: ([275],8), ([133, 275],5), ([125, 275],5), ([125, 142,
17, 275],4), ([125, 133, 142, 17, 275],3), ([125, 17, 275, 5],3), ([125,
142, 275, 5],3), ([125, 133, 17, 275],3), ([133, 275, 5],3)
```

The "Inspecting the Results" section makes more sense of the results. This method does not give us the whole story, as the frequent list file isn't produced, only the patterns. To get the proper results you need Hadoop to do the work for you. If you want to use Hadoop, then read the "Mahout Using Hadoop" section.

Mahout Using Hadoop

Assuming that the NameNode is formatted and ready to use (Chapter 10, "Machine Learning as a Batch Process," has a very quick run-through), then it's just a case of copying the `output.dat` file to the Hadoop Distributed File System (HDFS):

```
hadoop fs -put output.dat output.dat
```

Then you run Mahout in the same way described in the preceding section, but this time you're adding the –method flag in the command:

```
mahout fpg –i output.dat –o patterns –k 10 –method mapreduce –s 2
```

Mahout will use its MapReduce methods in Hadoop using the data stored in HDFS, which is great when there's a lot of data to process.

When the task is complete, you see a directory called patterns and within that directory there will be four files:

```
jason@myserver:~/mahoutdemo/patterns$ ls -l
total 20
-rwxrwxrwx 1 jason jason 6098 May  1 00:06 fList
drwxrwxr-x 2 jason jason 4096 May  1 00:06 fpgrowth
drwxrwxr-x 2 jason jason 4096 May  1 00:06 frequentpatterns
drwxrwxr-x 2 jason jason 4096 May  1 00:06 parallelcounting
jason@myserver:~/mahoutdemo/patterns$
```

The output files are specific to Mahout. You need to write some code to interpret the results.

Inspecting the Results

The results are in a raw format that Mahout can understand. There is a sequence dumper that's available, though, so you can have a look.

```
jason@bigdatagames:~/mahoutdemo$ /usr/local/mahout/bin/mahout seqdumper
-i patterns/frequentpatterns/part-r-00000
MAHOUT_LOCAL is set, so we don't add HADOOP_CONF_DIR to classpath.
MAHOUT_LOCAL is set, running locally
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/local/mahout-distribution-0.8/
mahout-examples-0.8-job.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/local/mahout-distribution-0.8/
lib/slf4j-jcl-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an
explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.JCLLoggerFactory]
May 01, 2014 1:50:45 PM org.slf4j.impl.JCLLoggerAdapter info
INFO: Command line arguments: {--endPhase=[2147483647],
--input=[patterns/frequentpatterns/part-r-00000], --startPhase=[0],
--tempDir=[temp]}
Input Path: patterns/frequentpatterns/part-r-00000
Key class: class org.apache.hadoop.io.Text Value Class: class org.
apache.mahout.fpm.pfpgrowth.convertors.string.TopKStringPatterns
Key: 1: Value: ([1],80), ([142, 1],42), ([133, 1],42), ([125, 1],39),
([133, 142, 1],30), ([142, 125, 1],29), ([133, 125, 1],29), ([133, 142,
```

```
125, 1],22), ([133, 142, 154, 1],20), ([133, 142, 125, 154, 1],15)
Key: 10: Value: ([10],39), ([133, 10],22), ([142, 10],21), ([125,
10],21), ([142, 125, 10],15), ([133, 142, 10],15), ([133, 125, 10],14),
([133, 142, 176, 10],12), ([133, 142, 125, 10],11), ([133, 142, 125,
176, 10],10)
Key: 100: Value: ([100],35), ([17, 100],22), ([142, 100],22), ([133,
100],20), ([142, 17, 100],18), ([133, 17, 100],17), ([133, 142,
100],15), ([133, 142, 17, 100],14), ([133, 125, 100],14), ([133, 142,
125, 17, 100],12)
```

The product item number (`Key`) is listed along with the top 10 associations. So, for example, product 1 appears in 80 transactions, and it also appears 42 times along with items 142 and 133, and so on. Code-wise it's quite long, so I've provided the edited highlights. You can see the full code listing in the code examples for this book, and you can download them from `http://www.wiley` `.com/go/machinelearning`.

There are two files you have to load in: the frequency file and the frequency patterns file. You read more about the latter shortly; here's the method for loading in the frequency file:

```
public static Map<Integer, Long> readFrequencyFile(Configuration
configuration, String fileName) throws Exception {
        FileSystem fs = FileSystem.get(configuration);
        org.apache.hadoop.io.SequenceFile.Reader frequencyReader = new
org.apache.hadoop.io.SequenceFile.Reader(fs,
                new Path(fileName), configuration);
        Map<Integer, Long> frequency = new HashMap<Integer, Long>();
        Text key = new Text();
        LongWritable value = new LongWritable();
        while(frequencyReader.next(key, value)) {
            frequency.put(Integer.parseInt(key.toString()), value.
get());
        }
        return frequency;
    }
```

This part is straightforward, but you're using Hadoop's sequence file reader and not a normal file class to read in the data. After it's opened, you iterate through the keys and the values adding them to the HashMap.

The products are still in your MySQL database, so you need a method to store those in a Map, too.

```
public static Map<Integer, String> loadItems() {
        Map<Integer, String> products = new HashMap<Integer, String>();
        try {
            Connection con = DriverManager.getConnection("jdbc:mysql://
localhost/apriori","root","");
```

```
            PreparedStatement pstmt = con.prepareStatement("SELECT *
FROM products");
            ResultSet rs = pstmt.executeQuery();
            while(rs.next()) {
                products.put(new Integer(rs.getInt("id")),
rs.getString("productname"));
            }
            rs.close();
            pstmt.close();
            con.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
        return products;
    }
```

Next, you execute the methods to calculate the lift and conviction of the results and the main method to start everything off.

```
private static double calcLift(double occurrence, int transcationcount,
long firstfreq, long otheritemoccurences) {
        return ((double)occurrence * transcationcount) / (firstfreq *
otheritemoccurences);
    }

    private static double calcConviction(double confidence, int
transactioncount, double otheroccurrences) {
        return (1.0 - otheroccurrences / transactioncount) / (1.0 -
confidence);
    }

    public static void main(String[] args)  throws Exception{
        Configuration configuration = new Configuration();
        processResults(configuration,loadItems());
    }
```

The final part is the `processResults` method; it's quite a long routine that reads in the frequency patterns file and then processes the data. Finally, it prints out the item and the associated item with the support, confidence, lift, and conviction.

Putting It All Together

You need to extract the results from HDFS before you can process them, so you need to run the following two lines from the command line:

```
hadoop fs –getmerge patterns/frequentpatterns fpatters.seq
hadoop fs –get patterns/fList flist.seq
```

Now, you can run the DataReader program to work through the results. When the program is run, you'll see the following output:

```
White Bread >> 2pct. Milk: support=0.051, conf=0.470, lift=3.947,
conviction=1.662
Potato Chips >> 2pct. Milk: support=0.045, conf=0.409, lift=4.189,
conviction=1.528
White Bread >> Tomatoes: support=0.040, conf=0.611, lift=5.134,
conviction=2.265
White Bread >> Eggs: support=0.055, conf=0.449, lift=3.773,
conviction=1.599
2pct. Milk >> Eggs: support=0.052, conf=0.425, lift=3.883,
conviction=1.549
```

Mahout doesn't handle multiple item sets, which might be a drawback to you. For many of us, though, it's a good starting point.

Further Development

The boilerplate code offers a brief introduction on how to get association rules learning up and running in Java. By extending your use of the MySQL database, you can set up another table of the rules output.

```
mysql> create table associations(
    -> id int(11) not null primary key auto_increment,
    -> productid int(11) not null default -1,
    -> associationproductid int(11) not null default -1,
    -> support_value double,
    -> lift_value double,
    -> confidence_value double,
    -> conviction_value double);
Query OK, 0 rows affected (0.26 sec)

mysql>
```

The `processResults` method can be changed so that instead of printing to the console, it can save data to this table instead. This makes retrieving information, especially in a web context, much easier.

It's also worth varying the minimum support and confidence values, because you might see some small changes in the output that can work to your advantage.

Summary

Association rules learning is very domain specific, so the case for how it will work in your organization will vary from case to case. This chapter offered a brief overview and a working demo with Mahout either in a standalone run or in Hadoop.

Chapter 10 covers the journey with Hadoop a little deeper. Chapter 7 covers support vector machines.

Support Vector Machines

With most machine learning tasks, the aim is usually to classify something into a group that you can then inspect later. When it's a couple of class types that you're trying to classify, then it's a fairly trivial matter to perform the classification. When you are dealing with many types of classes, the process becomes more of a challenge. Support vector machines help you work through the challenging classifications.

This chapter looks at support vector machines: how the basic algorithm works in a binary classification sense, and then an expanded discussion on the tool.

What Is a Support Vector Machine?

A *support vector machine* is essentially a technique for classifying objects. It's a supervised learning method, so the usual route for getting a support vector machine set up would be to have some training data and some data to test the algorithm. With support vector machines, you have the linear classification—it's either that object, or it's that object—or non-linear. This chapter looks at both types.

There is a lot of comparison of using a support vector machine versus the artificial neural network, especially as some methods of finding minimum errors and the Sigmoid function are used in both.

139

It's easy to imagine a support vector machine as either a two- or three-dimensional plot with each object located within. Essentially, every object is a point in that space. If there's sufficient distance in the area, then the process of classifying is easy enough.

Where Are Support Vector Machines Used?

Support vector machines are used in a variety of classification scenarios, such as image recognition and hand-writing pattern recognition.

Image classification can be greatly improved with the use of support vector machines. Being able to classify thousands or millions of images is becoming more and more important with the use of smartphones and applications like Instagram. Support vector machines can also do text classification on normal text or web documents, for instance.

Medical science has long used support vector machines for protein classification. The National Institute of Health has even developed a support vector machine protein software library. It's a web-based tool that classifies a protein into its functional family.

Some people criticize the support vector machine because it can be difficult to understand, unless you are blessed with a very good mathematician who can guide and explain to you what is going on. In some cases you are left with a black box implementation of a support vector machine that is taking in input data and producing output data, but you have little knowledge in between.

Machine learning with support vector machines takes the concept of a perceptron (as explained in Chapter 5) a little bit further to maximize the geometric margin. It's one of the reasons why support vector machines and artificial neural networks are frequently compared in function and performance.

The Basic Classification Principles

For those who've not immersed themselves in the way classification works, this section offers an abridged version. The next section covers how the support vector machine works in terms of the classification. I'm keeping the math as simple as possible.

Binary and Multiclass Classification

Consider a basic classification problem: You want to figure out which objects are squares and which are circles. These squares and circles could represent

anything you want—cats and dogs, humans and aliens, or something else. Figure 7-1 illustrates the two sets of objects.
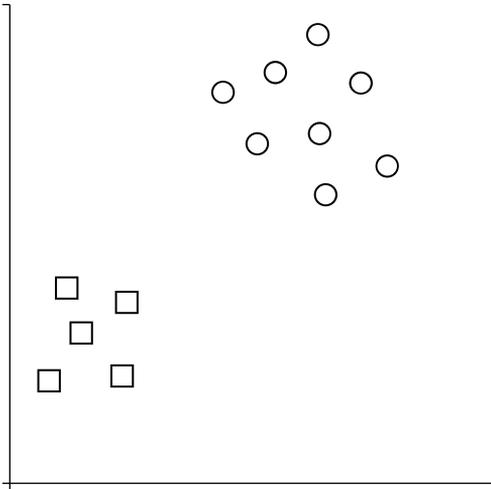
Figure 7-1: Two objects to classify

This task would be considered a binary classification problem, because there are only two outcomes; it's either one object or the other. Think of it as a 0 or a 1. With some supervised learning, you could figure out pretty quickly where those classes would lie with a reasonable amount of confidence.

What about when there are more than two classes? For example, you can add triangles to the mix, as shown in Figure 7-2.
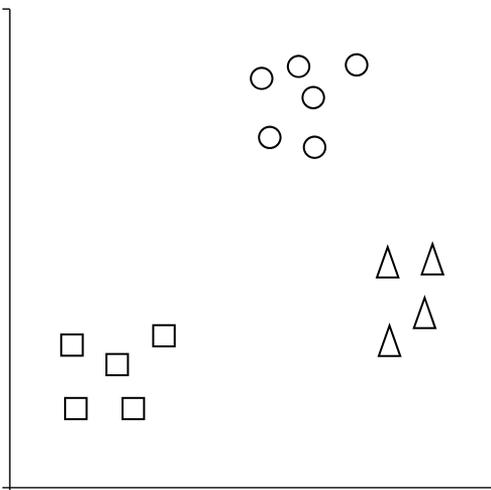
Figure 7-2: Three objects to classify

Binary classification isn't going to work here. You're now presented with a multiclass classification problem. Because there are more than two classes, you have to use an algorithm that can classify these classes accordingly. It's worth noting, though, that some multiclass methods use pair-wise combinations of binary classifiers to get to a prediction.

Linear Classifiers

To determine in which group an object belongs, you use a linear classifier to establish the locations of the objects and see if there's a neat dividing line—called a *hyperplane*—in place; there should be a group of objects clearly on one side of the line and another group of objects just as clearly on the opposite side. (That's the theory, anyway. Life is rarely like that, which is something that's covered more later in the chapter.) Assume that all your ducks are in a row. . .well, two separate groups.

As shown in Figure 7-3, visually it looks straightforward, but you need to compute it mathematically. Every object that you classify is called a point, and every point has a set of features.
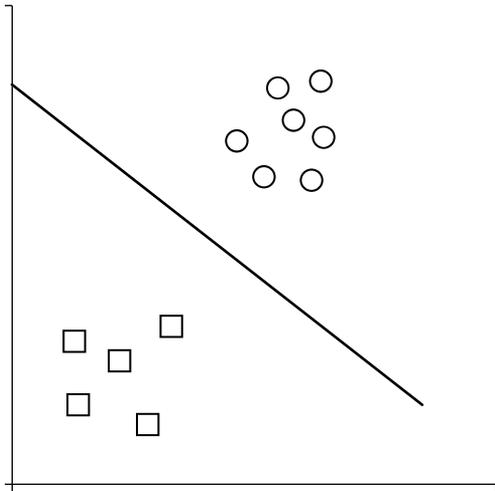
Figure 7-3: Linear classification with a hyperplane

For each point in the graph, you know there is an x-axis value and there is a y-axis value. The classification point is calculated as

$$sign(ax + by + c)$$

The values for a, b, and c are the values that define the line; these values are ones that you choose, and you'll need to tweak them along the way until

you get a good fit (clear separation). What you are interested in, though, is the result; you want a function that returns +1 if the result of the function is positive, signifying the point is in one category, and returns -1 when the point is in the other category. The function's resulting value (+1 or -1) must be correct for every point that you're trying to classify.

Don't forget that you have a training file with the correctly classified data so that you can judge the function's correctness; this approach is a supervised method of learning. This step has to be done to figure out where the line fits. Points that are further away from the line show more confidence that they belong to a specific class.

Confidence

You've just established that each point has a confidence based on its distance from the hyperplane line. The confidence can be translated into a probability. That gives the equation of

$$P[l = +1 \mid x] = \frac{1}{1 + \exp(-(ax + by + c))}$$

This is for one point. What you need is the probability for every set of lines; these are then assigned to each of the objects in the training data.

$$\prod_{i=1}^{N} P[l_i \mid x_i] = \prod_{i=1}^{N} \frac{1}{1 + \exp(-l_i(ax_i + by_i + c))}$$

Probabilities are multiplied because the points have been drawn independently. You have an equation for each point that indicates how probable it is that a hyperplane is producing the correct categorization. Combining the probabilities for each point produces what is commonly defined as the "likelihood of the data"; you are looking for a number as close to 1 as possible.

Remember that probability is based on a value between 0 and 1 (for a recap, check out Chapter 4). Within a set of objects, you're looking for a set of line parameters with the highest probability that confirms the categorization is correct.

Maximizing and Minimizing to Find the Line

Using a log function that is always increasing maximizes values that are above the equation. So, you end up with a function written as

$$\sum_{i=1}^{N} -\log(1 + \exp(-l_i(ax_i + by_i + c)))$$

To achieve minimization, you just multiply the equation by -1. It then becomes a "cost" or "loss" function. The goal is to find line parameters that minimize this function.

Linear classifiers are usually fast; they will process even large sets of objects with ease. This is a good thing when using them for document classification where the word frequencies might require measuring.

How Support Vector Machines Approach Classification

The basic explanation of linear classification is that the hyperplane creates the line that classifies one object and another. Support vector machines take that a step further.

Within the short space available, I outline how support vector machines work in both linear and non-linear form. I also show you how to use Weka to do some practical work for you.

Using Linear Classification

Look at the set of circle and square objects again. You know how a hyperplane divides the objects into either 1 or -1 on the plane.

Extending that notion further, support vector machines define the maximum margin, assuming that the hyperplane is separated in a linear fashion. You can see this in Figure 7-4 with the main hyperplane line giving the written notation of

$$w \bullet x - b = 0$$

This dot product shows the normal vector, and x is the point of the object. There is an offset of the hyperplane that goes from the origin to the normal vector.

As the objects are linearly separable, you can create another two hyperplanes—edge hyperplanes—that define the offset on either side of the main hyperplane. There are no objects within the region that spans between the main hyperplane and the edge hyperplanes.

On one side, there's the equation

$$w \bullet x - b = 1$$

and on the other side there's

$$w \bullet x - b = -1v$$

The objects that lie on the edge hyperplanes are the support vectors. (See Figure 7-5.)
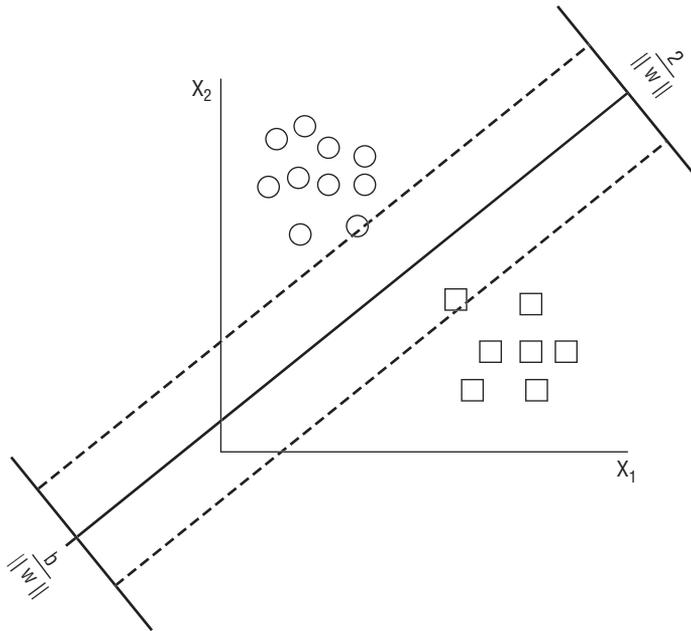
Figure 7-4: Support vector machines max margin hyperplane
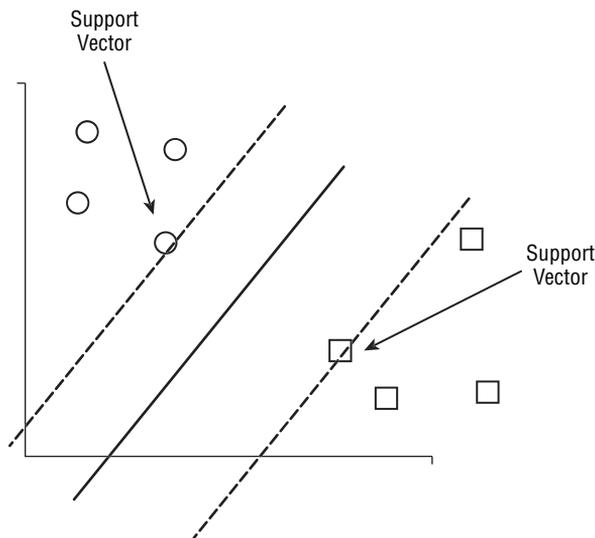
Figure 7-5: The support vectors on the hyperplane edges

When new objects are added to the classification, then the hyperplane and its edges might move. The key objective is to ensure a maximum margin between the +1 edge hyperplane and the -1 edge hyperplane.

If you can manage to keep a big gap between the categories, then there's an increase in confidence in your predictions. Knowing the values of the hyperplane edges gives you a feel for how well your categories are separated.

After minimizing the value w (called $||w||$ in mathematical notation), you can look at optimizing w by applying the following equation:

$$\frac{1}{2}||w||_2$$

Basically, you're taking half of $||w||$ squared instead of using the square root of $||w||$. Based on Lagrange multipliers, to find the maxima and minima in the function, you can now look for a saddle point and discount other points that don't match zero (fit inside the saddle).

> **NOTE** For those that don't know, a saddle point is a mathematical function where you have two variables that meet at a critical point when both function values are zero. It's called a saddle point as that's the shape it produces in graphic form. You can read more about it at this URL `http://wikipedia.org/wiki/Saddle_point`.

You're shaping the graph into a multidimensional space and seeing where the vectors lie in order to make the category distinctions as big as possible. With standard quadratic programming, you then apply the function expressing the training vectors as a linear combination

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

Where α_i is greater than zero, the x_i value is a support vector.

Using Non-Linear Classification

In an ideal world, the objects would lie on one side of the hyperplane or the other. Life, unfortunately, is rarely like that. Instead, you see objects straying from the hyperplane, as shown in Figure 7-6.

By applying the kernel function (sometimes referred to as "the kernel trick"), you can apply an algorithm to fit the hyperplane's maximum margin in a feature space. The method is very similar to the dot products discussed in the linear methods, but this replaces the dot product with a kernel function.

With a radial basis function, you have a few kernel types to choose from: the hyperbolic tangent, Gaussian radial basis function (or RBF, which is supported in Weka), and two polynomial functions—one homogenous and the other inhomogeneous.

The full scope of non-linear classification is beyond the means of the introductory nature of this book. If you want to try implementing them, then look at the radial basis functions in the LibSVM classes when you use Weka.
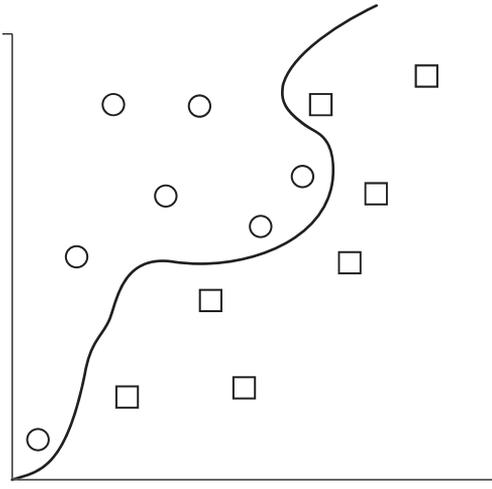
Figure 7-6: Objects rarely go where you want them to.

Now take a look at what Weka can do for you to perform support vector machine classification.

Using Support Vector Machines in Weka

Weka can classify objects using the support vector machines algorithm, but the implementation isn't complete and requires a download before you can use it. This section shows you how to set it up and run the support vector machines algorithm on some test data.

Installing LibSVM

The LibSVM library is an implementation of the support vector machines algorithm. It was written by Chih-Chung Chang and Chih-Jen Lin from the National Taiwan University. The library supports a variety of languages as well as Java including C, Python, .NET, MatLab, and R.

Weka LibSVM Installation

You can install LibSVM from Github. You can clone the binary distribution by running the following command (assuming you have git installed):

```
git clone https://github.com/cjlin1/libsvm.git
```

The required files download into a clean directory.

You need to copy the `libsvm.jar` file to the same directory as your Weka installation directory (usually in the `/Applications` directory). You can easily drag and drop the file if desired; I work from the command line most of the time:

```
cp ./libsvm-3.18/java/libsvm.jar /Applications/weka-3-6-10
```

With the library in place, you can start Weka. If you are a Windows user just start Weka as normal, but if you run Mac OS X or Linux then you have to do it from the command line:

```
java -cp weka.jar:libsvm.jar weka.gui.GUIChooser
```

If you do not start Weka from the command line, then the classifier gives you an error to let you know that the SVM libraries were not found in the classpath.

A Classification Walkthrough

You will see the GUI Chooser application open as you would when you open Weka by starting the GUI instead of using the command line (see Figure 7-7). Choose the Explorer option.
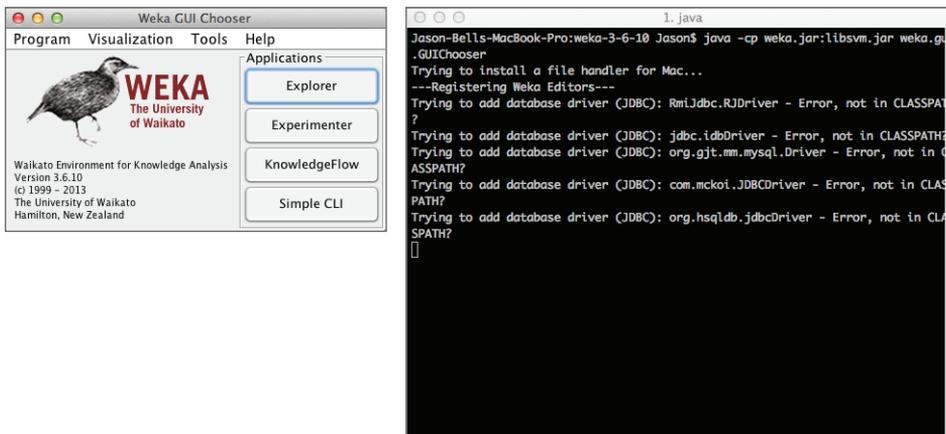
Figure 7-7: GUI Chooser

I'm going to use the 100,000 rows of vehicle data that I created in Chapter 5 for the artificial neural networks; you can do the same. Find the `.csv` file and open it in Weka, as shown in Figure 7-8. Don't forget to change the file type from `.arff` to `.csv`.

Setting the Options

Click the Classify tab and then click the Choose button to select a different classification algorithm. Within the tree of algorithms, click Functions and then select LibSVM, as shown in Figure 7-9.
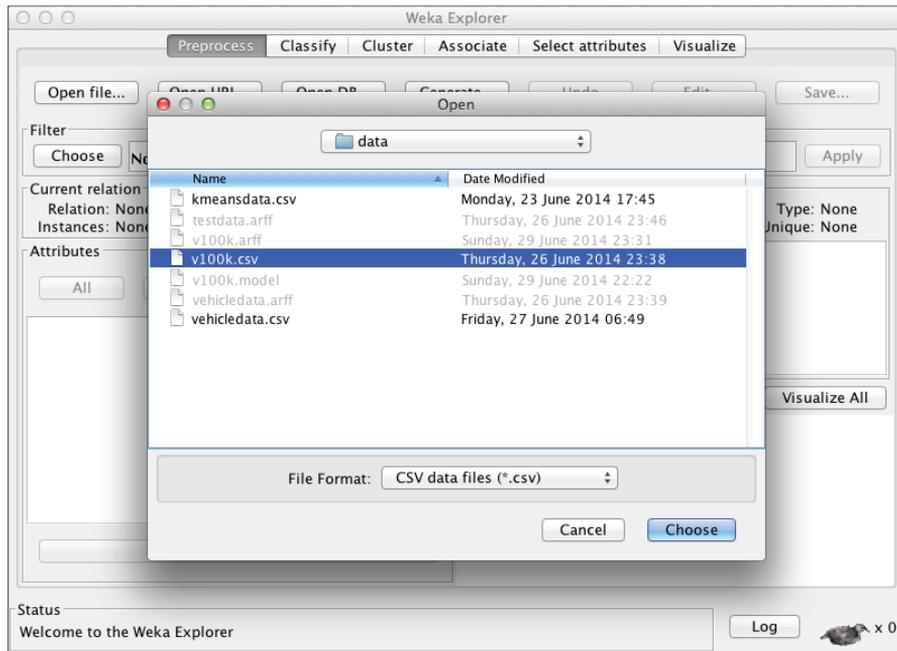
Figure 7-8: Loading the .csv file
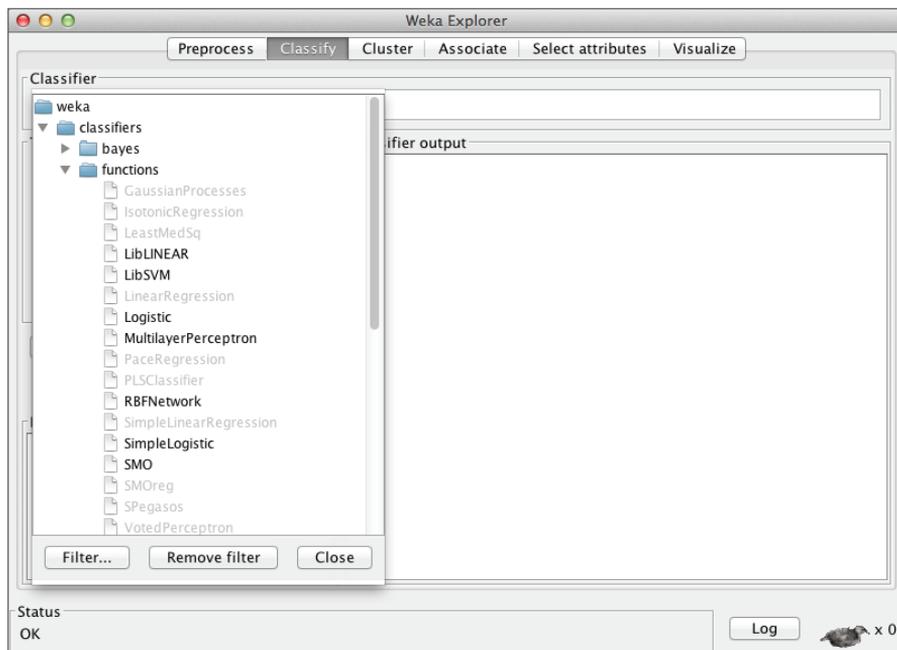
Figure 7-9: Choosing the LibSVM classifier

There are a couple of changes to make before you set the classifier off to work. First, you want a percentage split of training data against test data. In this case, you can be fairly confident that the data is not going to be difficult to classify and it's not going to be a non-linear classification problem; you can train with 10 percent of the data (10,000 rows) and test with the 90 percent to see how it performs.

Click the Percentage Split option and change the default value of 66 percent to 10 percent, as shown in Figure 7-10.
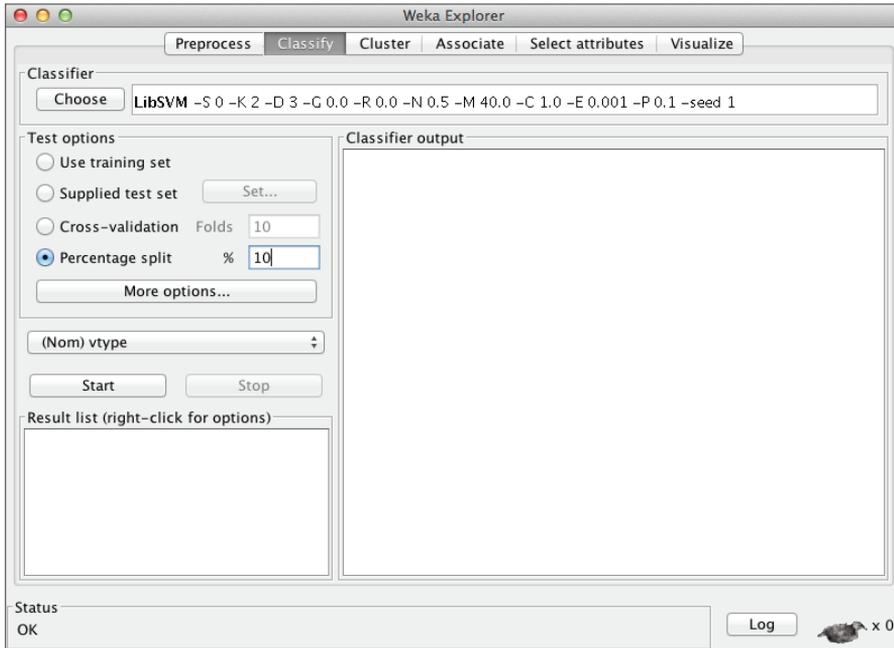
Figure 7-10: Changing the percentage split

You want the results of the test data, the 90 percent to be output to the Weka console so you can see how it's performing. Click the Options button and ensure that the Output Predictions checkbox is ticked, as shown in Figure 7-11.

The LibSVM wrapper defaults to a radial basis function for its kernel type. Change that to the linear version you've been concentrating on by clicking on the line with all the LibSVM options. This is located next to the Choose button within the Classifer pane.

Change the kernelType drop-down menu from Radial Basis Function to Linear. Leave the other options as they are. (See Figure 7-12.)
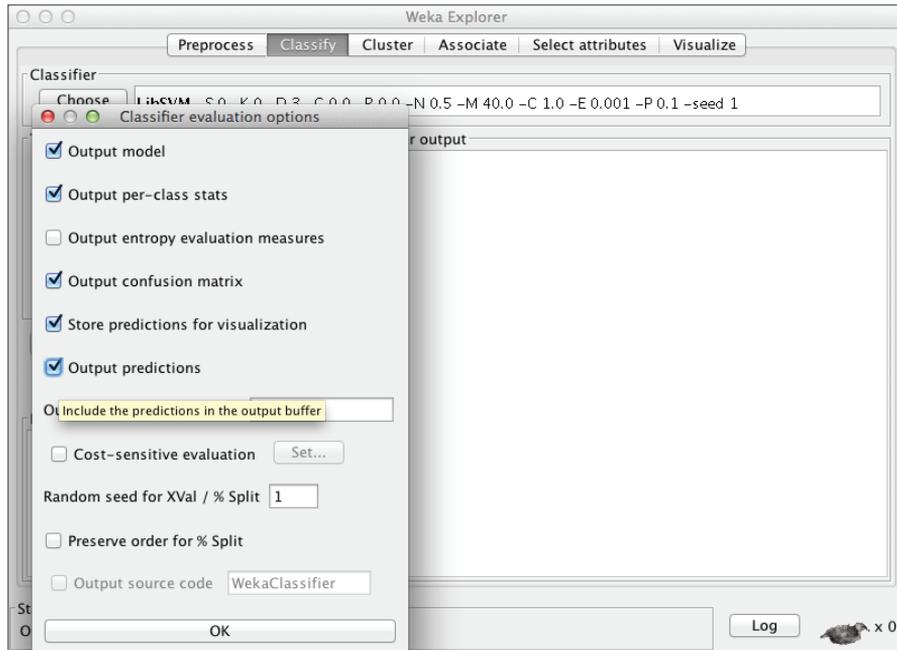
Figure 7-11: Classifer Evaluation Options dialog box
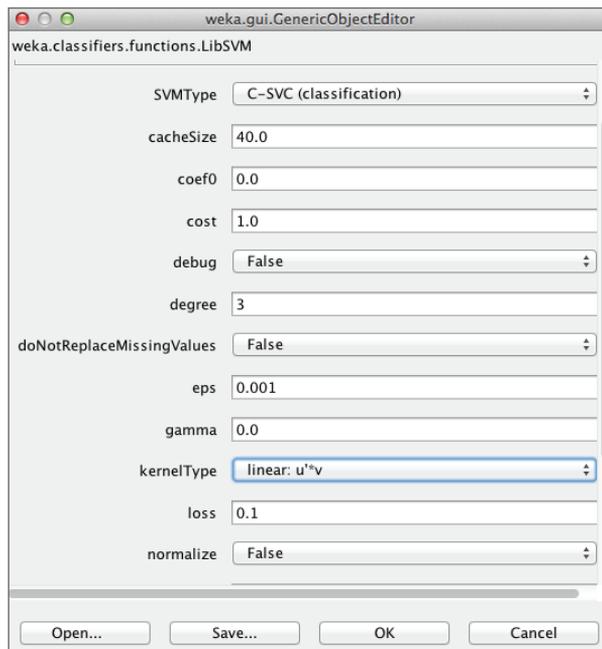
Figure 7-12: Changing the kernel type

Running the Classifier

With everything set, you can run the classifier. Click the Start button, and you see the output window start to output information on the classification.

First, you have the run information, or all the options that you just set:

```
=== Run information ===

Scheme:weka.classifiers.functions.LibSVM -S 0 -K 0 -D 3 -G 0.0 -R 0.0 -N
0.5 -M 40.0 -C 1.0 -E 0.001 -P 0.1 -seed 1
Relation:     v100k
Instances:    100000
Attributes:   4
              wheels
              chassis
              pax
              vtype
Test mode:split 10.0% train, remainder test
```

Next, you get some general information on the classifier model:

```
=== Classifier model (full training set) ===

LibSVM wrapper, original code by Yasser EL-Manzalawy (= WLSVM)

Time taken to build model: 3.08 seconds
```

On my machine, the classifier trained on 10,000 instances in just over three seconds, which is 3,246 rows per second.

As you've set the output for the predictions to be shown, you get that next:

```
=== Predictions on test split ===

inst#,    actual, predicted, error, probability distribution
     1    4:Bike    4:Bike        0      0      0     *1
     2    4:Bike    4:Bike        0      0      0     *1
     3   3:Truck   3:Truck        0      0     *1      0
     4     1:Bus     1:Bus       *1      0      0      0
     5     1:Bus     1:Bus       *1      0      0      0
     6     2:Car     2:Car        0     *1      0      0
     7    4:Bike    4:Bike        0      0      0     *1
     8   3:Truck   3:Truck        0      0     *1      0
     9   3:Truck   3:Truck        0      0     *1      0
    10     2:Car     2:Car        0     *1      0      0
    11    4:Bike    4:Bike        0      0      0     *1
    12     2:Car     2:Car        0     *1      0      0
    13   3:Truck   3:Truck        0      0     *1      0
    14   3:Truck   3:Truck        0      0     *1      0
    15    4:Bike    4:Bike        0      0      0     *1
```

```
16      1:Bus       1:Bus       *1      0       0       0
17      1:Bus       1:Bus       *1      0       0       0
18      2:Car       2:Car        0     *1       0       0
19      1:Bus       1:Bus       *1      0       0       0
```

Based on the training data of 10,000, you've instructed Weka to try and predict the remaining 90,000 rows of data. The output window will have all 90,000 rows there, but the main things to watch out for are the actual and predicted results.

You get the evaluation on the test data showing the correct and incorrect assignments:

```
=== Evaluation on test split ===
=== Summary ===

Correctly Classified Instances      90000                100      %
Incorrectly Classified Instances        0                  0      %
Kappa statistic                         1
Mean absolute error                     0
Root mean squared error                 0
Relative absolute error                 0      %
Root relative squared error             0      %
Total Number of Instances           90000
```

The confusion matrix shows the breakdown of the test data and how it was classified:

```
=== Confusion Matrix ===

    a     b     c     d   <-- classified as
22486     0     0     0 |    a = Bus
    0 22502     0     0 |    b = Car
    0     0 22604     0 |    c = Truck
    0     0     0 22408 |    d = Bike
```

Dealing with Errors from LibSVM

There are variations of the LibSVM library around the Internet and also different ways the random number generator handles numbers on differing operating systems. If you come across an error like the following:

```
java.lang.NoSuchFieldException: rand
java.lang.Class.getField(Unknown Source)
weka.classifiers.functions.LibSVM.buildClassifier(LibSVM.java:1618)
weka.gui.explorer.ClassifierPanel$16.run(ClassifierPanel.java:1432)
at java.lang.Class.getField(Unknown Source)
at weka.classifiers.functions.LibSVM.buildClassifier(LibSVM.java:1618)
at weka.gui.explorer.ClassifierPanel$16.run(ClassifierPanel.java:1432)
```

then it's worth looking at later versions of Weka with the new package manager (version 3.7 and later).

Saving the Model

You can save the model for this classification. On the result list, you see the date and time that the LibSVM classification was run. Right-click (Alt-click if you are a Mac user) on functions.LibSVM and select Save Model. Find a safe place to save the model for future use.

Implementing LibSVM with Java

Using LibSVM within the Weka toolkit is easy to implement, but there comes a time when you'll want to use it within your own code, so you can integrate it within your own systems.

Converting .csv Data to .arff Format

`.csv` files don't contain the data that Weka will need. You could implement the `CSVLoader` class, but I prefer to know that the `.arff` data is ready for use. It also makes it easier for others to decode the data model if they need to.

From the command line, you can convert the data from a `.csv` file to `.arff` in one command:

```
java -cp /Applications/weka-3-6-10/weka.jar weka.core.converters.
CSVLoader v100k.csv > v100k.arff
```

To ensure that the conversion has worked, you can output the first 20 lines with the `head` command (your output should look like the following sample):

```
$ head -n 20 v100k.arff
@relation v100k

@attribute wheels numeric
@attribute chassis numeric
@attribute pax numeric
@attribute vtype {Bus,Car,Truck,Bike}

@data
6,20,39,Bus
8,23,11,Bus
5,3,1,Car
4,3,4,Car
5,3,1,Car
4,18,37,Bus
```

With everything looking fine, you can now set your attention on the Eclipse side of the project.

Setting Up the Project and Libraries

Using the same data, create a coded example with Java using Eclipse to create the project. Create a new Java Project (select File ➭ New ➭ Java Project) and call it `MLLibSVM`, as shown in Figure 7-13.
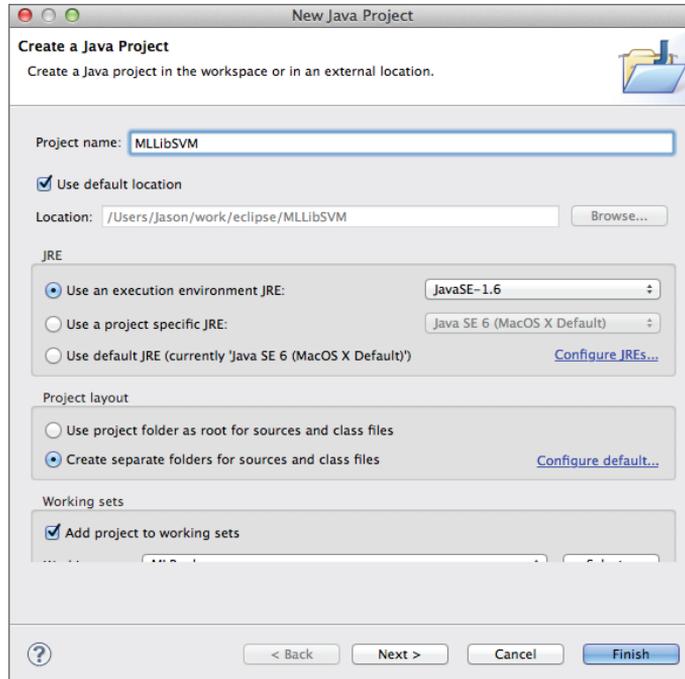
Figure 7-13: Creating the new Java project

The Weka API and the LibSVM API need to be added to the project. Select File ➭ Properties and then select Java Build Path. Click the Add External JARs button. When the File dialog box displays, locate the `weka.jar` and `libsvm.jar` files and click Open. (See Figure 7-14.)

You have everything in place, so you can create a new Java class (File ➭ New ➭ Class) called `MLLibSVMTest.java` (see Figure 7-15) and put some code in place.
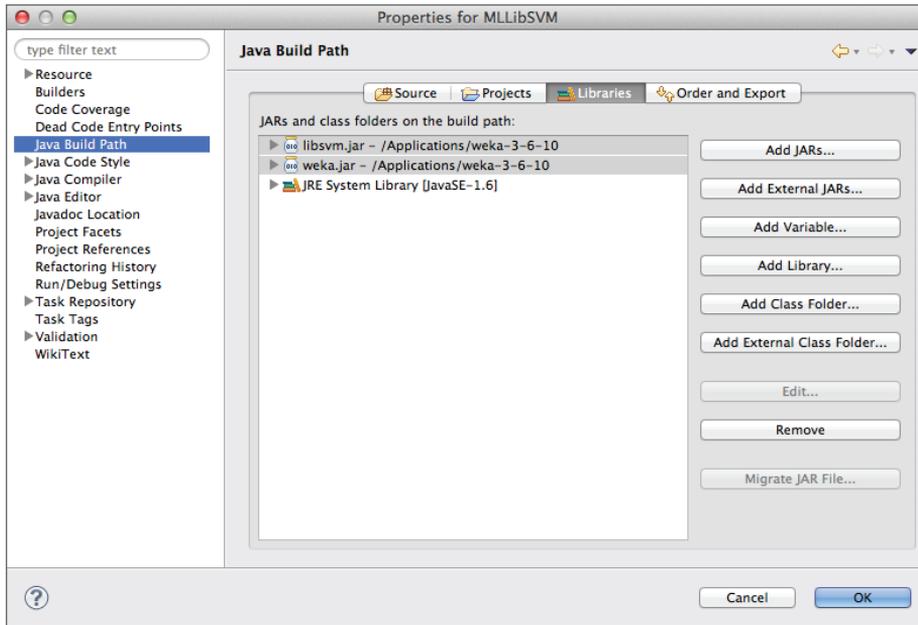
Figure 7-14: Adding the required jar files
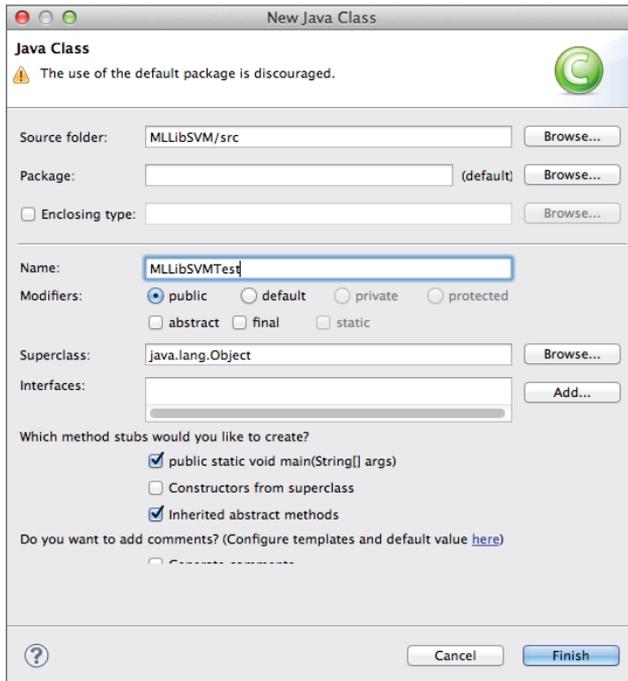
Figure 7-15: Creating a new Java class

The basic code to get a support vector machine working in Weka is a fairly easy task.

```java
public class MLLibSVMTest {
    public MLLibSVMTest(String filepath){
        Instances data;
        try {
            data = DataSource.read(filepath);


            if (data.classIndex() == -1)
                data.setClassIndex(data.numAttributes() - 1);
            LibSVM svm = new LibSVM();
            String[] options = weka.core.Utils.splitOptions("-K 0 -D 3");
            svm.setOptions(options);
                svm.buildClassifier(data);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        MLLibSVMTest mllsvm = new MLLibSVMTest("v100k.arff");
    }
}
```

There are a lot of option settings for the LibSVM library, but the main one I want to focus on is the kernel type. As in the Weka workbench, the default is the radial basis function: In the options, the number 2 designates this. For the linear kernel function, you change that to zero.

To run the code from Eclipse, select Run ⇨ Run. This takes the training data and makes the model. It won't do anything else just yet.

```
Zero Weights processed. Default weights will be used
*
optimization finished, #iter = 9
nu = 7.999320068325541E-7
obj = -0.019999999949535163, rho = 2.1200468836658968
nSV = 4, nBSV = 0
*
optimization finished, #iter = 9
nu = 5.508757892156424E-7
obj = -0.013793103448275858, rho = -1.013793103448276
nSV = 5, nBSV = 0
*
optimization finished, #iter = 3
nu = 3.801428938130698E-7
obj = -0.009478672985781991, rho = 1.2180094786729856
```

```
nSV = 2, nBSV = 0
*
optimization finished, #iter = 5
nu = 1.8774340639289764E-7
obj = -0.004705882352941176, rho = -1.6070588235294119
nSV = 4, nBSV = 0
*
optimization finished, #iter = 6
nu = 8.90259889118131E-6
obj = -0.2222222222222227, rho = 1.6666666666666679
nSV = 3, nBSV = 0
*
optimization finished, #iter = 3
nu = 1.2308677001852457E-7
obj = -0.003076923076923077, rho = 1.1107692307692307
nSV = 2, nBSV = 0
Total nSV = 14
```

The output looks confusing, but what it is telling you is the number of support vectors (nSV), the number of bound support vectors (nBSV), and obj is the optimum objective value of the dual support vector machine.

Training and Predicting with the Existing Data

So far, you've trained with the full 100,000 lines of data from the .arff file. I want to train with 10 percent and then predict the remaining 90 percent in the same way as the workbench walkthrough.

The Weka API lets you add the options as you would in the workbench, so where you split the data for training, you can do the same within the code.

Amend the options line and add the training split percentage like so

```
String[] options = weka.core.Utils.splitOptions("-K 0 -D 3");
```

and it now becomes

```
String[] options = weka.core.Utils.splitOptions("-K 0 -D 3 -split-
percentage 10");
```

To show the predictions of the data, add a new method that iterates through the instance data:

```
public void showInstanceClassifications(LibSVM svm, Instances data) {
        try {
            for (int i = 0; i < data.numInstances(); i++) {
                System.out.println("Instance " + i + " is classified as
a "
```

```
                        +
          data.classAttribute().value((int)svm.classifyInstance(data.
          instance(i)))));
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

The classifier always returns a numerical value as its result; it's up to you to turn that number into an integer and run it past the class attribute value to find out whether it's a bike, car, bus, or truck.

When you run the code again, you see the classifier generate as before with 10 percent of the training data, and then it classifies the whole data set.

```
Instance 99991 is classified as a Truck
Instance 99992 is classified as a Bus
Instance 99993 is classified as a Car
Instance 99994 is classified as a Truck
Instance 99995 is classified as a Car
Instance 99996 is classified as a Bus
Instance 99997 is classified as a Bike
Instance 99998 is classified as a Truck
Instance 99999 is classified as a Bike
```

Summary

This chapter was a whistle stop tour of support vector machines. Whole books have been written on the subject, going deep into the intricacies of the vector machine and its kernel methods.

From a developer's point of view, treat this chapter as a launch pad for further investigation. In a practical scenario, you might gloss over the heavy theory and make Weka do the heavy lifting on a sample or subset of your data.

Before you continue your journey, I think it's only fair that you reward yourself with your beverage of choice and a short rest before you continue into the next chapter on clustering data.

Clustering

One of the more common machine learning strands you'll come across is clustering, mainly because it's very useful. For example, marketing companies love it because they can group customers into segments. This chapter describes the details of clustering and illustrates how clusters work and where they are used.

> **NOTE** Please don't confuse machine learning clustering with clusters of machines in a networking sense; we're talking about something very different here.

What Is Clustering?

If you boil down all the definitions of clustering out there, you get "organizing a group of objects that share similar characteristics." It's classed as an unsupervised learning method, which means there's no prior training data from which to learn. In Figure 8-1 you see there are three distinct groupings of data; each one of those groups is a cluster.
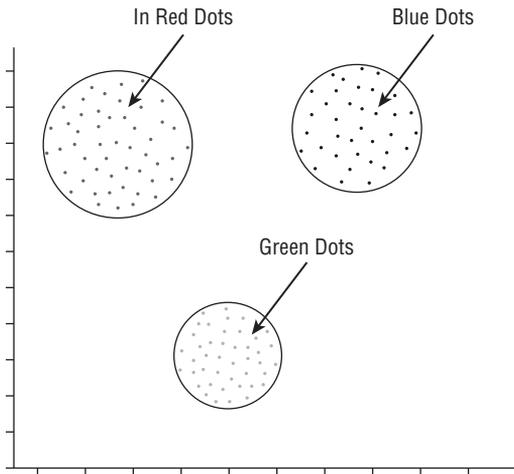
Figure 8-1: A graph representation of a cluster

The main aim is to find structure within a given set of data. Because there are a lot of algorithms to choose from, clustering casts a wide net. This is where experimentation comes in handy; which algorithm is the right choice? Sometimes you just need to put some code together and play with it. You'll do that shortly.

Where Is Clustering Used?

Clustering is a widely-used machine learning approach. Although it might seem simple, do not underestimate the importance of grouping multivariate data into refined groupings.

The Internet

Social media network analysis uses clustering to determine communities of users. With so many users on Facebook, for example, using these sorts of techniques can refine advertising so that certain ads go to specific groups of customers.

If you've ever searched on one of many mapping websites, you might have seen clustering at work when there are a lot of interest pins built up in a given location. Instead of showing all the pins, which would provide a bad user experience, clustering is used to define the group of pins within the given location.

Website logs and search results are often clustered to show more relevant search result groups. A number of companies are using clustering to refine search engine queries.

Business and Retail

Market research companies use clustering a lot. With surveys that contain many variables, a multivariate system like clustering gives marketers better definition of groups of customers in relation to the answers given in the survey. This might be broken down by population, location, and previous buying habits, for example. With the clusters defined, the marketing companies can try to develop new products or think about testing products for certain clusters in the results.

Along with association rules learning (discussed in Chapter 6), clustering is also used for basket analysis. Certain auction sites use clustering because there is no defined stock number in the listings. It's easier to run clustering and group items and preferences than use association rules.

Law Enforcement

Crimes are logged with all the aspects of the felony listed. Police departments are running clustering and other machine learning algorithms to predict when and where future crimes will happen. The result of this might be that patrol cars are deployed to certain problem areas at certain times, or specialist help is sent to areas where certain sorts of crimes show high numbers.

Computing

With the rise of the "Internet of Things," we are now collecting more data from sensors than ever before. Clustering can be used to group the results of the sensors. For example, thinking of a temperature sensor, you might cluster date and time against the temperature. Another example would be motion detection; a number of passive infrared sensors could be generating data on movement within a location. Is a certain location a hotspot at a specific time? This information can be easily clustered and inspected.

Course work in the education sector, especially with the advent of large-scale learning online, can be clustered into student groups and results. For example, do certain clusters of students excel at courses compared to other students?

Clustering is used often in digital imaging. When large groups of images need to be segmented, it's usually a cluster algorithm that works on the set and defines the clusters. Algorithms can be trained to recognize faces, specific objects, or borders, for example.

Clustering Models

As previously mentioned, the goal of clustering is to segment data into specific groups. There are many different clustering algorithms for the simple fact that there is really only one common denominator among all clusters—that you're trying to find groups of objects.

For example, there are distribution models that use multivariate distributions for their modeling. Graph models can show cluster-like properties when the nodes start showing as small subsets connected with one main edge as shown in Figure 8-2.
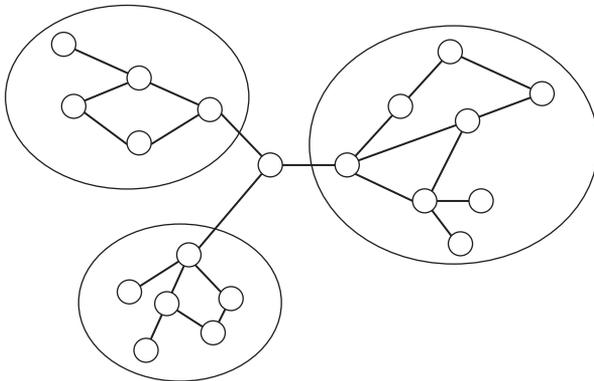
Figure 8-2: Nodes and edges as clusters

You can also approach simple clustering with groups in the same way you group in a structured query language.

One of the more commonly used cluster models is the centroid model, which is where the k-means algorithm comes in. The *k-means algorithm* is basically vector quantization. This chapter concentrates on the k-means algorithm and creates a basis of the walkthrough later in the chapter.

How the K-Means Works

If you have a group of objects, the idea of the k-means algorithm is to define a number of clusters. What's important is that it's up to you to define how many clusters you want. For example, say I have 1000 objects and I want to find 4 clusters:

$$n \text{ (objects)} = 1000$$
$$k \text{ (clusters)} = 4$$

Each one of the clusters has a centroid (sometimes called the mean, hence the name "k-means"), a point where the distance of the objects will be calculated. The clusters are defined by an iterative process on the distances of the objects to

calculate which are nearest to the centroid. This is all done unsupervised; you just have to let the algorithm do its processing and inspect the results. After the iterations have taken place to the point where the objects don't move to different centroids then it's assumed that the k-means clustering is complete.

The following pseudocode describes what's happening:

```
calculate inital values for means m[1],m[2],m[3],m[4]

assign object to nearest center

while(there are changes in the mean position) {
    estimate the means to classify into clusters
    for(i in 1 to k) {
        m[i] = mean of the samples for cluster i
    }
}
```

Initialization

First, the algorithm must initialize by assigning a cluster to every observation made. The *random partition method* places the cluster points toward the center of the dataset. Another initialization method is the *Forgy method*, which spreads out the randomness of the initial location of the cluster.

After the initial cluster observations are assigned, you can look at the assignment and updating of the algorithm.

Assignments

Each observed object is assigned to the clusterl; to find out which cluster centroid it's assigned to, the algorithm uses a Euclidean distance measurement. The sum of squares is then calculated by squaring the Euclidean distances to each cluster centroid, and the one with the smallest value is the cluster which the object is assigned to.

To calculate the Euclidean distance is quite simple and requires only some entry-level math; if you can remember how to do Pythagoras' theorem, then you are already there.

Assume a basic grid of six positions on the X-axis (horizontal) and four positions on the Y-axis (vertical). The center point of my cluster is currently at 1,6 and the object is located at 3,1, as shown in Figure 8-3.

The distance is 3-1 = 2 on the vertical side and 6-1 = 5 on the horizontal axis. Using Pythagoras' theorem, the squared distance is

$$2 \wedge 2 + 5 \wedge 2 = 4 + 25 = 29$$

The square root of 29 is 5.38. This carries on for all the objects in the dataset that are assigned to the clusters.
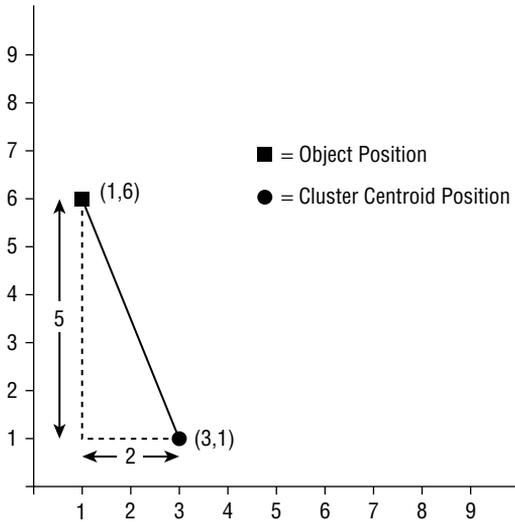
Figure 8-3: Euclidean distances

Update

In the update step, you assign the object values to the cluster. The recalculation of the center points of the cluster (the centroids) is taken as the average of the values of the objects that are part of the cluster. This process carries on as a loop until the entities in each group no longer change.

The k-means algorithm is very effective, but it's not without its problems. It can take a few runs with the data to get a decent fit of clusters. When you choose too few, objects can easily spill into the incorrect cluster over a period of time during the processing.

Calculating the Number of Clusters in a Dataset

When presented with a dataset, it can be hard to define the number of clusters that you want to classify against. Sometimes this number will already be determined by a stakeholder. For example, in a marketing initiative you might have the following:

- Low Frequency, Low Value Customers
- Low Frequency, High Value Customers
- High Frequency, Low Value Customers
- High Frequency, High Value Customers

There are times when this information is not available, and you have to find a balance for making your decisions. There are a number of methods for calculating the optimum.

The Rule of Thumb Method

Nothing beats wetting your finger and sticking it in the air to see which way the wind is blowing. There's a simple calculation that is roughly the equivalent for clusters: The number of clusters (k) is equal to the square root of the number of objects divided by two.

$$k = \sqrt{objects/2}$$

If you have 250 objects, then half of that is 125, and the square root of 125 is 11.18—so, there are 11 clusters. This can obviously be tested and reapplied depending on how the trial runs go.

The Elbow Method

You can calculate the variance of the dataset as a percentage and plot against the number of clusters. There's a point at which the clusters are at an optimum—that point after which adding more clusters will not make a huge difference to the final classifications. You can see in Figure 8-4 how the elbow method shows the optimum number of clusters is four, which has classified 80 percent of the data.
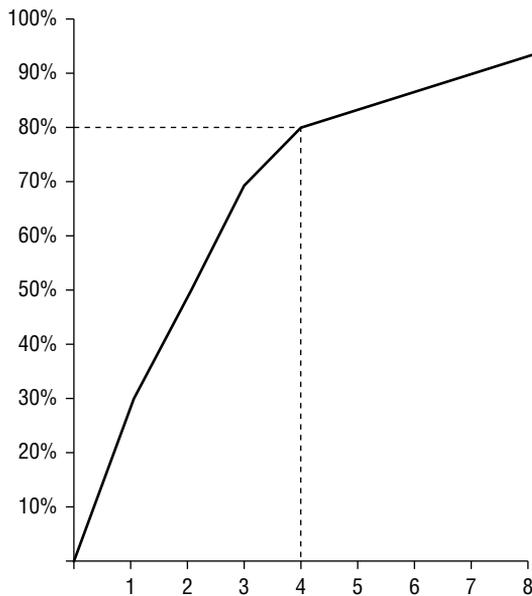
Figure 8-4: The elbow method graph

The Cross-Validation Method

By splitting the dataset into separate partitions, you can apply the analysis on the dataset and then on the remaining partitions. By averaging the results of the sum of squares, you can determine the number of clusters to use.

Weka supports cross-validation with the `weka.clusterers.MakeDensity BasedClusterer` class. This class is covered in more detail in the command-line-based walkthrough later in the chapter.

The Silhouette Method

Peter J. Rousseeuw first described the *silhouette method* in 1986. It is a method for suggesting a way of validating where the objects lay within a cluster.

For any object, you can calculate how similar an object is with another object within the same cluster. By calculating the averages of objects that connect to a cluster and then evaluating how dissimilar they are in relation to the other clusters, you can determine an average score. The main aim is to measure the grouping of the objects in the cluster—the lower the number the better. When comparing the averages for each cluster, they are expected to be similar. When silhouettes are very narrow and others are large, it might point to the fact that not enough clusters have been defined when the computation process began.

K-Means Clustering with Weka

The Weka machine learning application comes with an algorithm for processing k-means clusters, a class called `SimpleKMeans`. In this walkthrough, you'll work with three approaches: one from the workbench application, one that works directly from the command line, and finally one that's a Java coded example.

The aim is to take some marketing data and use the k-means to generate some segmentation of the customers; this will potentially give the marketing department an increase in successful transactions in the long run.

Before you get into the three walkthroughs, you need to prepare some data with which to work.

Preparing the Data

The following Java code generates 75 instances of random numbers for two integer variables, x and y. You import the saved file (`kmeansdata.csv`) into Weka and also load it in programmatically.

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
```

```java
import java.util.Random;

public class DataSet3D {
    public static void main(String[] args) {
        Random r = new Random(System.nanoTime());
        try {
            BufferedWriter out = new BufferedWriter(new
FileWriter("kmeansdata.csv"));
            out.write("x,y\n");
            for(int count = 0; count < 75; count++) {
                out.write(r.nextInt(125) + "," + r.nextInt(150) + "\n");
            }
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
}
```

The output looks something like this:

```
x,y
78,29
0,55
101,19
52,146
49,140
44,97
65,45
41,49
66,141
111,100
23,128
101,1
1,113
88,100
```

If you want to create more instances you can do so by adjusting the number of iterations in the `for` loop of the code. I've set it to 75 as a starting point. The upper limit is really based on the amount of memory your computer has.

Have a look at the Weka workbench method first.

The Workbench Method

The *workbench method* uses the Weka user interface to load, cluster, and then visualize the data. There's no actual programming involved, but it's useful to see what Weka is doing before you progress on to the command-line and coded samples.

Loading Data

The process for the workbench is very similar to other examples you've run though. The first thing you need to do is load in the CSV data.

Click the Open File button and select the `kmeansdata.csv` file that you created earlier. Ensure that the file format drop-down menu shows CSV and not ARFF (see Figure 8-5); otherwise you won't be able to open the file.
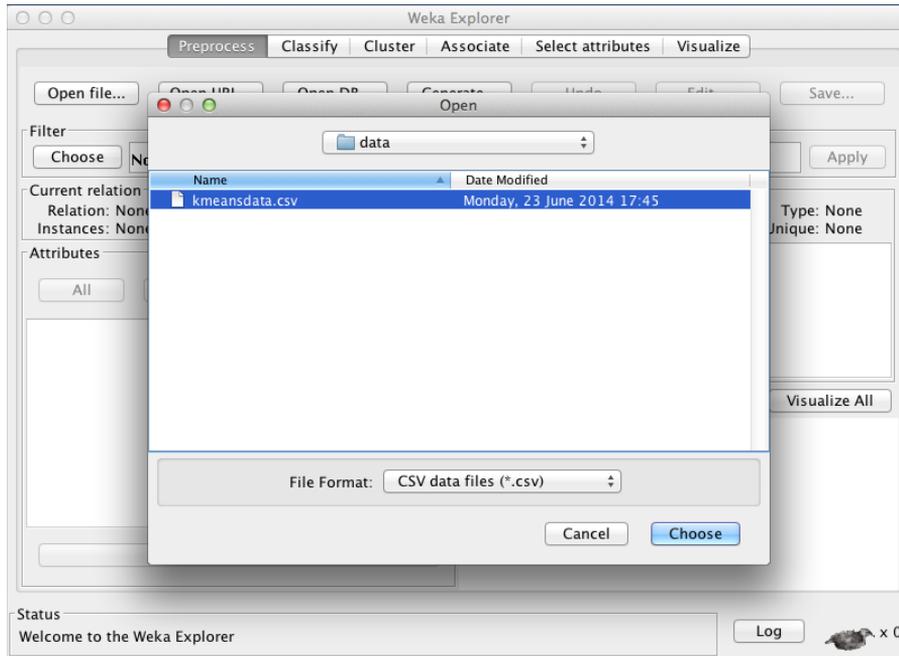
Figure 8-5: Loading CSV data into Weka

When the data has loaded, the explorer shows various pieces of information. The Current Relation pane shows that there are two attributes and 75 instances. (See Figure 8-6.) The attribute information shows the two attributes: x and y.

On the right-hand panel, the Selected Attribute pane shows some statistics of the data that's been loaded, which includes the minimum and maximum values along with the mean and the standard deviation. Finally, there's a graph of the distribution of the values and the frequency of them.

Clustering the Data

Click the Cluster tab at the top to select the clustering method. By default, the Weka clusterer uses the Simple EM method (expectation maximization).

Clicking the Choose button displays a tree of other cluster algorithms that you can use. For this example, select SimpleKMeans and then click Close; finally click Start to run the algorithm, as shown in Figure 8-7.
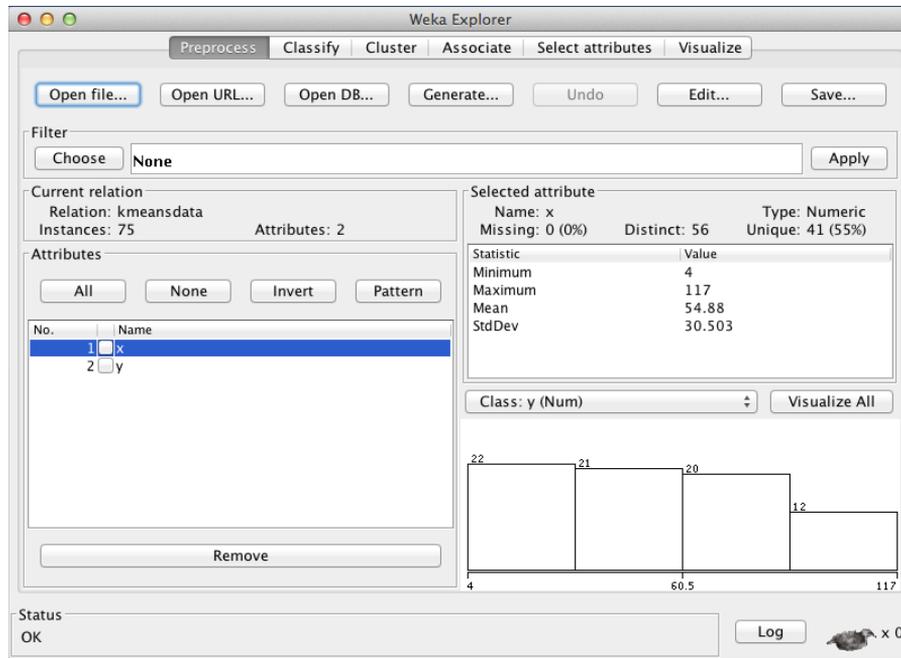
Figure 8-6: The Preprocess window

The clusterer line in the Clusterer output pane has updated and shows the new cluster algorithm you'll be using:

```
SimpleKMeans –N 2 –A "weka.core.EuclideanDistance –R first –last" –l 500
–S 10
```

There are a few flags that need a little explanation.

- ■ -N determines the number of clusters that the SimpleKMeans is going to create.
- ■ -A is the distance function used. It defaults to Euclidean distance and uses the entire range of values as its range to act on (-R first –last).
- ■ The –l flag defines the number of iterations the k-means does to define the cluster.
- ■ -S is a random number seed. It can be any value you want.
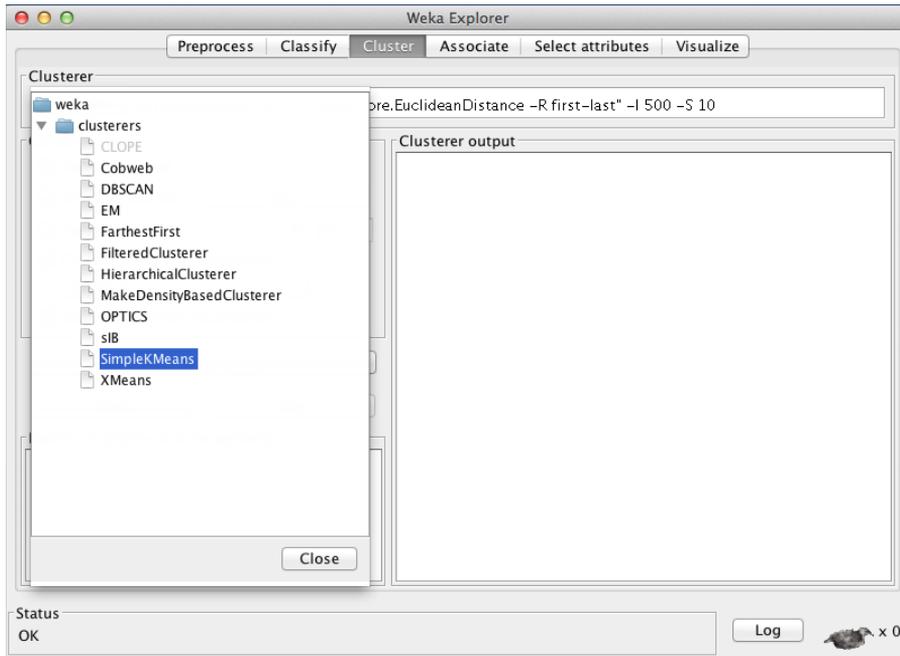
Figure 8-7: Selecting SimpleKMeans

Clicking the line with all the command options shown next to the Choose button displays a pop-up window where you can alter the values. In the numClusters field, change the number from 2 to 4. (See Figure 8-8.) You're going to create four clusters with this data. Click OK to close the window.
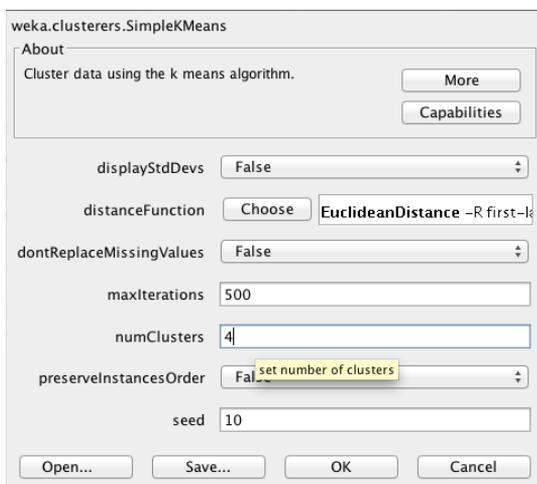
Figure 8-8: Changing the SimpleKMeans options

Start the clustering process by clicking the Start button. For long periods of processing, it's worth keeping an eye on the status in the bottom-left corner of the explorer.

When the process is complete, you see the output appear in the Clusterer Output window. The following sections describe the output.

Run Information

The first block of information gives the settings of the data and the algorithm selected.

```
Scheme:weka.clusterers.SimpleKMeans -N 4 -A "weka.core.EuclideanDistance
-R first-last" -I 500 -S 10
Relation:     td
Instances:    75
Attributes:   2
              x
              y
Test mode:evaluate on training data
```

K-Means

The k-means output shows the actual work that Weka did to reach the results. This includes the number of iterations that were performed on the data and the sum of squared errors.

```
Number of iterations: 3
Within cluster sum of squared errors: 0.8072960323968902
Missing values globally replaced with mean/mode

Cluster centroids:
                        Cluster#
Attribute    Full Data        0         1         2         3
                (75)       (15)      (23)      (17)      (20)
==================================================================
x               54.88    68.9333    43.913   98.1765     20.15
y             92.0267       19.4  146.0435  114.8824     64.95
```

The cluster centroid data information is shown in relation to the instance data, showing the final value locations of the centroids for each cluster.

Clustered Instances

Finally, the percentage of the data within each cluster is shown. It gives you an idea of how the data is distributed.

```
Clustered Instances

0      15 ( 20%)
1      23 ( 31%)
2      17 ( 23%)
3      20 ( 27%)
```

Visualizing the Data

The last thing to do is look at the visualization of the clustering. In the explorer window, you see the result list on the bottom left. Right-clicking (or Alt + clicking on the Mac) the SimpleKMeans brings up the visualize window, as shown in Figure 8-9.
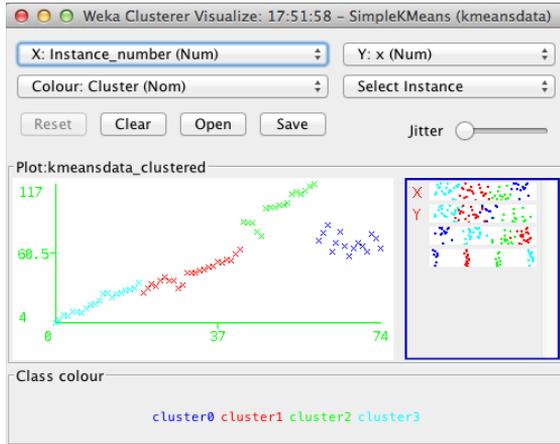
Figure 8-9: Visualize window

Each cluster has its own color scheme, and the plot shows values. Along the top of the visualization window are two, drop-down menus. To alter the plot, select a value from each drop-down menu. To show the x and y instance values, select the X: x(Num) and Y: y(Num) values, and you see the plot update.

Using the workbench method gives you a quick route to some insight, and it is very useful if you want to get a grasp on where the clusters lie. To start automating the process, you need to look at the other two approaches, starting with the command-line method.

The Command-Line Method

The *command-line method* is very similar to the workbench method but it gives you a little more flexibility in terms of running within `cron` jobs. This means that you can collate more data and rerun the analysis on a regular basis.

Converting CSV File to ARFF

Weka uses the `.arff` format to determine object types and have data ready for processing. The GUI enables you to import `.csv` files directly, but it's nice to have a tool that converts files for you.

You can use the `CVSLoader` class on the command line to convert `.csv` files to `.arff`. For example, the `.csv` file you created in the previous walkthrough

was `kmeansdata.csv`. Use the converter by running the following from the terminal command line:

```
java –cp weka.jar weka.core.converters.CSVLoader kmeansdata.csv >
kmeansdata.arff
```

NOTE **If you're using the Windows operating system then you can omit the** `–cp weka.jar` **from the** `java` **command.**

The command-line output might give some warnings about database drivers not being available, but there's no need to worry about that. The main thing is that in the `.arff` file you have the proper definition.

Do a quick inspection to see the following definition:

```
@relation kmeansdata

@attribute x numeric
@attribute y numeric

@data
```

The First Run

Test out the command-line methods by starting with just running the `SimpleKMeans` class as- is on the `.arff` file. This is your training file:

```
java -cp /path/to/weka.jar weka.clusterers.SimpleKMeans -t kmeandata.arff
```

The `–t` flag gives you the name of the training file that Weka is attempting to cluster. Running this as-is gives the following output:

```
kMeans
======

Number of iterations: 3
Within cluster sum of squared errors: 5.839457872519278
Missing values globally replaced with mean/mode

Cluster centroids:
                    Cluster#
Attribute    Full Data         0          1
                 (75)        (35)       (40)
============================================
x              54.88      41.0571     66.975
y            92.0267      45.4286      132.8
```

```
=== Clustering stats for training data ===

Clustered Instances
0      35 ( 47%)
1      40 ( 53%)
```

That works okay, but it's only two clusters and there's a good chance there are more.

Refining the Optimum Clusters

Working out the optimum with the rule of thumb method described earlier in the chapter is easy enough. You can find out the number of object instances using the UNIX wc command.

```
wc kmeansdata.csv
      75      76     494 kmeansdata.csv
```

There are 74 lines (excluding the top line, which gives you the data labels x and y). A quick calculation of 75 divided by 2 results in 37.5, and the square root of that is 6.12.

By altering the command line, you can add that target cluster number (using the –N flag) along with a random seed number to work off (using the –S flag):

```
java -cp /path/to/weka.jar weka.clusterers.SimpleKMeans -t kmeansdata.
arff -N 6 -S 42
```

The output this time gives the same output but with more clusters:

```
kMeans
======

Number of iterations: 3
Within cluster sum of squared errors: 0.523849925862059
Missing values globally replaced with mean/mode

Cluster centroids:
                        Cluster#
Attribute  Full Data    0         1         2         3        4        5
           (75)         (10)      (12)      (15)      (5)      (10)     (23)
==========================================================================
x          54.88        11.8      105.0833  68.9333   81.6     28.5     43.913
y          92.0267      65.9      118.3333  19.4      106.6    64       146.0435
```

```
=== Clustering stats for training data ===

Clustered Instances
0      10 ( 13%)
1      12 ( 16%)
2      15 ( 20%)
3       5 (  7%)
4      10 ( 13%)
5      23 ( 31%)
```

Now, you have six clusters with a good definition of objects going into their specific clusters. The only problem is that you don't know to which cluster the objects belong.

Name That Cluster

From the command line, the –p flag tells Weka to display the assignment of each row's cluster instance. To use this feature, you have to instruct Weka which data attribute to use for each row. From the command line, use the following:

```
java -cp /path/to/weka.jar weka.clusterers.SimpleKMeans -t kmeansdata.
arff -N 6 -S 42 -p 0
```

The –p 0 flag tells Weka to display the row and cluster based on the row number of the data. When this is run, you see the following output to the console:

```
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 4
11 4
12 4
13 4
14 4
15 4
16 4
17 4
18 4
19 4
```

All that's being shown is the row number and the numeric identifier of the cluster to which the row belongs. If you set the -p flag to 1 or 2, then you'd get the value of the x or y positions respectively.

With these workbench and command-line examples, you should now have a good idea how all this is put together. Take a look at the Java coded method using the Weka application programming interface (API). It demonstrates how you can integrate these clustering methods into your own server-side projects.

The Coded Method

The workbench works well, and the command line suits development needs better when scheduled jobs need to be done. But, for ultimate flexibility, there's nothing better than coding your own program using the API to get things done.

The same core Weka classes are used by the workbench, command line, and Java coded examples. It's a case of figuring out how the data is loaded and the clustering done with the options you've used in previous examples.

Creating a k-means cluster in Java is actually a fairly trivial matter; it's a matter of knowing what elements go where. You are going to create a simple Java program to complete what you've done in the previous two walkthroughs. I'll be using Eclipse.

Create the Project

Select File ⇨ New ⇨ Java Project and call it WekaCluster, as shown in Figure 8-10.

There's only one library to install; that is the weka.jar file. On Mac OS X machines, Weka is usually installed within the Applications directory. The location on Windows machines varies depending on the specific operating system.

With the WekaCluster project selected, click File ⇨ Properties and look for the Java Build Path. Then, click the Libraries tab. Add the external jar file by clicking Add External JARs. In the File dialog box, find the weka.jar file, as shown in Figure 8-11.

The last thing to do is create a new class called WekaCluster.java (use File ⇨ New ⇨ Class); see Figure 8-12 for what this should look like.

The Cluster Code

You're going to do the following actions to get your cluster working:

- Write the main method, passing in the location of the .arff file.
- Write a rule of thumb routine to advise the number of clusters you should be aiming for.
- Use Weka's SimpleKMeans class to build the cluster model.
- Print out the location of the centroids of the cluster.
- Print out to which cluster each instance object belongs.

This sounds like a lot of work, but it's actually quite simple. The following sections break down each individual step.
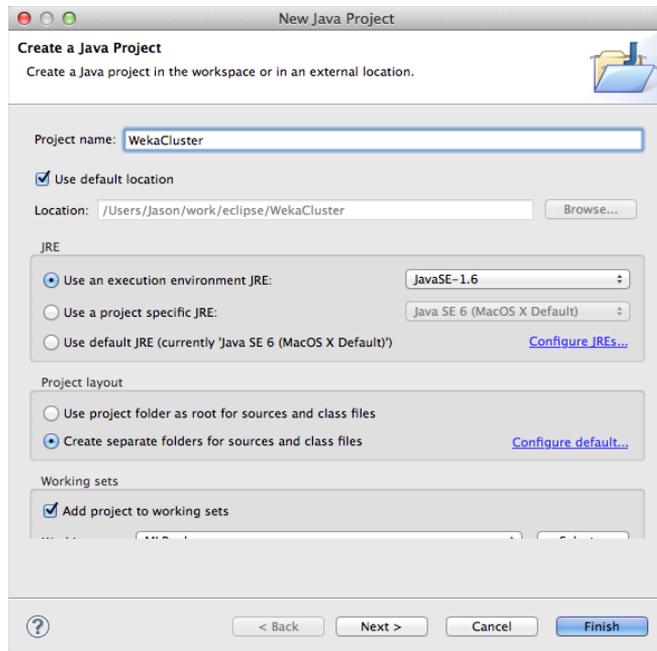
Figure 8-10: Eclipse New Java Project dialog box
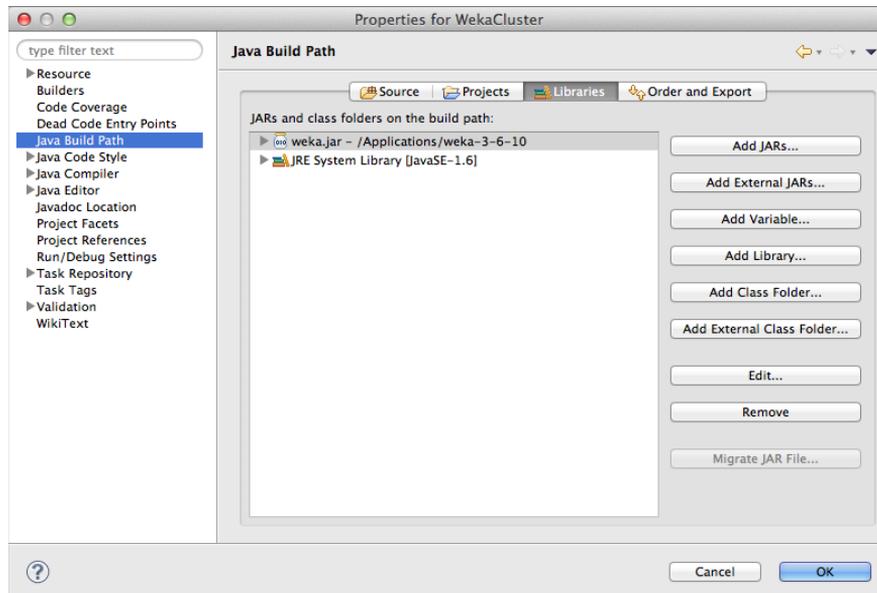
Figure 8-11: Adding an external jar
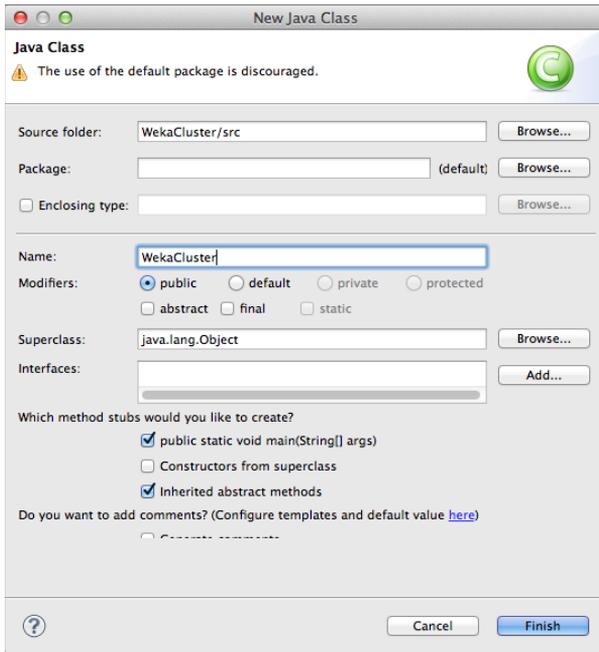
Figure 8-12: Creating a new class file

The main Method

The `main` method is the starting point for the program. It's simple—just one line to create an instance of the constructor and pass in the location of the `.arff` file:

```
public static void main(String[] args) {
    // Pass the arff location and the number of clusters we want
    WekaCluster wc = new WekaCluster("/Users/Jason/kmeandata.arff");
}
```

Because you're passing the filepath as a string, you need to reflect that in the constructor for the class. I talk more about this in a moment.

Working Out the Cluster Rule of Thumb

Earlier, I established that you could quickly estimate the optimum number of clusters to generate. I've included a method to give you the number of clusters to generate based on the instance rows:

```
public int calculateRuleOfThumb(int rows) {
    return (int)Math.sqrt(rows/2);
}
```

The number of rows is passed in as an integer variable. You return the square root of the row count divided by two. If you already know how many clusters you want, just hard code that number.

Building the Cluster

The `main` constructor handles the building of the cluster using the Weka API. As in the previous examples, you're using the `SimpleKMeans` class to build the cluster.

It's a small block of code. Weka handles things for you, so there's not a large amount of preparation to do.

```
public WekaCluster(String filepath) {
    try {
        Instances data = DataSource.read(filepath);

        int clusters = calculateRuleOfThumb(data.numInstances());
        System.out.println("Rule of Thumb Clusters = " + clusters);

        SimpleKMeans kMeans = new SimpleKMeans();
        kMeans.setNumClusters(clusters);
        kMeans.setSeed(42);
        kMeans.buildClusterer(data);

        showCentroids(kMeans, data);
        showInstanceInCluster(kMeans, data);

    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

The data is read in using the `DataSource.read()` method. This takes the string path name (or an InputStream is preferred) and saves the data as instances.

Next, you calculate the number of clusters to define using the method you created earlier with the rule of thumb calculation.

The actual building of the cluster is handled in the next four lines. The `SimpleKMeans` class is the same as the one used in the workbench and command line. You set the number of clusters you want to define (`setNumClusters()`) and a random number seed (`with setSeed()`) and then build the cluster.

Finally, you call two methods: One shows the location of the centroids of each cluster, and the second shows in which clusters the instances are located.

Printing the Centroids

Now that the model is built, you can start to show some results from it. First you print out the location of each cluster centroid.

```
public void showCentroids(SimpleKMeans kMeans, Instances data) {
        Instances centroids = kMeans.getClusterCentroids();
        for (int i = 0; i < centroids.numInstances(); i++) {
            System.out.println("Centroid: " + i + ": " + centroids.
instance(i));
        }
    }
```

The `getClusterCentroids()` method returns a set of instances. It's a case of iterating through these and printing the result of each instance. As six clusters were created (via the rule of thumb method calculation), there should be six instances printed.

Printing the Cluster Information

To show which cluster the instance belongs to, the `showInstanceInCluster()` method takes the k-means model and the assigned instances. The code then iterates each of the instances and prints which it is assigned to based on the model.

```
public void showInstanceInCluster(SimpleKMeans kMeans, Instances data) {
        try {
            for (int i = 0; i < data.numInstances(); i++) {
                System.out.println("Instance " + i + " is in cluster "
                        + kMeans.clusterInstance(data.instance(i)));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

The Final Code Listing

Here's the code assembled and ready to run:

```
import java.util.Random;

import weka.clusterers.SimpleKMeans;
import weka.core.Instance;
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
public class WekaCluster {

    public WekaCluster(String filepath) {
        try {
```

```java
            Instances data = DataSource.read(filepath);

            int clusters = calculateRuleOfThumb(data.numInstances());
            System.out.println("Rule of Thumb Clusters = " + clusters);

            SimpleKMeans kMeans = new SimpleKMeans();
            kMeans.setNumClusters(clusters);
            kMeans.setSeed(42);
            kMeans.buildClusterer(data);

            showCentroids(kMeans, data);
            showInstanceInCluster(kMeans, data);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public int calculateRuleOfThumb(int rows) {
        return (int)Math.sqrt(rows/2);
    }

    public void showCentroids(SimpleKMeans kMeans, Instances data) {
        Instances centroids = kMeans.getClusterCentroids();
        for (int i = 0; i < centroids.numInstances(); i++) {
            System.out.println("Centroid: " + i + ": " + centroids.
instance(i));
        }
    }

    public void showInstanceInCluster(SimpleKMeans kMeans, Instances
data) {
        try {
            for (int i = 0; i < data.numInstances(); i++) {
                System.out.println("Instance " + i + " is in cluster "
                        + kMeans.clusterInstance(data.instance(i)));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        // Pass the arff location and the number of clusters we want
        WekaCluster wc = new WekaCluster("/Users/Jason/kmeandata.arff");
    }

}
```

Running the Program

With the hard work done, you can run the program and inspect the results. From Eclipse select Run ⇨ Run and the program will start. The output in the console window should look something like this:

```
Rule of Thumb Clusters = 6
Centroid: 0: 11.8,65.9
Centroid: 1: 105.083333,118.333333
Centroid: 2: 68.933333,19.4
Centroid: 3: 81.6,106.6
Centroid: 4: 28.5,64
Centroid: 5: 43.913043,146.043478
Instance 0 is in cluster 0
Instance 1 is in cluster 0
Instance 2 is in cluster 0
Instance 3 is in cluster 0
Instance 4 is in cluster 0
Instance 5 is in cluster 0
Instance 6 is in cluster 0
Instance 7 is in cluster 0
Instance 8 is in cluster 0
Instance 9 is in cluster 0
Instance 10 is in cluster 4
....
```

As you can see, the rule of thumb calculation recommended creating six clusters. After executing the k-means clustering method, you displayed the centroid of each cluster in order to "eyeball" the distances of any data point in the cluster from its cluster's center; finally, you displayed cluster membership for each element of our original object data.

Making Predictions

The program so far covers the creation of clusters and reporting the results of the instances. What happens when new data comes in? At present, you're not able to predict anything. It would be nice to have a method you can access that takes new values and predicts in which cluster the result would be grouped.

Instances can be created within code and then run against the clustering model to see where the new values would lie. You need to create another method to return the cluster prediction.

```
public int predictCluster(SimpleKMeans kMeans, double x, double y) {
        int clusterNumber = -1;
        try {
            double[] newdata = new double[] { x, y };
            Instance testInstance = new Instance(1.0, newdata);
```

```
            clusterNumber = kMeans.clusterInstance(testInstance);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return clusterNumber;
    }
```

You're passing the model and the values of the x and y variables (in the same way the original data was in two attributes). A double array is created and the two values are stored.

The `Instance` class is created. The first value is the weight that is to be assigned to the instance. This is a value between 0 and 1. The second value is the double array that you've just created with the x and y values.

In the same way that you showed the cluster by using the `clusterInstance()` method, you run the new instance and get the cluster number. This value is then returned back to the calling method.

To test this, I'm going to create another method, which will iterate 100 times and generate random values. Obviously, in your code you'll be calling the predictor as required.

```
public void testRandomInstances(SimpleKMeans kMeans) {
        Random rand = new Random();
        for (int i = 0; i < 100; i++) {
            double x = rand.nextInt(200);
            double y = rand.nextInt(200);
            System.out.println(x + "/" + y + " test in cluster " +
predictCluster(kMeans, x, y));
        }
    }
```

The method is generating random numbers for the x and y values and passing them to the prediction method. Add this to the `main` constructor after the centroids and clusters are first printed by inserting the line

```
testRandomInstances(kMeans);
```

before the `catch` block is reached in the `WekaCluster` constructor. When you rerun the program, you see the random tests:

```
146.0/167.0 test in cluster 1
109.0/67.0 test in cluster 1
95.0/80.0 test in cluster 3
29.0/160.0 test in cluster 5
165.0/193.0 test in cluster 1
33.0/167.0 test in cluster 5
108.0/73.0 test in cluster 1
63.0/63.0 test in cluster 2
186.0/176.0 test in cluster 1
```

```
67.0/47.0 test in cluster 2
43.0/5.0 test in cluster 2
85.0/9.0 test in cluster 2
152.0/60.0 test in cluster 1
```

Further Development

You will discover putting together a basic cluster algorithm with SimpleKMeans is a fairly straightforward matter. I've covered the main aspects of coding a solution. There are obvious developments from this point, such as connecting to a database table with Java Database Connectivity (JDBC) and extracting the data into instances.

One thing to remember with Weka is that when huge volumes of data are applied, the memory performance can suffer. I suggest that most needs of enterprise are still covered using this method and can be developed with scale in mind. In particular, sampling the data to fit in Weka memory will give very good results.

Summary

Clustering will be one of those machine learning techniques that you'll pull out again and again. To that end, it does need some thought before you go building clusters and seeing what happens.

You've created simple k-means clusters in Weka via the workbench, the command line, and within code. Obviously, there are plenty of options from this point on, but with what you've read in this chapter, you'll be able to get a system up and working quickly.

Chapters 3 through 8 have concentrated on specific types of machine learning, their applications, and some examples in various forms. The following chapters discuss some tools that will become useful to you in data collection and processing. I could even go as far as to say you're about to unlock the door to Big Data.

Machine Learning in Real Time with Spring XD

Consider the amount of data that is being generated as you read this paragraph. Much of that data will be produced, stored, and processed to gain insight and value—for example, a temperature monitor within the house that gives constant updates or a feed from various social media platforms. There's an awful lot of potential in the data that comes out in real time. Being able to capture, process, store, and learn from it can be difficult; but with emerging tools it's becoming easier to put solutions together.

This chapter covers the use of Spring XD for consuming real-time data using the Twitter streaming application programming interface (API). The examples show you how to write custom processors in Spring XD to perform real-time analysis on the incoming Twitter data (tweets).

Capturing the Firehose of Data

Companies that provide continuous streams of data often refer to it as "the firehose"; the data just flows out, and it's up to recipients to capture the data they want and process it as required. Often, some form of agreement with the data provider must be signed before the data is made available for consumption.

Considerations of Using Data in Real Time

Before dashing off to your desk and coding up a real-time application that scans the entire Twitter firehose, it's worth considering if real time is actually the way to go. Just because real-time processing is available doesn't mean you should always use it.

Financial services companies use real-time processing for applications that help subscribers decide whether to perform a trade on a stock; the decision to buy or to sell must be computed in milliseconds. In this context, data that's considered "old"—anything longer than 10 or 20 seconds—is not worth processing when you think about how many transactions might have occurred from other traders. The price could have changed many times within that duration.

On the other hand, an e-commerce site using machine learning to generate recommendations for customers could batch up several transactions and process them periodically every 3, 6, 12, or even 24 hours. Batching up larger volumes of transactions has certain advantages. Chapter 10 covers batch processing in more detail.

Newer processing systems make use of in-memory processing, which requires no slow, traditional secondary data store and obviates the traditional extract, transform, and load (ETL) as found in traditional SQL-based business intelligence systems.

Another consideration is storage volume: Is it necessary to store all the data? Will storing the processed data suffice for your purpose? Or will you need to store the origin data for further processing later? It is true that the cost of storage is decreasing (Moore's Law), but storage still has major cost implications that you must consider. How will the data be stored? Where will it be stored? If the data will be stored on the cloud (on Amazon S3 buckets, for example), then are there privacy concerns that you must address? Have you considered the data safety (backups and restore service levels) and data security (privacy, secrecy, and access control) for the data?

With the increasing speed of computing, the plummeting cost of storage, and the many high-performance, low-cost database systems available, there is nothing to stop you from using two or three data stores for future processing. You can consider a traditional relational database such as MySQL, a column store such as HBase, and a graph database such as Neo4J or Apache Giraffe. Consider your data lifecycle, use cases, and the strengths/purpose of each SQL and NoSQL system within your reach (`http://martinfowler.com/books/nosql.html`).

Potential Uses for a Real-Time System

The applications of a real-time data system are broad and far-reaching. With the expanding volume of valuable social media and mobile data, it's becoming increasingly important for real-time systems to enable instant connectivity

and recommendations for people wherever they are. Location-based targeted advertising is a good candidate for real-time analysis. Such a system must look for offers based on the location of the customer's device in addition to the real-time context of the customer's situation (for example, weather, traffic, time of day, day of year, and local, breaking news).

Financial trading has previously been mentioned as a candidate for real-time processing. Companies are investing heavily into real-time algorithms that can trade at extremely high speeds (microseconds) using algorithmic trading to compute and perform many thousands of trades per second. There are a number of algorithms that can be employed for these calculations. Some systems take into account news headlines and Twitter feeds, and then they trade on the sentiment of the story. Such systems have varying degrees of success. Payments and fraud detection algorithms can analyze transactions in real time and flag problematic transactions as they happen. As processing power increases, companies can embrace more variables such as previous customer transactions, location, and purchase behavior between clicks in soft real time.

In cases where an interactive response to an end user or system is required, a real-time system is worth considering. In such cases, keeping data exclusively in memory speeds things up considerably. Secondary storage should be used only to speed recovery, restarting the memory system, and if more in depth training is required over larger data sets.

Using Spring XD

This chapter uses the Spring XD framework that is designed for real-time processing. The goals of Spring XD are to simplify data ingestion, processing, and data export.

NOTE Data ingestion refers to multiple sources of data, so there's no issue in consuming log data, Twitter streams, and RSS feeds at the same time.

Spring XD runs either on a single server or on a cluster of machines in distributed mode. The examples in this chapter use a single server. The Spring XD software is released under the Apache 2 License and is completely open source, so you are free to download and use it with no restrictions. Spring has done a very good job of including the majority of common use cases in the base release; it's a good starting point to show how real-time analytics can be built quickly and with minimum effort.

If you are aware how the UNIX pipe commands work, then you'll have an easy time understanding how Spring XD functions. If you're not familiar with UNIX pipe commands then please read on; I explain how the basic command pipeline and data streaming in UNIX and Spring XD work.

Spring XD Streams

The Spring XD system is analogous to UNIX command pipelines, where an infinite stream of text is filtered through commands and manipulated in some way before passing the stream on to the next command. For example, in UNIX if I want to compute the frequency of letters, words, and lines from the contents of one or more text files, I can run the "concatenate and print" (`cat`) command on the files and pipe the stream through the word count (`wc`) command:

```
cat *.txt | wc
```

Taking this concept a step further, I might want to alter the output of the resulting word with a label:

```
cat *.txt | sed 's/^/(lines, words, characters) =/'
```

Spring XD streams work in a similar way to UNIX command pipelines. The server reads input data, processes it in stages, and the resulting output is sent to a specific destination. The processing stages are optional, but they become very useful in processing data for machine learning tasks. A single stream is not limited to one processing step; processing can be daisy chained along the stream.

Later in the chapter, there is a full tutorial on how to set up these streams and how to process them. First you should understand the core components in the pipeline stream and how Spring XD uses them.

Input Sources, Sinks, and Processors

Spring XD streams have three main components: an input source, an optional processor, and an output called a sink.

Input Sources

Using the Spring Integration Adaptors, the XD system provides a number of ready-to-go *input sources*. These cover a range of uses from Internet-based protocols, internal log output tools, and file-processing commands. Table 9-1 summarizes different types of input sources.

Table 9-1: Different Types of Input Sources for Spring XD

INPUT NAME	DESCRIPTION
HTTP	Reads the input data from the HTTP request—for example, a web page, RSS feed, or REST API call.
TCP	Handles the output of a raw TCP socket.
Mail	Reads incoming mail from an IMAP server.

INPUT NAME	DESCRIPTION
JMS	Receives incoming messages from a Java Message Service.
RabbitMQ	Subscribes to and receives incoming messages from a RabbitMQ server.
Twitter Stream	Uses the Twitter streaming API and reads in the JSON payload of each tweet.
Twitter Search	Uses the Twitter Search API and reads in the JSON payload of each tweet.
File	Reads a stream from a file.
Tail	Reads the tailed output stream from a given file.
MQTT	Connects to an MQTT (Message Queue Telemetry Transport) server; subscribes to and receives telemetry messages.
Time	Emits a periodic "heartbeat" time stamp string of the current time as perceived by the system on which Spring XD is running, with a defined duration between heartbeat messages.
Gemfire	Listens to either region events or continuous queries from a Gemfire server.

Each of the input sources comes with an array of configuration items that might be required before use in each source's stream definition. Some of these input sources and how to set their configurations is covered later in the chapter.

Sinks

To be useful, a system must produce an output, or *sink*, of some form. The most common output sinks are log files or a database table. Although Spring XD supports these log files and database tables, there are other options available. Table 9-2 lists output sinks.

Table 9-2: Output Sinks in Spring XD

OUTPUT NAME	DESCRIPTION
File	Output is appended to a text file on a file system.
Log	Using the internal logging function, output messages are written with INFO/WARN/ERROR as Log4J style, with rotation and time stamps.
JDBC	Saves output data to a relational database. Any database with a JDBC driver can be used. Default database is in memory HSQL.
Mail	Routes output to an SMTP mail server.
TCP	Output is routed to a TCP socket—for example, output could be routed to the `netcat` UNIX command.

Continues

OUTPUT NAME	DESCRIPTION
HDFS	Stores output data to the Hadoop Distributed File System. This is covered in more detail in Chapter 10.
RabbitMQ	Outgoing message is sent to a RabbitMQ exchange.
Splunk Server	Spring XD converts output to a SplunkEvent and is sent via TCP to a Splunk server.
Gemfire Server	Data is written to a running Gemfire Cache server. This can be to either the standard server or the JSON server.
MQTT	Output telemetry data is sent to a configured MQTT server.

Other sink output channels can be created if they are supported in the Spring Integration project.

Processors

So far, this chapter has covered the input and output data types. You could easily create Spring XD streams to read an input source and then pipe the data to an output source. *Processors* sit between the input and output sources and allow additional processing, parsing, and analyzing of the data as it passes through.

Spring XD comes with a set of ready-to-use processors (see Table 9-3); they offer some basic filtering, data extraction, and string manipulation.

Table 9-3: Spring XD Built-In Processors

PROCESSOR NAME	DESCRIPTION
Filters	Performs a grep-like expression filtering on the stream. The filter can be either a Spring Expression Language (SpEL) expression or a Groovy script.
JSON Field Value	Passes message through if the value of a JSON field matches.
JSON Field Extractor	Extracts the value of a JSON field and streams the value to the output.
Transform	Converts the input source message content and sends it on to the output.
Split	Consumes the message and splits it into a number of messages based on an expression.
Aggregator	Concatenates message payloads together a number of times to create one aggregated message.

Although Spring XD has comprehensive built-ins, there are plenty of options for customization creating new input sources, sinks, and processors.

Learning from Twitter Data

The remainder of the chapter outlines a solution for analyzing data from Twitter. The Twitter streaming API provides plenty of data that can be filtered down to a reasonable size when you track a tiny subset. It also gives the perfect platform for learning how Spring XD works, how to create streams, and how to start customizing processors for your own needs.

If you haven't already installed Spring XD, please refer to the "Software Used in This Book" section of Chapter 1. There you can find instructions on where to download and how to install Spring XD.

The Development Plan

This section includes step-by-step instructions for building up the full implementation of learning from the Twitter streaming data.

Step 1: Basic Streams in Spring XD

The first step in the project is to get each element up and running. When you know each element is working and data can be saved to a file, then you can move forward with filtering. The basic concepts are

- Setting up the Twitter API developer application
- Setting up the Spring XD Twitter credentials
- Testing Spring XD with a simple demo stream
- Configuring the first stream to consume Twitter data
- Emitting data to a file sink

Step 2: Developing a Processing Module

The second step introduces the development of your first processor:

- Writing the processor code
- Writing the XML configuration item
- Installing the code and configuration in Spring XD
- Reconfiguring the stream to include the new processor

Step 3: Developing a Sentiment Analysis Module

The last step involves developing a more involved processor to perform sentiment analysis on the incoming Twitter stream. You also configure Spring XD to

preserve the raw Twitter data before it's processed, in case you want to do further analysis after the fact. This step can be broken down into the following tasks:

- Writing the sentiment analysis processor
- Writing the XML configuration
- Installing the sentiment analysis code and configuration in Spring XD
- Reconfiguring the stream to hook up the existing processor to the sentiment analysis processor
- Configuring a second channel called a "Tap" enables you to store the streaming data while piping it to the sentiment analysis processor.

If the list seems overwhelming, don't worry; I've provided step-by-step instructions and code samples along the way. The full code and configuration data are also available from the Wiley website `http://www.wiley.com/go/machinelearning`.

Configuring the Twitter API Developer Application

Before you can start consuming data from Twitter into Spring XD, you need to set up a development account and install the Twitter development environment. This setup process uses Twitter's developer site (`http://dev.twitter.com`) and requires you to have an existing Twitter account. If you don't have a Twitter account, you can sign up for one at `http://www.twitter.com`.

The developer website enables users to create keys for their applications to access Twitter's APIs. I'm going step-by-step, assuming you've never done it before. If you already know how to create Twitter developer access and the required credentials, then you can skip the remainder of this section.

1. Open a web browser and go to `http://dev.twitter.com`. Log in with your Twitter credentials.

2. Find your Twitter avatar image at the top right. Hover your mouse pointer over the arrow next to it. Click the My Applications link in the drop-down menu.

3. Click the Create a New Application button, as shown in Figure 9-1.

4. Fill in the required fields for the application name, description, and a URL for your website (see Figure 9-2). This information is required if Twitter users decide to use their accounts with your application. You can leave the callback URL blank, because you won't be using it.

   You need to agree to Twitter's terms and conditions as well as fill in a Captcha code. Click the Create Your Twitter Application button.
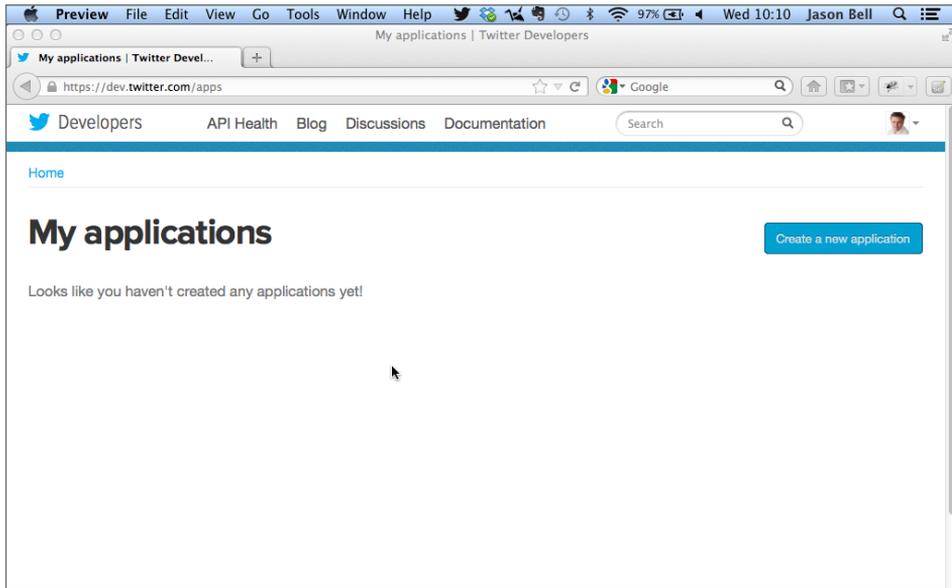
Figure 9-1: Creating a new Twitter application page
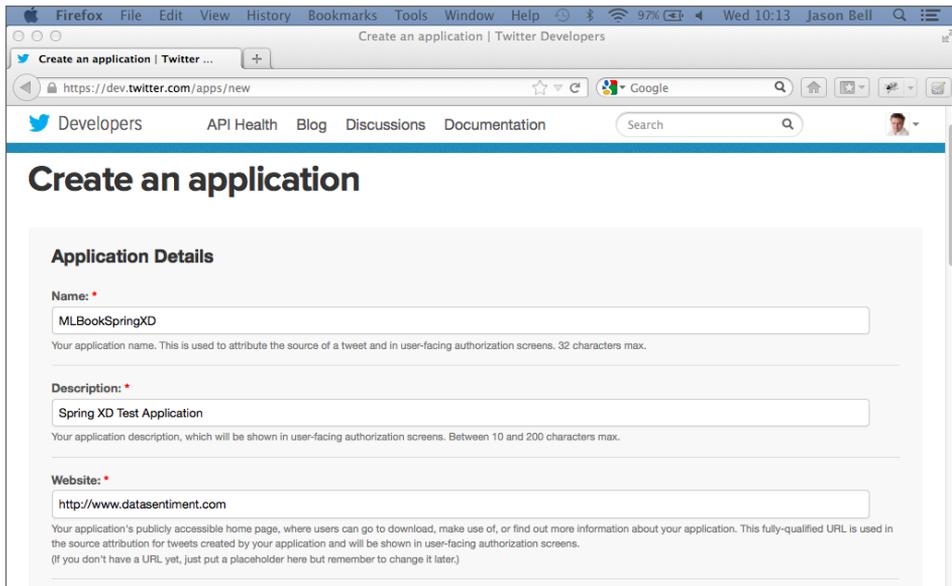
Figure 9-2: Completing the application detail page

Assuming that all went well, you are directed to the Details configuration screen of your application. These settings include organization, authorization, settings, access keys, and an option to delete the application.

5. In the OAuth Settings area of the Details tab (see Figure 9-3) is the information you require for the Spring XD system to collect Twitter streaming data. Make a note of the Consumer Key and the Consumer Secret.

6. Twitter requires an access token to go with your consumer key and secret. The easiest way to get one is to click the Create My Access Token button below the OAuth settings. It can take a moment for the access token to appear in the Details page. If it takes longer than 30 seconds or if the access token remains empty, refresh the page in your browser and the tokens appear.
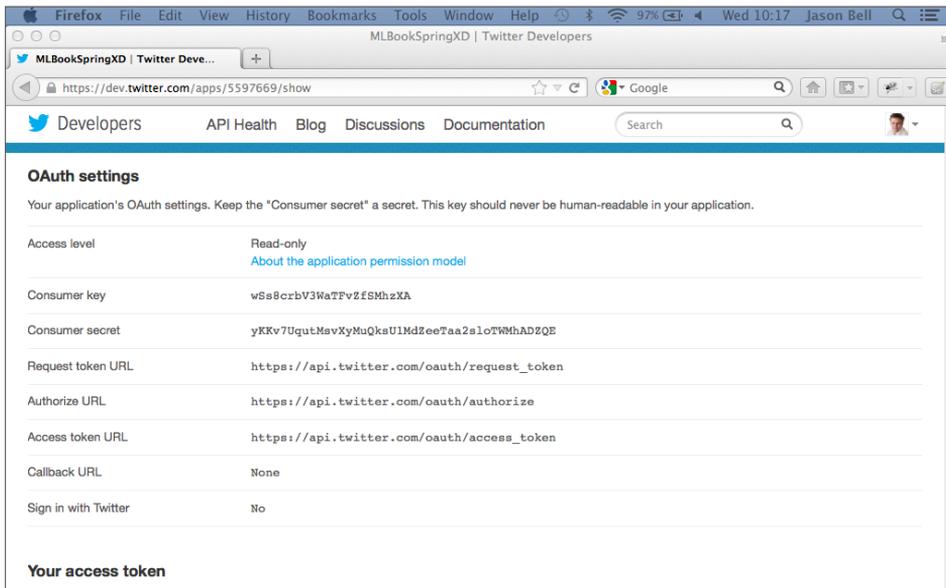
Figure 9-3: OAuth details

When the process is complete, you see the access token, an access token secret key, and the access level. A read-only token is fine for your purposes, because you'll be consuming data and not writing new tweets.

Configuring Spring XD

Now that you have the Twitter API keys set up for your application, you can set up Spring XD itself. The XD distribution has a fair number of components.

In the main directory you can find directories for Gemfire and Redis. You can install these separately; they aren't required for this walkthrough.

You'll be using the server directory xd and the command-line program directory called shell.

Starting the Spring XD Server

Within the Spring XD server directory called xd is a subdirectory called bin. The bin directory contains the scripts used to administer the XD server. For the first few examples, you start the server in single or "stand alone" mode. This mode configures and starts Spring XD to run on one machine. After you are more familiar with the framework, you can configure it to run on a cluster of machines.

In a UNIX shell, change to the directory where you installed Spring XD and descend into the xd/bin subdirectory. Start the server using the xd-singlenode shell script:

```
$ cd spring-xd
$ cd xd/bin
$ ./xd-singlenode
```

Within the terminal window where you issued the command, you see Spring XD emit diagnostic information about starting its required modules. When you see the string started container as shown in Figure 9-4, then the Spring XD server is ready to accept commands.

Figure 9-4: Spring XD server startup

You can stop the server by typing Ctrl + C in the terminal window where the server was started. Ctrl+C stops the server if it is not running as a background task in the terminal window.

If the Spring XD server process is running in the background, you can send it a TERM(inate) signal using the UNIX `kill` command. First display all `springxd` processes running to identify the process ID of your server, then use the `killall` UNIX command interactively to send the terminate signal to your server process.

```
$ ps –auxw | grep springxd
$ killall –i –v –TERM java
```

Creating Sample Data

Before you run the shell you are going to create a stream of data to read in. Assume that you want to read the uptime of the machine that you are working on. First you would write the contents of the uptime command to a file in the `/tmp` directory:

```
$ while true ; do uptime >> /tmp/xdin ;  sleep 1 ; done &
```

The output file `xdin` contains the uptime output updated every second.

```
19:07:47 up 35 days,  7:49,  1 user,  load average: 0.00, 0.00, 0.00
 19:07:48 up 35 days,  7:49,  1 user,  load average: 0.00, 0.00, 0.00
 19:07:49 up 35 days,  7:49,  1 user,  load average: 0.00, 0.00, 0.00
 19:07:50 up 35 days,  7:49,  1 user,  load average: 0.00, 0.00, 0.00
```

The Spring XD Shell

Now that the server is running you can use the client command-line shell program that comes with Spring XD. To run the interactive shell, open another terminal window, descend into the directory where you installed Spring XD, and run the `xd-shell` script.

```
cd /usr/local/springxd/
cd shell/bin
./xd-shell
```

The Spring XD shell prompt is `xd:>`. (See Figure 9-5.) From this command-line shell, you can issue the server commands as well as create and control streams. There are also commands to handle modules, jobs, built-in aggregate counters, and even Hadoop. (The next chapter includes more on Hadoop.)
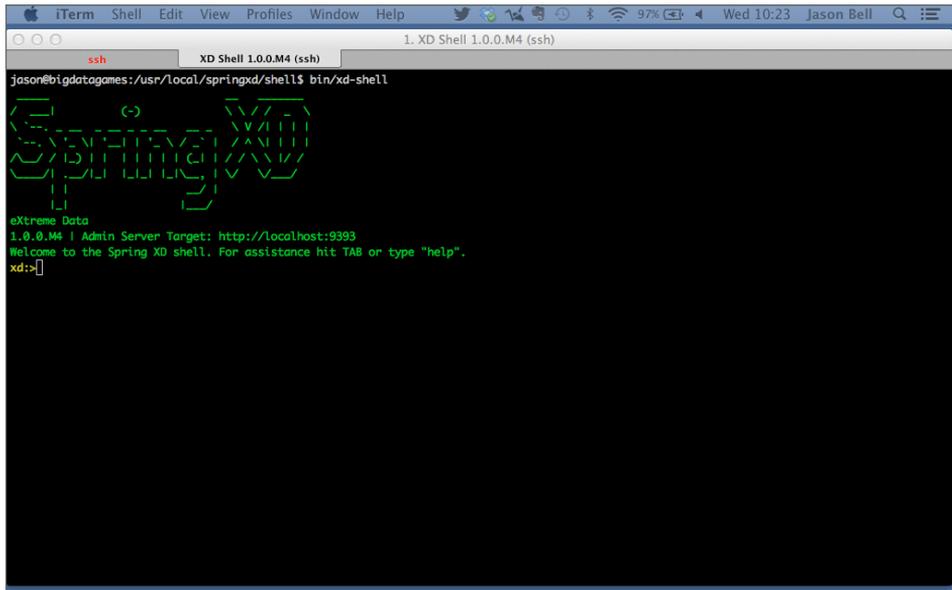
Figure 9-5: Spring XD shell

To exit the shell, type **exit** and press the Enter/Return key.

Streams 101

This section covers the concept of streams in Spring XD and looks at various types of streams that can be used. You find out how to create, delete, and list active streams within Spring XD.

Creating Streams

Before you dive in to working with a Twitter stream, try creating a simple stream to get comfortable with the concept. Imagine you want to log a series of timestamps to a file. Within the XD shell, you run the following command:

```
xd:>stream create --name myfirststream --definition "tail --name=/tmp/
xdin"
```

I'm going to break this command down in sections. First you're creating a new stream:

```
stream create
```

Each stream requires a name in order to monitor and manage several streams at once:

```
--name myfirststream
```

The definition is the process flow; this clause is required and defines the actions and pipeline of stages. In this instance, you're executing the Java equivalent of the UNIX `time` command and piping it to Spring XD's log:

```
--definition "tail --name=/tmp/xdin"
```

When you run the command, the XD shell returns with a response of:

```
Created new stream 'myfirststream'
```

Stream names are unique; if you attempt to create a stream with an existing name, you receive the following error message:

```
Command failed org.springframework.xd.rest.client.impl
.SpringXDException: There is already a stream named 'myfirststream'
```

As soon as the stream is created, the server deploys the stream and starts processing. If you switch to the terminal where you started the server, you see the log output of the stream on `stderr`:

```
17:06:03,961  WARN task-scheduler-8 logger.myfirststream:145 - 2013-12-
26 17:06:03
```

Note that Spring XD's default log level is `WARN`—a warning level of logging to the `stderr` sink. In this case the error output is diagnostic and there is nothing to be worried about. Spring XD is processing the stream correctly and logging information to the `stderr` sink.

Listing Streams

To list existing streams, run the following command:

```
xd:>stream list
```

A list of the streams, their names, definitions, and current status (deployed or undeployed) will be displayed as shown in Figure 9-6.

Deploying and Undeploying Streams

Streams in the list can be deployed or undeployed at any time. When streams are created, they are deployed by default, and the server starts executing and processing the stream. To halt the stream from running, you can undeploy it:

```
Xd:>stream undeploy --name myfirststream
```

Figure 9-6: Stream XD stream lists

When you list the streams after this `undeploy` command, the name and definition are shown but the status is blank. To redeploy the stream and start processing it again, run the `stream` command again:

```
Xd:>stream deploy –name myfirststream
```

After you deploy a stream, you can check the server output log to ensure the stream is running.

Deleting Streams

When you no longer require a stream, you can safely delete it from Spring XD. You do this from the shell with the `destroy` command.

```
Xd:>stream destroy –name myfirststream
```

The stream undeploys itself from the server and then is removed from the list. In order for that stream to work again it must be re-created.

Storing Stream Definitions

By default, Spring XD stores definitions and state information in memory and does not persist this data to disk. The information is stored only for the duration

of time the server is running. When the server is stopped, any definition data is lost and must be re-created when the server starts again. You can persist this data to disk using the Redis key-value store. Connection and other configuration details for Redis are set in the `redis.properties` file in the server configuration directory. A known-working version of Redis is part of the Spring XD distribution (for Linux or Mac OS X), but it must be compiled before you can use it. If you already have a version of Redis available then you can most likely use that version. For this walkthrough, the in-memory storage will suffice. In a production situation, persisting the streams in Redis is probably the better way to proceed.

Thus far, this chapter has covered streams and how they are created, deployed, listed, and deleted. Now you can move on to working with Twitter data streams.

Spring XD and Twitter

One of the more popular uses of real-time processing is to read and analyze social media data. The volume of opinion, sentiment, and information broadcasts make this kind of data the perfect use for a tool such as Spring XD.

Setting the Twitter Credentials

You need to set up the Twitter credentials in your Spring XD configuration. Locate where you installed Spring XD and go to that directory.

```
cd /usr/local/springxd
```

The Twitter configuration file is saved as a template within the `xd/config` directory. You need to make a copy of this file (or rename it), so Spring XD recognizes it when you start the server.

```
cp twitter.properties.template twitter.properties
```

Edit the `twitter.properties` file and add the consumer key, consumer secret, access token, and access token secret using their respective key names, as shown in Figure 9-7.

Save the file and exit your text editor.

The assumption is that Spring XD will be using only one Twitter user to consume all the Twitter data. If you are thinking of letting more than one Twitter user retrieve data via the API, then you need to add the credentials while creating the Spring XD stream.
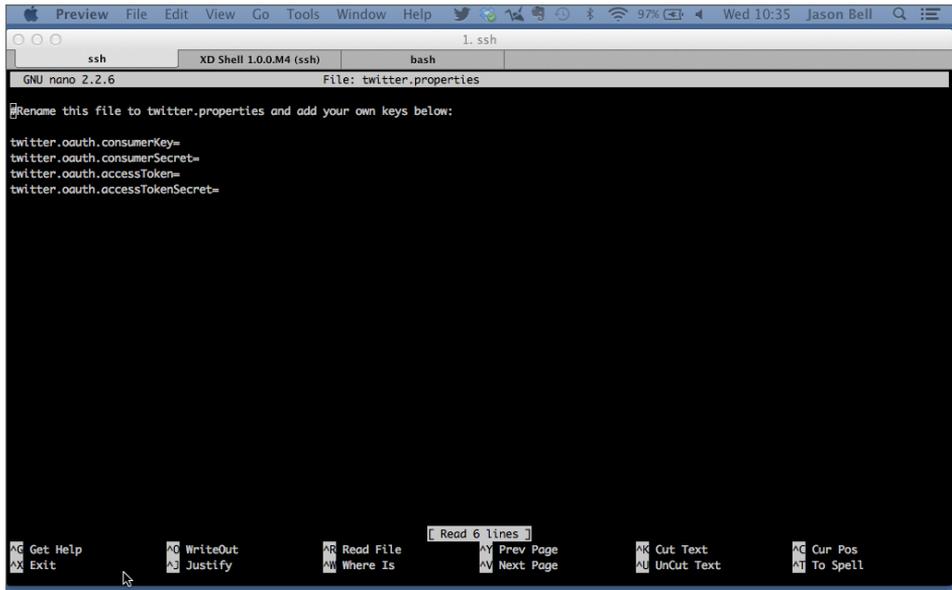
Figure 9-7: Twitter properties file for Spring XD

Creating Your First Twitter Stream

Creating a Twitter stream is similar to the streams covered in the "Streams 101" section. It's just a case of creating another definition.

Streaming Twitter data is based on the public tweets of its users. It's not the complete firehose of all tweets; instead it is a very small slice of everything on Twitter from the endpoints "sample" and "filter." Be very careful of the amount of data the stream produces without filtering for specific keywords; you will consume a huge amount of data in a very short time.

The Twitter output is in JSON notation and contains a lot of information about the date, time, user, the tweet itself, retweets, location (if available), and the reply. Unless the intention is to collect, store, and process all the information (assuming you have the storage capacity), using the public stream is good. If you want to have a certain amount of control of the data coming in, then tracking specific key terms is a better way to use the stream.

Create the Stream Definition

To create your first Twitter stream, run the following stream command from the shell:

```
xd:>stream create --name mytweetstream --definition "twitterstream
--track='#fashion' | file"
```

You should already be familiar with the general stream definition. Within the definition is where the work with the Twitter API happens. The `twitterstream` keyword tells Spring XD that you want to use the Twitter API. The `--track` flag is telling the API what streaming data it wants to receive. This could be a hashtag or a specific user mention, or, as in this instance, a keyword (for this example, the word `fashion`).

When the stream is deployed, check the server log to make sure it's registered correctly:

```
18:49:43,248  INFO http-bio-9393-exec-9 module.SimpleModule:137 -
initialized module: SimpleModule [name=twitterstream, type=source,
group=myfirsttwitter, index=0 @60cde43d]
18:49:43,266  INFO http-bio-9393-exec-9 module.ModuleDeployer:231
- deployed SimpleModule [name=twitterstream, type=source,
group=myfirsttwitter, index=0 @60cde43d]
```

If the Twitter credentials are incorrect, the Spring XD displays the error in the server's console and also relays the error response from the Twitter API. It's always worth keeping an eye on the console log output at these early stages.

Twitter Stream Definition Flags

I briefly mentioned the `--track` flag within the definition, but Table 9-4 lists a few others of which you should take note.

Table 9-4: Streaming API Flags for Spring XD

FLAG NAME	DESCRIPTION
`--track`	One or more Twitter "hashtags" to track. If you want to monitor more than one hashtag, separate them with commas.
`--follow`	A list of user IDs to track. The stream returns tweets based on matching users. To track more than one user, separate usernames with commas.
`--locations`	Tracks tweets within a pair of longitude/latitude points. Note if the `locations` flag is used with the `track` flag, matching tweets on either flag will be returned—that is, tweets matching either one or the other, not tweets matching both.

FLAG NAME	DESCRIPTION
`--delimited`	Setting this parameter to the string `length` indicates that statuses should be delimited in the stream, so that clients know how many bytes to read before the end of the status message. Statuses are represented by a length, in bytes, a newline, and the status text that is exactly length bytes. Note that "keep-alive" newlines might be inserted before each length. It can also be set to true or false to specify if delimiters are desired.
`--stall-Warnings`	Sends warning messages to the client if the client risks being disconnected.
`--filter-Level`	Sets the level of tweets given back to the user. When set to `none`, then all available tweets are sent back.

The `--stallWarnings` and `--filterLevel` flags have different parameters when accessing the Twitter API directly. It's worth experimenting with a few streams and setting different values for these flags to see how the output behaves.

Inspecting the Output

Where the sink output is set to "file," the file stream is stored in the `/tmp/xd/` output directory. The filename is determined by the name of the stream created. So, for the Twitter streaming example, the output is

```
/tmp/xd/output/mytweetstream.out
```

It's useful at this point to use the UNIX `tail` command to see the velocity of the output. When developing Spring XD streams, especially for Twitter applications, I like to have three terminal windows open: one for the server, one for the client shell, and another tailing the output in the temporary directory, as shown in Figure 9-8.

Where to Go from Here

With regard to the development plan at the start of the chapter, you've completed Step 1: getting the basics of the Spring XD server and the Twitter credentials up and running. You've gotten the basic stream up and running and inspected the output.

At this stage you're only consuming data and then choosing a method to store the output. Now it's time to move on to processors and manipulating the data in real time.

Figure 9-8: JSON output from the Twitter stream

Introducing Processors

So far, Spring XD is consuming the incoming stream of data and saving it to a file. For some purposes, this step suffices to write further software that would parse, perform analysis, and mine the data as required. One of the benefits of Spring XD is that you can extend the stream to process the incoming data with processors as it comes in without paying the heavy overhead of file I/O. This set of features also saves writing a custom program that is external to the system and means that most of the work can be performed as Spring XD is handling the data, which prevents technical debt and maintenance.

How Processors Work within a Stream

So far when creating streams, you have set an input source and an output type. Data enters via Spring XD and can be written or logged depending on the output sink you've specified.

The processor is another stage in the stream definition, so instead of having what's shown in the top half of Figure 9-9, you can add a processor in the stream and transform the data as it flows, as shown in the bottom half of Figure 9-9.
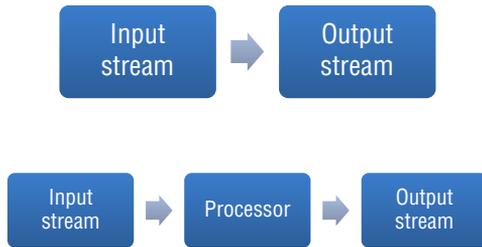
Figure 9-9: The processor information flow

The nice thing about processors is that you can chain many of them together in the stream. With built-in processors, you can start transforming the data very quickly. For example, if you want to extract only the tweet text from the incoming Twitter stream then you can use Spring XD's JSON field extractor; pulling only the content becomes straightforward:

```
stream create --name prctest --definition "twitterstream
--track='BigData' | json-field-extractor --fieldName=location | file"
```

This is very helpful for selecting single fields that you want to extract. If you want to take things further, then crafting your own processor is the better way to go.

Creating Your Own Processor

This section walks you through creating a processor, building it, and then deploying it on Spring XD. At present, you're consuming far too much Twitter data from the stream and it's using up precious storage space. The JSON field extractor is a good start but only extracts one field name and becomes limited when extracting huge amounts of JSON data. A processor module to extract the required fields of the Twitter stream and forward those to the output sink or the next stage in the pipeline is the purpose of this example. After this step is completed, you'll extend it further to perform some simple sentiment analysis. To understand the steps to create a module and deploy it, I'm going to cover the basic data extract first and then refactor the code.

Spring XD expects all modules to be deployed within `jar` files. Ensure that the full package structure is saved within the `jar` file, otherwise Spring XD does not deploy the processor. The `jar` files are stored under the `/lib` directory of your Spring XD distribution.

The application context is saved as an XML file but not saved within the `jar` file. It contains the module details of the incoming and outgoing channels and

which Java class to use to perform the processing. The context file is saved under the `/modules/processors` directory of your Spring XD distribution.

This walkthrough uses Eclipse to enter the code and configuration. The general concept and steps would be the same if you were using another IDE such as Netbeans or IntelliJ IDEA. The complete source code for the Spring XD processor modules is on the companion web page for this book.

Creating the Project

Within Eclipse, create a new Java project using File ⇨ New ⇨ Java Project. It will hold all the package information, the code, and a directory for the required libraries to read the incoming JSON data.

Call the project `XDTwitterProcessor`, as shown in Figure 9-10, and check the default settings. When you are happy with them, click the Finish button. Eclipse creates the project structure for you.
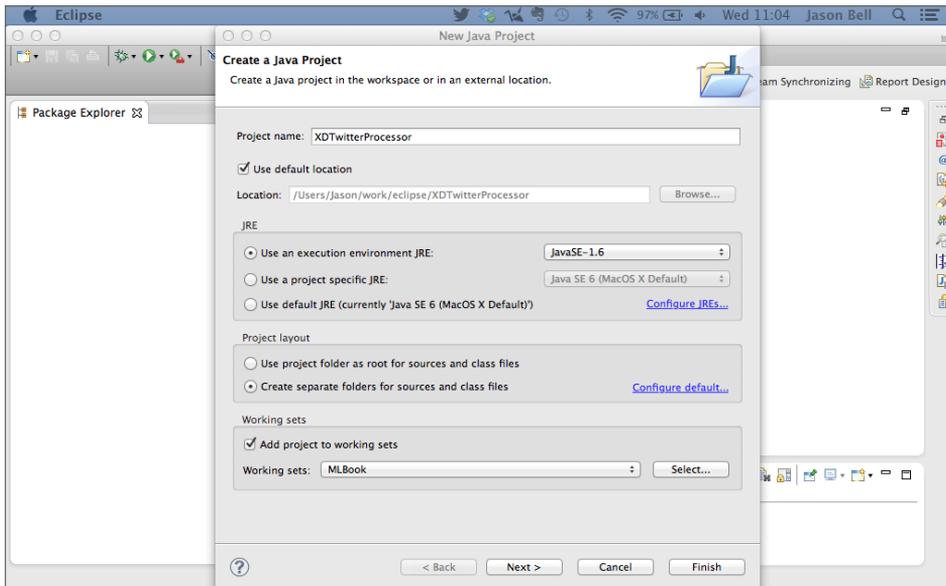
Figure 9-10: Eclipse new project dialog

Next find the location of the Jackson JSON Parser `jar` files and the Spring Integration `jar` file. They are usually under `xd/lib` in the Spring XD distribution. The `XDTwitterProcessor` project properties need the locations within the Java build path. Otherwise, trying to compile your project results in errors. To add these locations to your project's build path, right-click the project name and select Project Properties.

Click Java Build Path on the left-hand side list of options and then click the Add External JARs button, as shown in Figure 9-11. In the dialog box that opens, find the location of the required `jar` files and select the following (note the version numbers might vary from release to release):

- `jackson-core-2.2.2.jar`

- `jackson-annotations-2.2.2.jar`

- `jackson-core-asl-1.9.13.jar`

- `jackson-databind.2.2.2.jar`

- `jackson-mapper-asl.1.9.13.jar`

- `spring-integration-core-4.0.0.M1.jar`
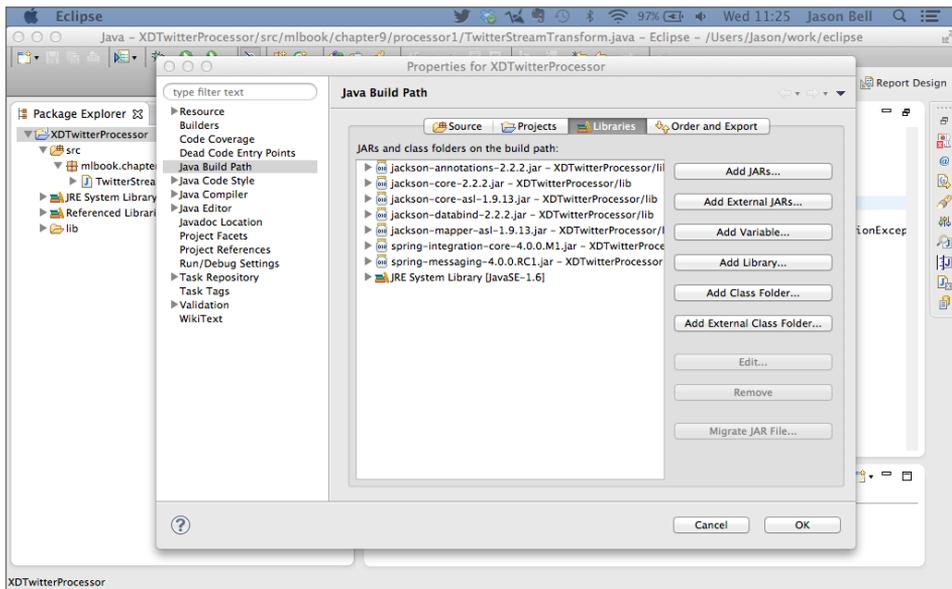
- `spring-messaging-4.0.0-RC1.jar`

Figure 9-11: Eclipse project Java build properties

After you have selected the required `jar` files, click Open and return to the Java Build dialog box. Then click OK to save your selections.

A Note to Maven Users

Developers who use the Maven build tool can add the two dependencies in the build file:

```
<dependency>
    <groupId>org.springframework.integration</groupId>
    <artifactId>spring-integration-core</artifactId>
    <version>2.2.3.RELEASE</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.2.0</version>
</dependency>
```

This retrieves the required libraries when you build the project. If you're not familiar with the Maven build tool, you can learn more at `http://maven.apache.org`.

Writing the Code

Now that the project is set up, you can proceed to writing some code. First of all, create a new package in the project (using File ⇨ New ⇨ Package) and give it the name `mlbook.chapter9.processor1`.

Next, create a new Java class (using File ⇨ New ⇨ Class) and give it the name `TwitterStreamTransform`, as shown in Figure 9-12.
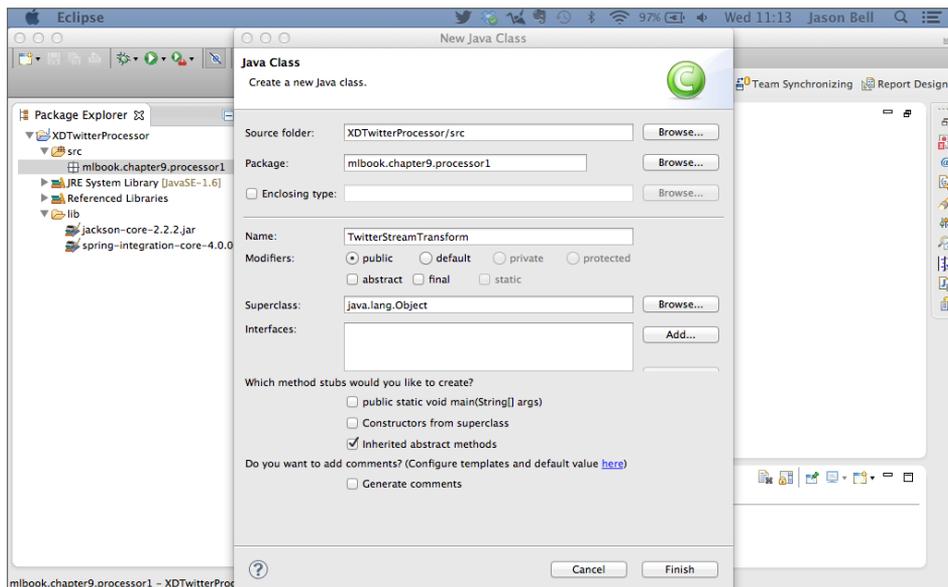

Figure 9-12: Eclipse new class dialog box

Eclipse creates the bare template of the class file for you with the package name and the class definition. You can either delete it all and type the following code or fill in the required segments around the generated template. Don't

forget to organize the imports (Source ➪ Organize Imports), and Eclipse looks after the classes required to import from the external libraries for the project.

```java
package mlbook.chapter9.processor1;

import java.io.IOException;
import java.util.Map;

import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.type.TypeReference;
import org.springframework.integration.transformer
.MessageTransformationException;


public class TwitterStreamTransform {
    private ObjectMapper mapper = new ObjectMapper();

    public String transform(String payload) {
        try {
          StringBuilder sb = new StringBuilder();

          Map<String, Object> tweet = mapper.readValue(payload,new
TypeReference<Map<String, Object>>() {});
          sb.append(tweet.get("created_at").toString());
          sb.append("|");
          sb.append(tweet.get("text").toString());
          return sb.toString();
        } catch (IOException e) {
          throw new MessageTransformationException(
          "[MLBook] - Cannot work on this tweet: " + e.getMessage(), e);
        }
    }
}
```

When the Twitter API sends JSON data to Spring XD, the incoming stream is called `payload` and is defined as a `string` type. The method `transform` takes the incoming stream and uses the Jackson JSON parser API to convert it to a `Map`. The JSON field names now become map keys and the values are Java objects.

For this example you're concerned only about two elements: the date the tweet was created and the actual text of the tweet.

Using the `StringBuilder` class, the outgoing string is constructed and uses a pipe character (|) to separate the two values; the resulting string is then returned to the stream.

Writing the Application Context

Before you can deploy the processor code, you need to create the application context file so Spring XD can recognize the class you've created. Although

the XML file isn't deployed with the final `jar` file, it's worth creating the XML file within the project so everything is together while you are working on the development.

Create a new file (using File ⇨ New ⇨ File) and call it `twitterstreamtransformer` `.xml`. The file should be in the `src/mlbook/chapter9/processor1` directory under the `XDTwitterProcessor` project directory.

Add the XML markup and ensure that the start and closing tags are in place. Otherwise you'll have problems later when trying to deploy.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/
integration"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/integration
    http://www.springframework.org/schema/integration/spring-
integration.xsd">
  <channel id="input"/>

  <transformer input-channel="input" output-channel="output">
    <beans:bean class="mlbook.chapter9.processor1.
TwitterStreamTransform" />
  </transformer>

  <channel id="output"/>
</beans:beans>
```

Spring XD looks at the XML configuration and loads the specified `bean` class within the `transformer` tags. The filename of the XML file is the name that will be used in the stream definition. You'll see how it all fits when you test the processor module.

Exporting the jar File

The final step in the development process is to export the `jar` file. Within Eclipse it's a straightforward case of exporting the project. To export to a `jar` file, select File ⇨ Export and you see the Jar File option under the Java folder, as shown in Figure 9-13.

Click the Next button, and you see the Jar File Specification panel. Ensure that you have the right project selected and that the Export All Output Folders for Checked Projects check box is selected. Give the `jar` file a name, such as

XDTwitterProcessor. Finally, select a target destination folder for your `jar` file and click Finish. (See Figure 9-14.)
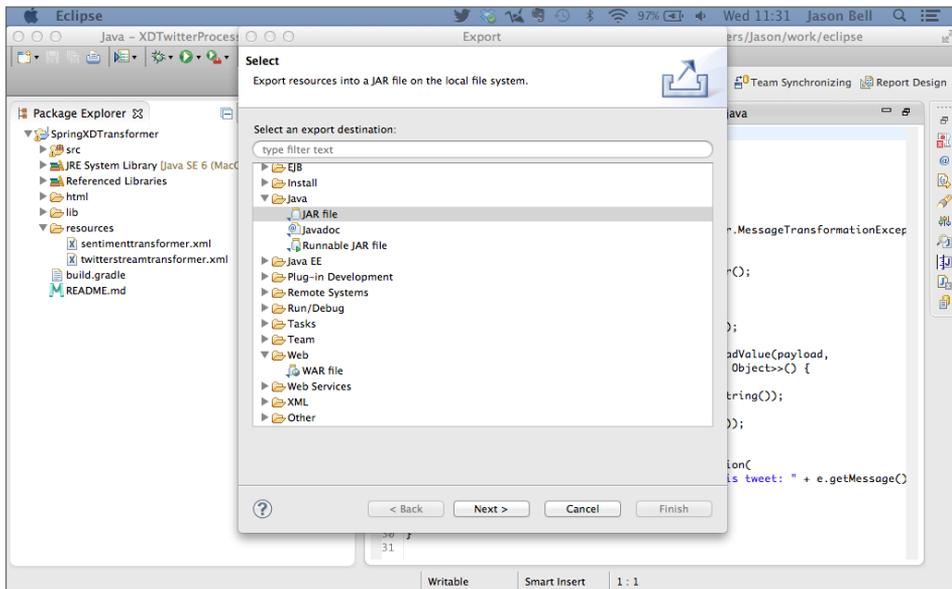

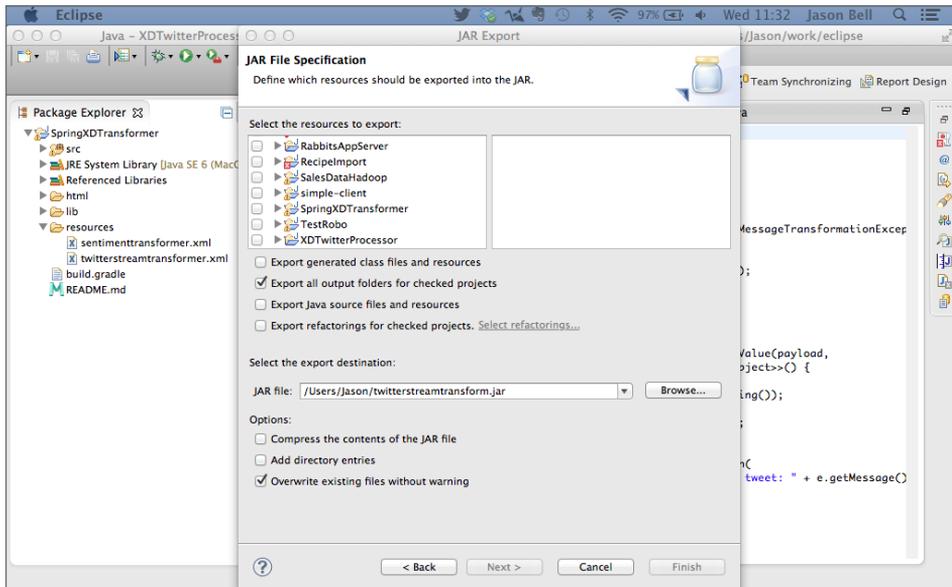**Figure 9-13:** Export selection dialog

Figure 9-14: Jar file detail dialog

The project is now complete. You've created a project with the processor module code and the required XML configuration for Spring XD. Next you'll deploy the project into Spring XD and test it.

Deploying the Project into Spring XD

Before you can test your new module, you need to deploy it within Spring XD. If your Spring XD server is running, you need to shut it down, deploy the module, and then restart for your changes to take effect.

First, copy the `twitterstreamtransformer.xml` file to the processor directory:

```
cp ./twitterstreamtransformer.xml \ /usr/local/springxd/xd/modules/
processor
```

Next, copy the `jar` file you exported within Eclipse and copy that to the `lib` directory:

```
cp ./twitterstreamtransformer.jar /usr/local/springxd/lib
```

With those two files in place, you can restart your Spring XD server. Keep a close eye on the output while Spring XD initializes. If you see any exception traces, it's worth checking the XML context to see if it's well formed and the package and class names are correct.

Assuming everything has deployed okay, you can test your new module.

Testing the New Processor Module

Now it's time to see the fruits of your labor; after the development and the deployment comes the testing. The first thing you have to do is open a Spring XD shell. You'll create a new stream but you're going to include the new processor in the definition.

```
xd:>stream create --name ttstest --definition "twitterstream --
track='#fashion' | twitterstreamtransformer | file"
```

Because you named the XML file `twitterstreamtransformer.xml`, that's what Spring XD is expecting as the name of the processor in the stream. After the stream is created, go to your output directory and look at the text file.

```
Sun Dec 29 19:35:54 +0000 2013|Sigue nuestro #blog y entérate de todas
nuestras promociones y novedades http://t.co/b8v01LOHa8 #fashion #style
http://t.co/YZfLj9myBf #moda
Sun Dec 29 19:35:55 +0000 2013|Leonardo DiCaprio Talks
&lt;em&gt;Wolf&lt;/em&gt;'s Craziest Scenes http://t.co/8Si5zL0pMd
#important #Jordan #movie #fashion
Sun Dec 29 19:35:58 +0000 2013|#fashion #design #styling #denim
#sweatpants #hat #flannel #plaid #kibwe #sandal #birkenstock… http://t.
```

```
co/Ij1ZjfM1Gs
Sun Dec 29 19:36:01 +0000 2013|http://t.co/YH4Z25C2YS Outfit of the Day
#ootd #style #fashion #ootn #fbloggers http://t.co/mbviiwDUNz
Sun Dec 29 19:36:03 +0000 2013|Photo: #fashion #design #styling #denim
#sweatpants #hat #flannel #plaid #kibwe #sandal #birkenstock...
http://t.co/STRlQ4Llqw
Sun Dec 29 19:36:03 +0000 2013| @Argento_Fiore Awesome Silver Gemstone
&amp; Shablool Jewelry #Fashion #Silver #Gemstone #Jewelry #Shop #eBay
http://t.co/Ip3w8UQgPT
Sun Dec 29 19:36:06 +0000 2013|Artistic Custom Shoes - Wear Your
Favorite Video Game Icons on Your Feet with These Video Game...
http://t.co/zY4G8iN3qu #fashion #trends
Sun Dec 29 19:36:06 +0000 2013|Fabulous Find! It's on my mind &amp; it's
on @eBay. #Style #Fashion #Deal http://t.co/Cx3Ne4vlm8
Sun Dec 29 19:36:09 +0000 2013|Los mejores #FashionFilm de las marcas de
#Moda http://t.co/9ZIeG9Jcr8 #Fashion #Video
Sun Dec 29 19:36:10 +0000 2013|CONTEMPORARY ART http://t.co/YbhYv1btoo
#luxury #home #decor #trending #style #elegant #interior #design
#fashion #shopping #online #fb #g+
Sun Dec 29 19:36:13 +0000 2013|RT @SUNNIDAYZ: BIG ANNOUNCEMENT Coming
soon from @athompsonii #Music #Sports #Film #Fashion http://t.co/
cI4w8xelO6
```

The processor has achieved its aim by generating output for the tweet data and the text only. This sort of processor is useful for extracting what you need, but in this example you have sacrificed a little control in putting the required fields within the code.

In the next example you start to add some analytics into the output by measuring the sentiment of the tweet text as it comes through and giving each tweet a score.

Real-Time Sentiment Analysis

One of the most common uses of machine learning with volumes of comment data is *sentiment analysis*. Twitter is the perfect platform for this sort of machine learning for the simple reason that the data is generated in volume and, depending on the tracked topics, the velocity can be high.

There are a number of ways to perform sentiment analysis on text, but this second example concentrates on a simple point score on the polarity of the tweet by inspecting each word to see if it's positive or negative.

How the Basic Analysis Works

On its most basic level, sentiment analysis inspects words and, based on a list of positive and negative word lists, determines the score of the incoming string. It's

easy to implement, and for a lot of cases it works well. The longer the sentence the more accurate the sentiment score will be, so for very short tweets it might not give results.

The word sets I'm using for this demonstration are free to use and were originally published by Bing Liu and Minqing Hu as part of their work on sentiment analysis and opinion mining at the University of Illinois. You can download the two data files from `www.cs.uic.edu/~liub/FBS/sentiment-analysis.html`.

Here are two examples of how the sentiment analysis works. Consider the following two sentences:

- "This is the best concert I've been to!"
- "That's bad, really bad, horrible!"

You can easily identify which is a positive statement and which is negative. For our two lexicons, you have to iterate the string and for each word see if it exists in the two lexicons. If it matches in the positive lexicon, then you add a point; likewise, you subtract a point if the word appears in the negative lexicon. After the string has been analyzed, the score is returned back.

You can work out the scores for the two sentences:

- "This is the **best** concert I've been to!" = 1 (best)
- "That's **bad**, really **bad**, **horrible**" = -3 (bad, bad, horrible)

Most of the time this method works out okay, but there are some caveats you should keep in mind. Especially with Twitter data, you can never tell if a user will abbreviate or compact certain words to save space (a tweet is only 140 characters, after all). Also, this sort of analysis does not take into account false interpretations. In some local colloquialisms, it's not uncommon for seemingly negative words to be used in a positive context, and some tweets are sarcastic.

For example, "That new Rihanna album is bad!" Is that a positive review or a negative one? This is where some domain knowledge of the types of subject you are dealing with comes in very helpful; you can adjust the lexicons for context if required.

A NOTE ABOUT TWITTER DATA

Before you start the next project, it's worth having a quick discussion about the nature of the data you're going to use. If you use Twitter regularly, you know that there's no easy route to working on tweets. The data quality is poor—"noisy"—and requires extensive "cleaning" before simple, straightforward analysis can be performed.

The two lexicons provide positive and negative words in lowercase. The Liu and Hu collection includes the most common misspellings of certain words, because misspellings are very common in online media. You might want to add some of your own words to the lexicons for your own purposes.

> For the example given here, I'm removing the hash symbol (#) from tweets, because there are no hashtags in the lexicons, and also removing the @ sign from Twitter usernames. I'm also converting the whole string to lowercase for simpler, direct comparison to the list of words in the lexicons. The alternative would be to transform some alternatives of words to lowercase for comparison to the words in the lexicon. You might want to collapse some variations of a word to a canonical form for sentiment word list comparison but not other variations. I'm not adding any "sanity" filters or other cleanup to the stream.

Creating a Sentiment Processor

Because you have a project with a processor module created, it makes sense to create the new sentiment processor in there. The steps are the same as before; create a new Java class and application context file.

The process flow for this new class is fairly easy to understand. You load the positive and negative word lists and then read the incoming payload in the same way as the first project extracted the created date and the text of the tweet. Then you plug in the new processes, namely cleaning up the tweet text the best you can and calculating the sentiment score. Finally, you return the date, tweet, and score as a string.

Writing the Code

Before you start, make a careful note of where the two lexicon text files are, because you'll be referring to the file paths in the code. If Spring XD can't find the text files while initializing the processor, then all your sentiment scores will be zero. It's worth keeping a close eye on the server log output while testing.

With that in mind, it's time to get started. Create a new package and call it **mlbook.chapter9.processor2** (File ➪ New ➪ Package).

Next, create a new Java class and call it **SentimentTweetScore** (File ➪ New ➪ Class). The code will resemble the same structure as the first processor you worked on, but this time the sentiment score will be a little more involved.

Here's the code:

```java
package mlbook.chapter9.processor2;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
```

```java
import java.util.StringTokenizer;

import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.type.TypeReference;
import org.springframework.integration.transformer.
MessageTransformationException;

public class SentimentScoreTransform {
    private Set<String> poswords = new HashSet<String>();
    private Set<String> negwords = new HashSet<String>();

    public SentimentScoreTransform() {
        loadWords("/usr/local/springxd/pos-words.txt", poswords);
        loadWords("/usr/local/springxd/neg-words.txt", negwords);
    }

    private ObjectMapper mapper = new ObjectMapper();

    public String transform(String payload) {
        try {
            StringBuilder sb = new StringBuilder();

            Map<String, Object> tweet = mapper.readValue(payload,new
TypeReference<Map<String, Object>>() {});
            sb.append(tweet.get("created_at").toString());
            sb.append("|");
            sb.append(tweet.get("text").toString());
            sb.append("|");
            sb.append(scoreTweet(tweet.get("text").toString()));
            return sb.toString();
        } catch (IOException e) {
            throw new MessageTransformationException(
                    "[MLBook] - Cannot work on this tweet: " +
e.getMessage(), e);
        }
    }

    private void loadWords(String filepath, Set<String> set) {
        try {
            BufferedReader in = new BufferedReader(new
FileReader(filepath));
            String str;
            while ((str = in.readLine()) != null) {
                set.add(str);
            }
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private String cleanTweet(String tweet) {
```

```
        tweet = tweet.replaceAll("#", "");
        tweet = tweet.replaceAll("@", "");
        return tweet.toLowerCase();
    }

    private int scoreTweet(String tweet) {
        int score = 0;
        StringTokenizer st = new StringTokenizer(cleanTweet(tweet));
        String thisToken;
        while(st.hasMoreTokens()){
            thisToken = st.nextToken();
            if(poswords.contains(thisToken)) {
                score = score + 1;
            } else if(negwords.contains(thisToken)){
                score = score - 1;
            }
        }
        return score;
    }
}
```

When the class is first requested within a stream definition, Spring XD loads the class and loads the positive and negative word lexicons, storing each in its respective HashSet; this is handled by the `loadWords` method. Once again, you're using the Jackson JSON parser to extract the elements of the tweet you require (the date and the tweet text) but adding another part to the outgoing string: a sentiment score.

The `scoreTweet` method takes the tweet text and passes it to the `cleanTweet` method that removes the # and @ symbols. The `StringTokenizer` class splits up the string by using a space as its delimiter. Iterating over each token, you check against both lexicons and, if a positive word appears, you increment the score by one point; you deduct a point if the word appears in the negative word list.

Finally the score is written to the end of the outgoing string and sent to the stream.

Creating the Application Context

Each processor module requires its own application context, even if the class files required are in the same `jar` file as the XML configuration. Create a new XML file called **sentimenttransformer.xml** (using File ➪ New ➪ File) and use the following XML definition:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/
integration"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:beans="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/integration
    http://www.springframework.org/schema/integration/spring-
integration.xsd">
  <channel id="input"/>

  <transformer input-channel="input" output-channel="output">
    <beans:bean class="mlbook.chapter9.processor2.
SentimentScoreTransform" />
  </transformer>

  <channel id="output"/>
</beans:beans>
```

The XML content is identical to the first processor you wrote except the package and class name have changed.

Packaging, Deploying, and Testing

You have to export the `jar` file again, so it includes the newly created class file for the sentiment scoring (use File ⇨ Export in Eclipse). Copy the `jar` file to the `lib` directory of your Spring XD distribution and also copy the new XML application context file to the module/processor directory.

```
$ cp sentimenttransformer.xml /path/to/springxd/xd/modules/processor
$ cp sentimenttransformer.jar /path/to/springxd/xd/modules/processor
```

For the new files to take effect you have to restart Spring XD. To test the new processor, all you have to do is create a new stream with the `sentimenttrans-former` declaration:

```
xd:>stream create --name sentimenttest --definition "twitterstream
--track='#fashion' | sentimenttransformer | file"
```

Now have a look at the file output (`/tmp/xd/output/sentimenttest.out`) and you see the tweet output and, at the end, the sentiment score based on the two lexicons. Though the output tweets are still the originals, your new class had cleaned up the text before calculating the score.

```
Mon Dec 30 14:29:18 +0000 2013|Miley Cyrus and Her Pink Mohawk Cover
Love http://t.co/Inz6eDxCtG #fashion|1
Mon Dec 30 14:29:26 +0000 2013|RT @PalettePale: In love with my @
Georgeatasda pj bottoms! So cute!  http://t.co/aoIOoJjboH #fbloggers
#blog #fashion #cute #pjs #pyjamas|2
Mon Dec 30 14:29:28 +0000 2013|#Gossip #CelebrityNews Miley Cyrus
Dances, Sings Along at Britney Spears' Piece of Me Concert in Las...
```

```
http://t.co/JmwdvvIsKY #Fashion|-1
Mon Dec 30 14:29:30 +0000 2013|RT @PureLondonShow: Register for FREE now
to Pure 2014 this February and glam up your business with the latest
#fashion designs &amp; ideas htt…|2
Mon Dec 30 14:29:33 +0000 2013|#Snap #Farouushe #Nails #Pink #Fashion
#Love #Girl #French #Lyon #Kiss http://t.co/DzT9HzxfQ8|1
Mon Dec 30 14:29:36 +0000 2013|RT @Fashion_Whipped: Winter fashion items
you will not want to live without! http://t.co/aTE61gRX6h @toryburch @
SINGER22 @shopbop #fashion …|0
Mon Dec 30 14:29:37 +0000 2013|Well, NYE is tomorrow. What are you
wearing? #NYE #fashion|0
Mon Dec 30 14:29:38 +0000 2013|Mi copiloto favorito... Él no me chilla
cuando aparco mal jajaja @GcrGcrshop #moda #bolsos #style #fashion
#musthave http://t.co/tuJVRGaJ3m|0
Mon Dec 30 14:29:39 +0000 2013|Cute. Ootd.  @hapatime #ootd #ootn
#fallfashion #tights #booties #purse #fashion #fashionistalovey…
http://t.co/fjpgoet1lw|0
```

So far, these two projects consumed the entire JSON payload from the tweet and extracted only two elements, namely the create date and the text of the tweet. Perhaps there'll be a time when you want to perform analysis on the rest of the data and extract other elements.

The next section looks at Taps, a feature within Spring XD that enables you to intercept the stream and process the information independent of the original stream definition.

Spring XD Taps

So far with Spring XD, you've concentrated on single-stream definitions with a source, a process transformer, and an output destination. One issue with this single-stream pipeline is that when there are large payloads of information, anything that's not passed along from one stage to the next is lost.

Taps enables you to listen to a stream and perform another function on it in addition to the processor at that level of the stream definition.

For example, suppose you want to preserve the original tweet data as it comes into Spring XD instead of extracting only date and text. Starting with the stream you created that does the sentiment scoring:

```
xd:>stream create --name sentimenttest --definition "twitterstream
--track='#fashion' | sentimenttransformer | file"
```

you can create a new stream but with a tap that points to the stream called sentimenttest.

```
xd:>stream create --name mytap --definition "tap:stream:sentimenttest >
file"
```

In this example, you directed the output to a file called `mytap.out` under `/tmp/xd/output` along with the other files. If you have a look now, you will see that `sentimenttest.out` contains the date, tweet, and sentiment score separated by the pipe character (|), and `mytap.out` has the entire JSON payload that the Twitter streaming API sent you.

Taps are similar to the UNIX pipeline `tee` command that splits (or T's) a stream into two streams or a file and a stream.

Summary

As you've seen throughout this chapter, Spring XD provides a good platform for processing a variety of data streams. The potential of creating your own processing modules means you can start rolling out your own real-time analytics with relative ease.

The sentiment analysis example, although simple, is used commonly in real Twitter sentiment applications, and it's a handy tool to have when monitoring incoming streams of tweet data. The same model could be applied to practically any type of social media status data—even customer reviews on e-commerce platform sites.

It's worth taking some time to become familiar with the APIs before automating tasks with large systems such as Spring XD. Also, as discussed, take into consideration how you are going to store the incoming stream data and its processed output. You might want to go back and re-analyze data and learn more from it.

I used Spring XD as the main platform for this chapter, because it was simple to get usable streams up and running. It's not alone, though, for a platform to consume data. You might want to investigate other systems such as Storm, Apache Flume, or RabbitMQ for constructing real-time systems. Just know that using them might require a little more planning and programming.

Chapter 10 looks at batch processing information with Hadoop, examines how the ecosystem fits together, and shows you how to perform analysis with MapReduce and Pig.

Machine Learning as a Batch Process

This chapter investigates using batch processing to mine and learn from larger amounts of data instead of streaming data. After you've considered the size of data and what you're hoping to learn from it, you then look at various tools to extract, transform, and then process the data for useful results.

This chapter covers using Hadoop, Sqoop, and Pig for large-scale batch processing; these tools enable large data sets to be processed with relative ease. The chapter also discusses more traditional methods of creating programs to run batch processes on data.

Is It Big Data?

Although this book is about machine learning, I can't ignore the term "Big Data" that is increasingly a topic in business today. The phrase is touted as the savior, because it enables companies to see new things in their existing data. The term is broad but ultimately reduces down to the concept of a data set that becomes so large that it is difficult to process with traditional tools.

Depending on whom you ask, you might hear, "It's not Big Data if it's not working on petabytes of data," or "When it becomes too big for a traditional database, then it's Big Data." Both statements are true and valid. Personally, I like the term "data" regardless of whether the amount of data is big or small.

As time marches on, the answer to the "What is Big Data?" question will constantly change. The tools will also adapt, improve, and provide different insight. The key question for many organizations is "What can we do with the data we have?"

Considerations for Batch Processing Data

As covered in Chapter 9's discussion about working with real-time data, you must give consideration to the data on which you want to work. The following sections cover a few key points. I'll refer to the data we used in the last chapter.

Volume and Frequency

The Twitter streaming API generates a lot of JSON data. In Chapter 9 you used only one aspect of it to perform primitive sentiment analysis on the fly. The rest of the data was basically redundant during the real-time process. With batch processing, you can perform more robust and detailed processing without the time-pressure and memory constraints of a real-time system. The trade-off is that you must store the data on secondary storage; hence, you use Spring XD taps to fork the raw data before it is processed and discarded in the stream.

It's important to forecast storage demand against the amount of data you'll bring into the system to be processed. Coupled with the Big Data paradigm of "delete nothing," you should assume that a lot of data will be stored over time, whether you store it in a data warehouse, relational database, NoSQL system, or files with the idea that, at some time, it can be used.

Consider the size of Twitter data, assuming you're tracking specific hashtags:

- 1 tweet = 2 kb
- 25 tweets a minute = 50 kb
- 1 hour = 3,000 kb (3 megabytes)
- 1 day = 72,000 kb (72 megabytes)
- 1 week = 504,000 kb (504 megabytes)
- 1 year = 26,208,000 kb (26 gigabytes)

As rough calculations go that makes sense, but in the real world I would wager that most people would be tracking more than one hashtag for analysis.

Frequency is another issue. For example, I created a stream that just stored the public Twitter stream to a file. Within one minute I'd already stored 8 megabytes (8,000 kilobytes) of data. If I want to store a year of data, I'd have 4 terabytes for that one hashtag. 4 terabytes of raw, uncompressed text for only one hashtag!

Twitter's public streaming API accounts for roughly 10 percent of the full firehose of data, so you can imagine the storage implications if you are to consume

the full firehose. While storage capacity increases, the costs associated are decreasing. This means that creating data warehousing for data is a workable reality for many companies, where originally it was out of reach.

How Much Data?

Data relevance is often more important than the quantity of data in hand. Have you ever wondered why financial websites operate publicly with a 15-minute delay? The sole reason is that the data is for a purely historical record and can't be used for establishing trades; it's too old. There's nothing wrong with using all the available data to process; the question is, will you gain anything from it? Sometimes it's better to look at the last year, the last month, or even the last week. There's nothing stopping you from looking at all three.

Which Process Method?

The current fashion is to unleash Hadoop on data and expect the answers to just jump out at you. I've run training sessions where someone has said, "I thought we put the data in and the answers just came out!" It's important to create a question or a hypothesis to come to some conclusion. For example, you might be looking for customers who've bought widgets A and B to determine what other widgets might be of interest to them.

   As you see in this chapter, it's not all Hadoop; algorithms can be crafted and used in a framework or run on their own. It's a case of planning what you're looking for within the data and defining a process to reach that conclusion.

Practical Examples of Batch Processes

The rest of this chapter is a detailed walkthrough of different approaches to batch data processing. As discussed in Chapter 2, there's no one solution that fits all for this sort of work; solutions can be comprised of different languages, scripts, and commands to get things how you want them. This chapter looks at working with the data you gathered with Spring XD from the previous chapter and also adds some new ingredients to make it work with some batch processing tools. The following sections describe the tools covered in this chapter.

Hadoop

Hadoop is a framework for the processing and storage of large volumes of data either on a single machine (called a single node) or a collection of machines. Written almost completely in Java, Hadoop was conceived in 2005 by Doug Cutting and Mike Cafarella.

The Hadoop system is primarily run on Linux or UNIX-based systems. If you use the Windows operating system, then there are some companies, including Microsoft and Hortonworks, which provide a distribution for Windows and cloud-based Hadoop clusters running Windows.

Instructions on how to install Hadoop are found in Chapter 2. This chapter covers how to get a single-node cluster running with the Spring XD data you collected in the previous chapter.

Sqoop

For data stored within traditional databases, Sqoop (pronounced *scoop*, like a scoop of ice cream) is a useful tool that extracts tables or selected data and outputs it in a form that you can process with Hadoop. Sqoop also creates the required Java class files for the table structures, so implementing them into Hadoop's MapReduce framework is easier. If your database is supported with a Java database (JDBC) driver, then you can implement Sqoop to extract the data from your database and stuff it into Hadoop.

Pig

The Apache Pig project is a high-level scripting language for creating MapReduce programs. It's based on the programming language Pig Latin. On the surface it looks a lot like Structured Query Language (SQL). User-defined functions can also be written in Java, Python, JavaScript, Ruby, Groovy, or other Java Virtual Machine (JVM) languages.

Mahout

The Apache Mahout project is a collection of scalable machine learning algorithms for clustering, collaborative filtering, and classification. Later in this chapter, you build a basket recommendation system that works with the Mahout algorithms that run on Hadoop. Mahout can run independently, but it also implements the MapReduce paradigm, so it works within Hadoop and can perform machine learning over large data sets on large Hadoop clusters.

Cloud-Based Elastic Map Reduce

If you don't have the hardware on hand for running Hadoop at a large scale, it's worth looking at some of the cloud-based offerings. Amazon has Elastic Map Reduce (Amazon EMR) as part of the Amazon Web Services offering. Google's AppEngine also hosts a MapReduce system. Microsoft offers HDInsight on its Windows Azure cloud platform.

Cloud providers that offer these sorts of processing services have their own control panels and processes that are not covered in this chapter. It's worth taking the time to investigate the major available options to see if one of them fits your needs better than buying, maintaining, and servicing lots of hardware yourself.

A Note about the Walkthroughs

This chapter has four complete scenarios that walk you through the process of mining batched data from start to finish.

The first scenario gives you a line-by-line tutorial on how to set up a single-node Hadoop cluster. This includes formatting a Hadoop Distributed File System (HDFS) and running a test job on your batches.

The second scenario takes the work you did in Chapter 9 with Spring XD and extends it further to show you Hadoop and Spring XD working together.

You then move on to recommendations using the machine learning libraries in Mahout and the bulk data collection framework Sqoop.

Last, you look at sales data analysis using Pig scripts and also plain Java code, so you can see how the two compare and can become comfortable with the abstractions Pig provides.

I've divided the tutorials into sections, so you can work through them one at a time, making it easier to leave off and come back and do another one when you feel you can.

Using the Hadoop Framework

One of the misconceptions about Hadoop is that it processes your data and has the answers for you. The notion that you can just throw data at Hadoop and it provides untold stories and new facts is fanciful to say the least. There's still a large amount of planning and preparation you have to do before you can enable Hadoop to do your work properly. Ultimately, you need to know what you're trying to find out; as I've said throughout the book, you need to know the question you are trying to answer.

The misconception aside, Hadoop is very effective at processing huge volumes of data. Even if you are not at the petabytes scale (where Facebook and Google are), you can do a lot with the framework. But don't forget that Hadoop is a means to an end, not an end in itself.

The Hadoop Architecture

Hadoop is built on a number of core components. The Hadoop Common package supplies the file and operating system components, the MapReduce engine

(now two versions: MR1, which is MapReduce, or the newer MR2, which is based on YARN), and the HDFS.

There are various distributions of the Hadoop framework available from Apache, Cloudera, Hortonworks, and MapR. Each vendor has alterations from the core for performance and features, so check them out to find one that suits your needs.

For these exercises I've used the older MR1 MapReduce engine from the Apache Hadoop distribution, so you can see in code how the systems are put together. When you are comfortable with this, then you can easily progress to the MR2, if you want. Existing jobs in MR1 usually work with the MR2 engine without changes.

Hadoop Distributed File System (HDFS)

The HDFS is designed to store a large number of large files. Files smaller than half a gigabyte are stored and managed inefficiently in HDFS. Typical sizes for collections on HDFS are usually terabytes and frequently petabytes. HDFS assumes that it has streaming data access to all other nodes in the cluster, and ideally they are within the same local area network. The philosophy of Hadoop and MapReduce is that it is easier to move the code to the data on a large, distributed system than to send all the data to the code. Hadoop uses a master/slave architecture. MR1 Hadoop implementations have a single node called a NameNode to regulate file access to the slave nodes and also to manage the file system namespace of the cluster. Additionally, there are (many) DataNodes, normally one per machine in the cluster, that manage the disk storage on each machine on which it runs.

References to blocks of data (usually between 128 MB and 512 MB in size) are assigned to DataNodes by the NameNode for processing. A file in the NameNode keeps a transaction log of all changes that happen within the distributed file system. DataNodes pass the data blocks around for replication and storage. The NameNode never passes data blocks.

Because HDFS is written in Java, each component of Hadoop can be run on any platform that has a Java runtime environment. In reality, Hadoop components usually run on UNIX-based machines, although Hortonworks does provide a Windows-based platform, if that's what you prefer.

The NameNode on MR1 engines is a single point of failure. If the NameNode fails, then the whole cluster fails, and data can't be processed. This issue was partially addressed in Hadoop 2.x (using the MR2 MapReduce engine) where federated NameNode service was introduced. Most corporate operations teams have implemented operational workarounds to NameNode's single-point-of-failure problem by having hardware-level and network-level fail-over. They usually use a network file system (NFS) volume for NameNode's data, enabling a "cold spare" machine to take NameNode's role in case of hardware failure.

In addition, the secondary NameNode acts as an inactive watcher of the trans-action logs of the primary NameNode. It periodically obtains a snapshot of the current state of HDFS metadata. If there's a failure of the primary NameNode, then the new NameNode attempts to use the data stored on the secondary NameNode and continues to recover the most recent consistent state of HDFS; then, it runs the outstanding transactions to restore the cluster.

Different File System Types

Although most texts talk about the HDFS, it is worth noting there are some other back-end storage options you can use when running Hadoop:

- Amazon S3 File System
- HTTP/HTTPS (working in read-only mode)
- CloudStore, which is also known as the Kosmosfs and has now evolved into the Quantcast File System
- Network File System (NFS) mounted volumes

In addition, FTP enables you to store data on remote FTP servers.

Setting Up a Single-Node Cluster

Before you can start working through the examples, you need to set up a single-node cluster of your Hadoop system. This is a straightforward exercise on the command line. I go through step by step to get the configuration and secure shell requirements complete. If you haven't read the basic Hadoop installation instructions in Chapter 2, you might want to read them before continuing.

Configuring Secure Shell (ssh)

Hadoop requires access to each node by way of a password-less `ssh` login, regardless if it's a single-node cluster or connecting to a number of machines. To create such a login, become the Hadoop user or log in to the account you will use for Hadoop and type the following command to generate the public and private keys:

```
ssh-keygen –t rsa –P '' –f ~/.ssh/id_rsa
```

The `–t` flag tells `keygen` to use the RSA encryption method, and the `–P` flag is for the password. As you can see, it's two single quotes; make sure there are no spaces between them. If you have previously created a key, you are asked if you want to overwrite it. Within the `.ssh` directory there are two files: the private key `id_rsa` and the public key `id_rsa.pub`.

Next copy the key to the `authorized_keys` file.

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Note the `>>` in the command, which means you are appending to the file. If you use one `>`, then it creates a new file and deletes all the other keys, so be careful when executing it.

Finally try to log in to the same machine on which you created the public/private keys:

```
ssh localhost
```

The first time you use the new key, `ssh` prompts by asking if you trust the key fingerprint. Press Return/Enter to indicate yes. Each time you `ssh` from the localhost to the localhost after that, you will not be prompted for a password; instead, you should immediately see a login banner and then your normal command prompt. Log out again; you're done setting up the keys.

Configuring HDFS

The HDFS requires some configuration changes before you can start processing data. Find the home directory of your Hadoop installation (I'm using `/usr/local/hadoop` as a guide; yours may differ) and go to the `conf` directory.

```
cd /usr/local/hadoop/conf
```

Use a text editor to edit the `core-site.xml` file. (I'm using the nano text editor.)

```
nano core-site.xml
```

In a clean installation, the `core-site.xml` file is basically empty except for the configuration tags. It looks like the following:

```
<configuration>
</configuration>
```

Add the following property for the default file system name:

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

`fs.default.name` is the key name that Hadoop is looking for to set the default HDFS name. The value is set to `localhost` and set to port `9000`.

The final thing to do on HDFS is format the file system. This must be done before you start using Hadoop to analyze the data. The `format` command is done via the `hadoop` command from the command line:

```
hadoop namenode -format
```

You see a few log messages output to the console, but the main thing to look out for is the success message:

```
14/01/14 21:17:01 INFO common.Storage: Storage directory /tmp/hadoop-
jason/dfs/name has been successfully formatted.
```

With the HDFS formatted, you are now ready to start and stop the Hadoop system, run a test job, and retrieve the results.

Running a Test Hadoop Job

So far, you've created a password-less login and formatted HDFS for use. Hadoop comes with some example jobs you can run, and one of those is the Word Count example. In Hadoop terms, the Word Count example has become the "Hello World!" of Big Data. It's become the butt of many jokes for being far too simple, but it's still a handy way to confirm your cluster is working properly.

Within the Hadoop directory there is an `examples jar` file. Make a note of the name, which is usually `hadoop-examples-1.2.1jar`, as you'll need it when doing your test.

First things first—you need to start your Hadoop cluster. The following shell script starts all the required services:

```
/usr/local/hadoop/bin/start-all.sh
```

When the startup process is complete, you can start to copy the required files to process into HDFS and run your Word Count job.

To perform functions on HDFS, you have to do that within Hadoop and not from the usual UNIX commands. However, familiar UNIX-like commands are used within the file system component of the `hadoop` command.

First, create an input directory called `input`:

```
hadoop fs –mkdir input
```

Check that the directory has been created within HDFS by running `–ls`:

```
hadoop fs –ls
```

You should see some output similar to the following:

```
Found 1 items
drwxr-xr-x   - jason supergroup          0 2014-01-14 21:34 /user/jason/
input
```

Now that the directory exists, you can start to copy files to it. I have a public domain text file of the book *Moby Dick*, so I'm going to copy that into HDFS. You can use any text file you want, just remember to alter the filename to the name of your text file.

```
hadoop fs –put mobydick.txt input
```

This command copies the file into the `input` directory ready for Hadoop to process. You're not limited to one file; it's a directory, and you can store as many different files in there as you want. For this exercise, though, just copy the one file.

Now you're ready to run the Word Count job. The basic `hadoop` command is naming the `jar` file to use with the MapReduce code and the input and output directories.

```
hadoop jar /usr/local/hadoop/hadoop-examples-1.2.1.jar wordcount input
output
```

When the Hadoop job runs, there is a lot of output and updates of what MapReduce is working on (you take a proper look at what MapReduce is up to shortly). Although it all looks a bit cryptic, there are some interesting lines of which to take note. First, the following line shows the number of lines of text that the mapper has processed:

```
14/01/14 21:41:36 INFO mapred.JobClient:    Map input records=23244
```

The second useful line to know is the number of output records that the reducer has written in the final phase:

```
Map output records=214112
```

As it stands, the output that Hadoop has created is still sitting within HDFS; and my preference is for it to be a text file in my home directory. With the `hadoop` command, you can merge all the result files within HDFS and create a file on the local file system:

```
hadoop fs --getmerge output output.txt
```

What you're telling Hadoop to do is merge all the output result files to the local file system and call the file `output.txt`. Remember that if `output.txt` already exists, it will be overwritten.

The last thing to do is to shut down the Hadoop server node. You don't want to waste memory while doing simple jobs like this:

```
/usr/local/hadoop/bin/stop-all.sh
```

You see all the services shut down in turn. There's no stipulation to do this, but for the sake of completeness in this walkthrough I've put it in.

Now it's time to inspect the fruits of your labor. The UNIX command `head` shows a certain number of lines of the start of the file instead of listing the entire file:

```
head –20 output.txt
```

The first 20 lines of the MapReduce job output should look like the following:

```
jason@myserver:~$ head -n20 output.txt
"'A    3
"'Also    1
"'Are    1
```

```
"'Aye,     1
"'Aye?     1
"'Best     1
"'Better     1
"'Bout     1
"'But     2
"'Canallers!'    1
"'Canallers,     1
"'Come     1
"'Cross     1
"'Damn     1
"'Down     1
"'Excuse     1
"'Hind     1
"'How?     1
"'I     4
"'Is     2
```

The basic Word Count example that comes with the Hadoop distribution splits a sentence by every space. If there are special characters within text files, such as quotation marks and apostrophes, then they'll be included. This is the reason that 'Aye, and 'Aye? are treated as two separate words during the mapping and reducing phases. I discuss more about these things in the "Mining the Hashtags" section of this chapter.

Quick Summary

Over the last few pages, you've set up the basic Hadoop single-node cluster, including creating a secure login with no password, formatting HDFS, and running a basic test job using the Hadoop examples. Next, you expand on this knowledge and create your own MapReduce job to extract the hashtags from the data collated from Spring XD.

How MapReduce Works

When you hear the name Hadoop, the phrase MapReduce isn't far behind. MapReduce is a programming model, a way of doing things, and it's not a unique feature of Hadoop.

So, how does the Word Count example work? Here's a really brief explanation for those who would like to know but don't want a bunch of jargon.

When the NameNode sends a block of data to be processed, it's sent to the Map phase first. Consider the following piece of text:

Red lorry yellow lorry

Red lorry yellow lorry

Red lorry yellow lorry

The mapper receives that data (let's assume one line at a time), and for each word the mapper assigns the number count of one.

Red 1

Lorry 1

Yellow 1

Lorry 1

Red 1

Lorry 1

…and so on.

In pseudo code it would look like this:

```
function mapper(String text)
    for each word wd in text:
    collect(wd, 1)
```

The data is sent back to the NameNode, and the mapper is free to process another block. When all the blocks are complete, then the reducer can collate the results and add up the occurrences of each word.

```
function reducer(String word, Iterator wordCounts)
    total = 0
        for each wordCount in wordCounts:
        total += wordCount
        return (word, total)
```

The final output will look like:

Red 3

Lorry 6

Yellow 3

It's a simple yet very powerful model, and when it's worked in parallel with many nodes, it enables Hadoop to process huge volumes of data over a number of machines. I avoided getting bogged down with large amounts of theory because it's far more fun coding your own and watching it work. So, it's time to do that and move on to mining the hashtags.

Mining the Hashtags

In the previous chapter, you used Spring XD to consume streaming Twitter data and store it on the file system. Using a custom processor, you extracted the data you wanted (the actual text of the tweet) and used a tap to preserve the full JSON payload that came in.

The one thing you haven't done is explored what was in that data apart from ranking each tweet with a basic sentiment score. With your single-node Hadoop cluster ready for action, you can take things further and make Spring XD and Hadoop integrate to work together.

Hadoop Support in Spring XD

You might have noticed when starting up Spring XD that within the XD configuration output there was an `XD_HADOOP_DISTRO` flag. This flag is telling Spring XD which version of Hadoop to use. At the time of writing, the following distributions are supported:

- **Apache Hadoop 1.2.1 (as default)**: `hadoop12`
- **Apache Hadoop 2.2.0**: `hadoop20`
- **Pivotal HD 1.1**: `phd1`
- **Cloudera CHD 4.3.1**: `cdh4`
- **Hortonworks Data Platform 1.3**: `hdp13`

Because you're working with Hadoop 1.2.1, you don't need to change anything in the configuration. If you were to use a different distribution, you would have to tell the shell which one you were using.

```
./xd-shell --hadoopDistro <your_distribution_number>
```

You're going to use Spring XD to output the processed tweet data directly to HDFS instead of to a file. This removes a lot of file system commands that are required to move the data from the local file system to HDFS. Spring XD takes care of all that for you.

Objectives for This Walkthrough

Remember, you're not dealing with one technology; you're using Hadoop for mining the hashtags based on the output that Spring XD has collated. For the custom MapReduce functions, you're going to use a Java program that uses the Hadoop framework.

After the routine is written, you turn your attention to Spring XD and reconfigure the stream to output to HDFS. Last, you run the Hadoop job to extract the hashtags and tell which is the most popular.

What's a Hashtag?

Say the word "hashtag" to most people and they automatically think of Twitter. However, the hashtag has been around a lot longer than some people imagine.

Hashtags were used to emphasize special meaning within the technology arena. Later, hashtags became widely used in Internet relay chat (IRC) as a means to label specific groups of information.

Hashtags are made up of unspaced words with a hash sign (#) at the beginning. So the hashtag `#MachineLearning` is valid but `#Machine Learning` is not because there is a space between the two words.

So looking at the `#fashion` tweet stream you created in Chapter 9, there are plenty of hashtags to be mined:

```
Sun Jan 19 11:04:11 +0000 2014|Billie Jean - Sporting Club - #top
#talent #pop #show #star #swag #smile #fashion #goodmorning ...
http://t.co/0VCPvcjVBl
Sun Jan 19 11:04:16 +0000 2014|Incredible gift from my grandparents
#katemoss #book #fashion #photography #kate #moss http://t.co/hWBKWxLPzx
Sun Jan 19 11:04:16 +0000 2014|RT @wallpapermag: At @LANVINofficial,
designer Lukas Ossendrijver makes a case for rollneck rebels #fashion
#Lanvin http://t.co/4oaZBZx1YW
Sun Jan 19 11:04:16 +0000 2014|#Gossip #CelebrityNews 2014 SAG Awards:
Cate Blanchett Wins Best Actress, Throws a Little Bit of Shade...
http://t.co/qH9SwUFigT #Fashion
Sun Jan 19 11:04:17 +0000 2014|NEW #NATURAL #BUTT #LIFTER WITH NO
PADDING (X-Large) #shapewear #curves #Fashion #Accessory #Deal http://t.
co/Hk5zFSXjc9
Sun Jan 19 11:04:19 +0000 2014|#shoes #fashion
Dolce &amp; Gabbana Leopard Print Pony Skin Peep Toe Sling Back 40 UK 7
```

All you have to do is create a MapReduce job to extract the hashtags and record each one. This is much like the way the Word Count example worked at the start of the chapter, but you refine the data at the mapping phase to ensure it's cleaned up. Doing so reduces the number of duplicate wordings recorded with punctuation characters.

Creating the MapReduce Classes

Every MapReduce is comprised of a job configuration, mapper class, and, optionally, a reducer. I say optionally for the reducer because, under some circumstances, you don't want to reduce results; instead, you just use the mapper as a means to allocate blocks of data to be processed. This example, though, requires a reducer to collate all the hashtags that were recorded.

Creating the Project

Open your integrated development environment (IDE) and create a new project. (I'm using Eclipse as my IDE.) Call it `HashTagsMapReduce`, as shown in Figure 10-1.
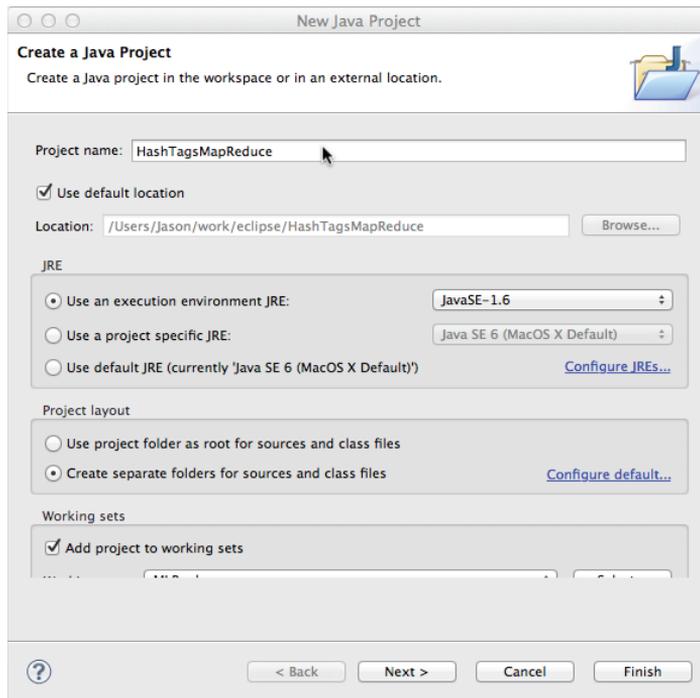
Figure 10-1: Creating a new project

There's no need for any folders to hold any `jar` files. You only use one within the build—the core Hadoop libraries.

Adding the Required Files for the Build

Go to the project properties by right-clicking the project and selecting Properties; alternatively, you can select Project ⇨ Properties. Next, select Java Build Path from the left-hand menu and then click Add External Jar and find the main Hadoop core libraries: `hadoop-core-1.2.1.jar`. (See Figure 10-2.)

Click OK and the project updates with the new libraries registered for use. Now you can start to create some code.

Creating the Mapper

Have a look at the `mapper` class first; there are some considerations you need to make with regard to it. The `mapper` class reads in each line of text and extracts what you're looking for (in this instance a hashtag). The saved data from Spring XD is the tweet date and then the content of the tweet. The data is pipe delimited, so you have to split that first.
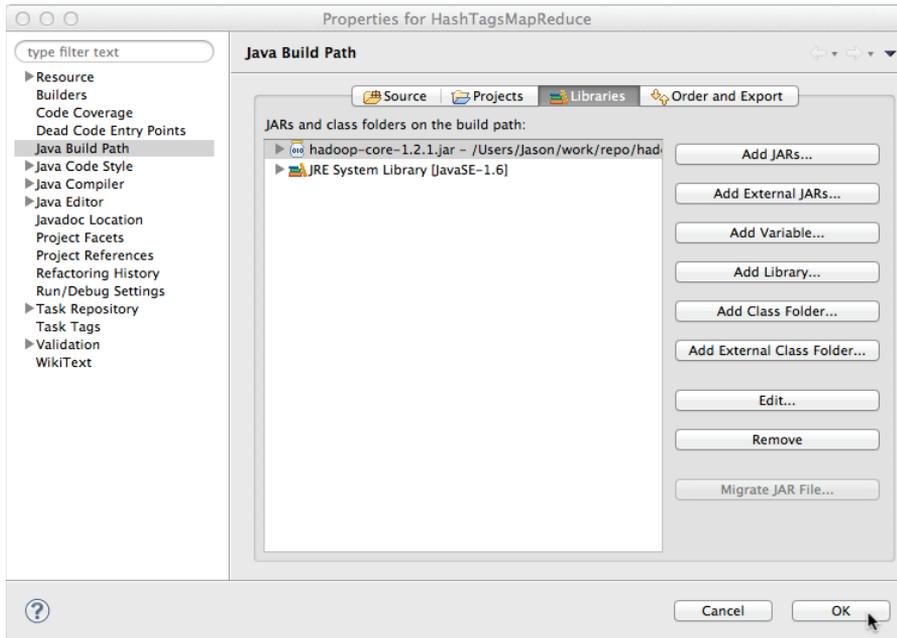
Figure 10-2: Adding the required libraries

Using a perl extended regular expression (peregex) as implemented in the Java language, you can easily match all the hashtags within the tweet. In a Java regular expression, there is a special, single-character matching pattern, \w, that matches a single character that is either a letter, a digit, or the underscore (_) character. (The mnemonic is "word" character.) So to match a hashtag, you can use a simple regular expression:

```
#[\w]+
```

This expression matches a "pound sign" or "hash" (#), and the set that contains a word character repeated one or more times.

Now it's time to put all that into practice in a Java class. Create a new `Java` class (using File ➪ New ➪ Class) and call it `HashtagMapper.java`, as shown in Figure 10-3. I'm using an empty package, but you can use your own naming methods if you want to.
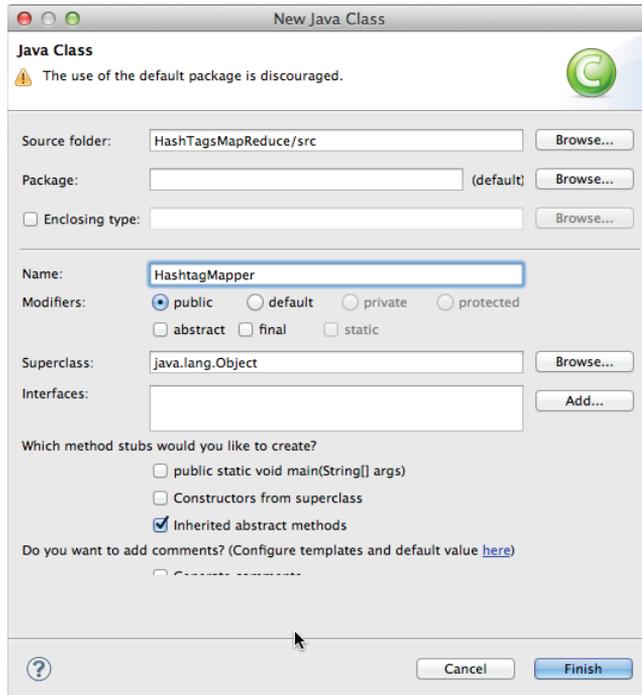
Figure 10-3: Creating the new mapper

Use the following `mapper` code within your class:

```java
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

public class HashtagMapper extends MapReduceBase implements
        Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    @Override
    public void map(LongWritable key, Text value,
            OutputCollector<Text, IntWritable> output, Reporter
reporter)
```

```
        throws IOException {

    String inputLine = value.toString().toLowerCase();
    String[] splitLine = inputLine.split("\\|");
    Pattern pattern = Pattern.compile("#[\\w]+");
    if (splitLine.length > 1) {
        Matcher matcher = pattern.matcher(splitLine[1]);
        while (matcher.find()) {
            word.set(matcher.group());
            output.collect(word, one);
        }
    }
}
}
```

As you can see, it's a simple class. You're going to scale it with Hadoop to cope with large volumes of data. Try walking through it line by line.

When a line of text is mapped, it's passed through the map method. The `Text` value saves that as a Java string and converts it to lowercase. Why lowercase? The mapping phase, for example, sees the words `#Fashion`, `#fashion`, and `#FASHION` as three separate entries and is scored accordingly. By converting them to lowercase, you have one hashtag that would be counted three times. It's cleaner and saves in post processing after the Hadoop job is finished.

Because the incoming data is pipe delimited, I've split that line into a string array. The date is going to be ignored, but you will process the tweet content.

Using Java's regular expression engine, you create a basic pattern to match all the hashtags that appear in the content. For every matched hashtag, the `mapper` registers the matched word and assigns the value 1 to it.

That's the basic `mapper`. As you can see, it's a simple piece of code that consumes the text and matches all the hashtags. The next section examines the `reducer` class.

Creating the Reducer

At present you have a `mapper` that is registering the value of 1 to every matching hashtag from the regular expression. The `reducer` collates all that data and comes up with the final total for each key (hashtag) that it finds.

Create a new class called `HashtagReducer.java` and copy the following code into it:

```
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
```

```java
import org.apache.hadoop.mapred.Reporter;


public class HashtagReducer extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable>{

    @Override
    public void reduce(Text key, Iterator<IntWritable> values,
            OutputCollector<Text, IntWritable> output, Reporter
reporter)
            throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

The `reducer` takes every word that has been recorded by the `mapper` and adds the totals. So, for example, the `mapper` has recorded the following:

```
#fashion,1
#shoes,1
#shoes,1
#fashion,1
#fashion,1
```

The `reducer` combines the results of each key (each hashtag in this case) and adds them, so the final result looks like the following:

```
#fashion,3
#shoes,2
```

You have your `mapper` and a `reducer`. The last thing to do is create a job configuration class so Hadoop can run the job.

Creating the Job Configuration

The job configuration is made up of a number of components that tell Hadoop what `mapper` and `reducer` classes to use and where to expect the input and output files to live.

Create a new `Java` class and call it `HashtagJob.java`. Then use the following code:

```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.mapred.FileInputFormat;
```

```java
import org.apache.hadoop.mapred.FileOutputFormat;

public class HashtagJob {

    public static void main(String[] args) throws Exception {
        JobConf conf = new JobConf(HashtagJob.class);
        conf.setJobName("HashtagMiner");

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(HashtagMapper.class);
        conf.setReducerClass(HashtagReducer.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);
    }
}
```

The output key and values are set (text and an integer). Next, you tell the job configuration which `mapper` and `reducer` classes to use. The input and output file types are defined—both text format for this example—and you specify where the input and output folders on the file system will live.

Notice that the file system folders are read in from the values of the command line, `args[0]` for the input directory and `args[1]` for the output directory.

The final line of the class is Hadoop's `JobClient` running the job configuration. The only thing left to do is export the `jar` file, and then you can do a first test.

Exporting the Jar

The `jar` file is comprised of the `mapper`, the `reducer`, and the job configuration. You can have as many classes in there as you want (if you have multiple jobs and functions), but I find it easier to stick with one specific function (in this case mining hashtags) in one `jar` file.

To export the `jar` file, select File ⇨ Export and select Jar File insert (see Figure 10-4). Ensure that you have the correct project selected and that you've selected the Export All Output Folders option. Call the `jar` file **hashtagmining.jar** and click Finish.

Testing on a File

It's always a good idea to test the new MapReduce algorithm against a small sample of data to ensure that the output is what you're expecting. I know it sounds like common sense, but experience has shown me many things.
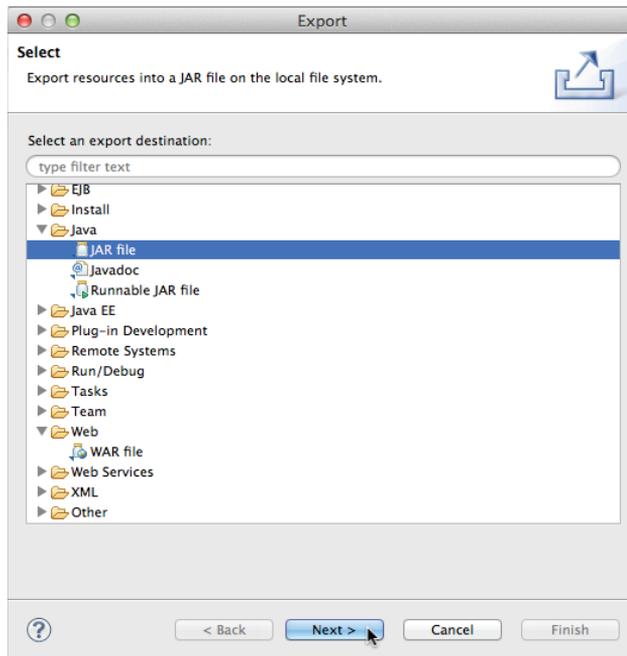
Figure 10-4: Exporting the jar file

First things first, create a directory for the test to happen; I'm going to call mine `hashtagtest`. Within that directory, create another directory called `input`.

```
mkdir hashtagtest
cd hashtagtest
mkdir input
```

I like to keep `jar` files within the testing directory, as it makes things easier when working with the command line. Therefore, I'm going to copy my `jar` file to the `hashtagtest` directory:

```
cp /home/jason/hashtagmining.jar ./
```

Assuming that you have the processed data output from Chapter 9, you can dump a certain number of lines into a new text file to use for the testing:

```
head –n 200 /tmp/xd/output/fashiontweets.out > ./input/input.txt
```

The `head` command should be familiar to you, as you used it earlier in this chapter. This time it's taking the first 200 lines of the processed tweets and directing the output to a new text file called `input.txt` in the `input` directory. Everything's in place now, and you can run the basic test.

Hadoop has three running modes: a single-node cluster, a multi-node cluster, and a local mode. On a development machine it's nice to have a local version of Hadoop on hand, so you can test your code before it's deployed to the main cluster.

The local mode doesn't use HDFS; instead it uses the data that's in the local file system. This is perfect for testing, as it means you're not copying small amounts of data to HDFS for testing. Nothing changes from the command-line perspective, so to test the new `jar` file you run the following command:

```
hadoop jar hashtagmining.jar HashtagJob input output
```

You see the MapReduce job run in exactly the same way as it did when the single-node cluster was tested in the first walkthrough. After it's finished, go to the output directory and you see two files. The first is a file called _SUCCESS to show that Hadoop had completed the job without any problems. The second file is called `part-00000`; this is the text file with the results from the MapReduce test.

Use the `more` command to dump the contents of the file out to the terminal screen, one screen (page) at a time:

```
more  output/part-00000
```

The basic MapReduce job you created doesn't order the results; it just outputs the hashtag and number of times it was found. For a quick look at the most popular hashtags, you can use the `sort` and `head` commands to show the top 20 hashtags.

```
sort -k 2 -n -r output/part-00000 | head -20
```

The `sort` command is using three flags to process the output: The −k 2 flag means that you're sorting on the second key position (the numeric output from the results); -n is to show a numeric sort; and −r is to show the results in reverse order. You see output that's something like the following:

```
Jason-Bells-MacBook-Pro:output Jason$ sort -k 2 -n -r part-00000 | head
-n 20
#fashion    157
#style    19
#etsy    12
#jewelry    12
#skirt    11
#outfit    10
#pencil    8
#vintage    8
#buciki    7
#fashionable    7
#fashionblogger    7
#fashiondiaries    7
#fashionstyle    7
#instafashion    7
#nowe    7
#sale    7
#szpilki    7
#deal    6
#dress    6
#cute    5
```

With the test working as expected, you can turn your attention to Spring XD again. It's time to get Hadoop and Spring XD to talk to each other.

Configuring Spring XD

Until now, the streams that were created directed their output sinks to a file or logged on to the console. Within Spring XD there is an option to output directly to HDFS. You're running Apache Hadoop 1.2.1, so there is nothing to change in the configuration because this is what Spring XD defaults to.

Previously, you've created Twitter streams that were piped through a customer processor to extract the date and content of the tweet. Here's a reminder from the XD shell:

```
xd:>stream create --name fashiontweets --definition "twitterstream
--track='#fashion' | twitterstreamtransformer | file"
```

To configure to use HDFS as the output sink, you need to stop that stream and create a new one. Before you do anything, you first need to tell Spring XD about your Hadoop NameNode:

```
xd:>hadoop config fs --namenode hdfs://localhost:9000
```

Make sure your Hadoop single-node cluster is running, then you can create a new stream.

```
xd:>stream create --name fashiontweets --definition "twitterstream
--track='#fashion' | twitterstreamtransformer | hdfs"
```

From the operating system command line, you can see the output now being stored within HDFS. Using the `hadoop` command you can list the data:

```
hadoop fs --ls /xd/fashiontweets
```

You see the output listed within HDFS:

```
jason@bigdatagames:~$ hadoop fs  --ls /xd/fashiontweets
Found 2 items
-rw-r--r--   3 jason supergroup     22753 2014-01-19 20:07 /xd/
fashiontweets/fashiontweets-0.log
```

If Spring XD is writing to the file while you are inspecting it, then the file has .tmp on the end of it. You're not limited to one file being dumped within HDFS. With the –rollover flag, you can control how big the file can be before a new one is created.

```
xd:>stream create --name fashiontweets –definition  "twitterstream
--track='#fashion' | twitterstreamtransformer | hdfs --rollover=64M"
stream create --name fashiontweets --definition "twitterstream
--track='#fashion' | twitterstreamtransformer | hdfs --rollover=64M"
18:46:26,272  WARN Spring Shell client.RestTemplate:566 - POST request
for "http://localhost:9393/streams" resulted in 500 (Internal Server
```

```
Error); invoking error handler
Command failed org.springframework.xd.rest.client.impl.
SpringXDException: error occurred in message handler [moduleDeployer]
```

This would create a new text file when the current output file is 64 mega-bytes in size. You can still use the `taps` functions to write raw data to the file system as you want, but the data for analysis—the hashtag content—now goes to HDFS. With Spring XD configured you can run your Hadoop MapReduce job properly.

Testing on Streaming Data

With your MapReduce job ready and Spring XD now sending streaming Twitter data directly to HDFS, you can perform some mining of hashtags.

Find the directory where your exported `jar` file is and run the following `hadoop` command:

```
hadoop jar hashtagmining.jar HashtagJob /xd/fashiontweets /fashionoutput
```

Hadoop processes the data within HDFS in the same way it did in the local test. The output is still held within HDFS, so you need to merge it and save it to the local file system.

```
hadoop fs -getmerge /fashionoutput fashionoutput.txt
```

As you inspect the output on the local file system, you can see the most popular hashtags at the moment:

```
jason@bigdatagames:~$ sort -k2nr fashionoutput.txt | head -n20
#fashion    145
#style    35
#deal    31
#love    8
#skirt    8
#jewelry    6
#me    5
#pencil    5
#vintage    5
#cool    4
#dress    4
#earrings    4
#model    4
#outfit    4
#smile    4
#swag    4
#80s    3
#accessories    3
#black    3
#fashionable    3
```

Although you can easily discard the `#fashion` hashtag, as you know that's going to be the most popular, there are some interesting tags of note. It seems that skirts, jewelry, and vintage are all talking points on this day. If you were analyzing this data for an e-commerce site, you could have some routines to respond to the originators of the tweets to show them sites of special offers and so forth.

MapReduce Conclusions

This walkthrough gives you the basic grounding of creating your own MapReduce functions to work with Hadoop. Along with the Spring XD framework, these are the basic components of a system that enable you to monitor Twitter data and get trends on the incoming data. There are other aspects of the data you could mine along with measuring sentiment, finding the most mentioned users, and finding out who's tweeting the most about a specific hashtag. Using the data coming out of Spring XD, with modifications to MapReduce files you have the basis for a good Twitter monitoring system.

Performing ETL on Existing Data

The last exercise was about collating information and processing it in a prompt, distributed manner. The Hadoop routines you wrote could have been run every ten minutes, hour, day, week, and so on. This tackles only the problem of data hosed out of specific application programming interfaces (APIs)—in our case the Twitter streaming API—and dealing with the data in real time or batching up and processing when required.

Companies are sitting on silos of existing data. It might be customer data, sales data, or even point-of-sale data that might be able to give some extra insight to customer behavior patterns. The emphasis isn't on creating new data; it's about making sense with the existing sets.

Historical data might be held in a number of different formats: text, spreadsheet, or relational database. With text, it's a case of copying it to HDFS and then doing the work on it with the required MapReduce classes. When you have data to extract from a database, then things become a little more involved.

The Sqoop project was designed to manage the bulk movement of data from relational databases to Hadoop. Any database server that has a JDBC driver associated with it can be used.

In times gone by, business intelligence specialists would talk about extract, transform, and load (ETL). Sqoop does the same thing for our relational data by extracting it from the database table, transforming it to a comma-separated file and loading it in the Hadoop's HDFS. Additionally, Sqoop also creates a template Java file of the imported data for use within MapReduce, which reduces the development time required to work with the newly acquired data.

Installing Sqoop

To install Sqoop, you need a Hadoop distribution to be available on the same machine, because Sqoop uses Hadoop's MapReduce functions to extract the data. The Hadoop distribution you are using determines which Sqoop version you need to download. Sqoop1 is used for the Hadoop MR1 engine, and Sqoop2 for the MR2 engine.

You can download Sqoop (I'm using 1.4.4 for use with the Hadoop 1.2.1 version) from the Apache download site at `www.apache.org/dyn/closer.cgi/sqoop/1.4.4`.

Extract the tarred file to a directory and export the `PATH` variable to include Sqoop's `bin` directory. You also need to tell Sqoop where your Hadoop installation is. With the following three lines from the command line, everything should be set up and ready to use:

```
export PATH=$PATH:/usr/local/sqoop/bin
export HADOOP_COMMON_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=/usr/local/hadoop
```

You can now run Sqoop to make sure that everything is set up okay. You should see output similar to the following:

```
jason@bigdatagames:/usr/local/hadoop$ sqoop version
Warning: /usr/lib/hbase does not exist! HBase imports will fail.
Please set $HBASE_HOME to the root of your HBase installation.
Warning: /usr/lib/hcatalog does not exist! HCatalog jobs will fail.
Please set $HCAT_HOME to the root of your HCatalog installation.
Warning: $HADOOP_HOME is deprecated.

Sqoop 1.4.4
git commit id 050a2015514533bc25f3134a33401470ee9353ad
Compiled by vasanthkumar on Mon Jul 22 20:01:26 IST 2013
jason@bigdatagames:/usr/local/hadoop$
```

NOTE Don't worry about the warnings, because they're related to HBase and HCatalog, and you're not using those.

Installing the JDBC Driver

Sqoop requires a copy of the JDBC driver to be in the `lib` directory so it can interact with your database. For this walkthrough, I'm using MySQL as the database and the `Connector/J` JDBC driver, which is available from `www.mysql.com/`.

Basic Sqoop Usage

Sqoop is run from the command line, and with a number of flags you can start to extract data. Let's assume you have a MySQL database called "mydb" and

within a table called "mytable," to extract an entire table from the database you would run the command:

```
sqoop import --connect jdbc:mysql://localhost/mydb --username myuser
--table mytable
```

As you can see, Sqoop makes a connection to the database using the standard JDBC connection string like it would if it were connecting within Java code.

Sqoop lets you save the command-line options into a file where each line in the file is a separate command-line option.

```
import
--connect
jdbc:mysql://localhost/mydb
--username
myuser
```

This is saved as a text file, and then it's passed to the Sqoop from the command line with the –options-file flag:

```
sqoop import –options-file /home/jason/options.txt
```

If you are repeatedly using Sqoop commands with a set of common options, then it makes sense to keep all the options in a text file. You can use your favorite text editor instead of the command-line editor to make tweaks and changes. If you apply this technique, consider adding a shell function or alias in your shell's startup configuration (for example, ~/.bashrc).

```
.alias sq='sqoop import –options-file /home/Jason/options.txt'
```

Handling Database Passwords

The majority of the time the database tables are protected. There are two methods to use a password with Sqoop. The first method is just to include the password in plain text in the command line:

```
sqoop import –connect jdbc:mysql://localhost/mydb –username myuser –
password mypass –table mytable
```

As you can imagine, this is not an ideal solution, as you are showing the plain password to anyone who happens to be passing your terminal, anyone who happens to look at what is running on the system, or even someone who looks back at what was run on the system in the past. It's a very big security risk.

The more sensible and preferred approach is to hide the password in a file and reference the file with Sqoop:

```
echo [your password] > ~/.password
chmod 400 .password
```

Now when you run Sqoop you can add the password file to the command-line arguments:

```
sqoop import --connect jdbc:mysql://localhost/mydb --username myuser
--password-file /home/Jason/.password –table mytable
```

The job configuration now has no direct access to the password. It also means that if another user peeks into the process table (for example, runs the command to show running processes [ps] in UNIX) that person does not see the password referenced in the process list.

Product Recommendation with Mahout

To show you Sqoop in action, you're going to create an e-commerce recommendation system. You'll use Hadoop and the Mahout machine learning libraries as well.

The source code and the database import files are on the Github repository that accompanies this book. You can download the full code from `http://www.wiley.com/go/machinelearning`.

> **NOTE** There are many e-commerce websites available today, and they sell pretty much everything. When the topic of product recommendation comes up, the main business many people talk about is Amazon.com, because that company really got to grips with mining the data early on. Making recommendations based on what shoppers have previously purchased and just viewed is a powerful seller's tool in the right hands. Harnessing this data is now critical to the success of all retail merchants. The ability to push a product in front of customers with a degree of targeting increases the top line (sales) of any company.

So What's in a Recommendation?

Consider what a recommendation is for a second. Imagine an online store with many product items and many customers. Now offer each customer the honor of rating an item from 1 (worst) to 5 (best). The database table is very simple.

- User ID
- Stock ID
- A rating
- A timestamp

Over time, you collect many user ratings in that table and periodically mine them to give users recommendations on products rated by other users. It doesn't matter what type of items or the background of the users (age range, income bracket, and so on); all that matters is who rated what.

Using Mahout's similarity co-occurrence algorithm, similar items that users have rated can be recommended to users who viewed or purchased the other items.

Bill gives King Crimson's album Discipline five stars.

Sid gives King Crimson's album Discipline five stars.

Sid gives King Crimson's album Beat five stars.

Then we can recommend *Beat* to Bill, as both Sid and Bill gave *Discipline* a good rating. Now, at scale you might have thousands of albums for sale, tens of thousands of customers, and millions of ratings. The more data you have, the better the mining and recommending that can be done. Too few ratings, and you won't have enough data to do decent calculations.

I know what you're thinking: "This sounds like a lot of programming."

The reality is that all this correlation is already in place with Mahout's API, so the only real programming involved it is to interpret the output of the recommendations for each user.

Although I'm using Java, it would be easy to put this together in PHP, Python, or Ruby.

Setting Up the Database

I'm going to use MySQL as my database of choice. From the command line, I'm going to create the database:

```
mysqladmin –u root create mahoutratings
```

The next thing to do is create two tables: one for the stock items and another for the user ratings. The easiest way is to copy the following SQL code and save it to a text file called stockitems.sql:

```
create table stockitems (
    id int(11) not null primary key auto_increment,
    albumartist varchar(200) not null default "",
    albumname varchar(200) not null default ""
);
```

Next, create a text file for the user ratings table and call it userratings.sql:

```
create table userratings (
    userid int(11) not null default -1,
    stockid int(11) not null default -1,
    rating int(4) not null default 3,
    created_at int(11) not null default -1
);
```

With these two text files, you can now create the tables in MySQL from the command line:

```
mysql –u root mahoutratings < stockitems.sql
mysql –u root mahoutratings < userratings.sql
```

That's the bare database. Now, you need to add some data before you do any analysis on it.

Importing the Initial Data

I've created the basic data for you; this saves writing programs to create random data and so on. From the command line, you run the two commands into MySQL:

```
mysql –u root mahoutratings < stockdata.sql
mysql –u root mahoutratings < ratingsdata.sql
```

To make sure that all went well with the import, use the MySQL client to confirm:

```
mysql –u root mahoutratings
```

Then, from the MySQL client command line, type the following commands:

```
mysql>select count(*) from stockitems;
mysql>select count(*) from userratings;
```

If the tables are empty, then make sure that you typed the import commands correctly and try again.

Installing the Mahout Libraries

The Mahout libraries are extra to the Hadoop distribution, so they need to be downloaded and installed before you start. You can download the Mahout binary package from `www.apache.org/dyn/closer.cgi/mahout/`.

To install it, either untar or unzip the file to a directory. Keep a note of the directory because you'll need it for finding the core `jar` files. The routines you're going to use are working in conjunction with Hadoop, so please make sure you have HDFS working and the Hadoop processes running before you start mining the data. If you need a reminder on how to set up your single-node Hadoop cluster then look at the "Setting Up a Single Node Cluster" section earlier in this chapter.

Extracting Data with Sqoop

The plan of execution is simple. Extract the data from the database and then copy to HDFS. Have Mahout run its recommender job on the data. This generates the results with recommendations for each user.

Last, you create a small Java program to read the results and show the actual stock items recommended for a specific user ID.

From Sqoop, you can pull the entire contents of the `userratings` table out in one line:

```
sqoop --connect jdbc:mysql://localhost/mahoutratings --username root --
table userratings
```

Sqoop extracts the table contents and writes them as a CSV file. Now, you need to copy that into HDFS so Hadoop and Mahout can do their work on it. It's also worthwhile to rename the output file that Sqoop created so you know exactly what it is:

```
mv part-00000 userratings.csv
```

Then copy that to HDFS:

```
hadoop fs –put userratings.csv userratings.csv
```

With the data in HDFS, you can perform the Hadoop job on the data and let it generate the recommendations.

Running Hadoop

When you installed the Mahout libraries, I said to keep a note of the installation directory. Now, you're going to use it with the Hadoop command to do the recommendation processing.

To set everything going, run the following command (all in one line):

```
hadoop jar [path to mahout]/ mahout-core-0.8-job.jar org.apache.mahout.
cf.taste.hadoop.item.RecommenderJob –s SIMILARITY_COOCCURRENCE –input
userratings.csv --output recommendationoutput
```

Mahout runs a series of MapReduce jobs to complete the recommendations. It can take some time to complete. Obviously this is not something you'd really want to do in real time, but it's a perfect use of batch processing.

Inspecting the Results

When the job is complete, you can get the results from HDFS and inspect them:

```
hadoop fs –getmerge recommendationoutput recommendations.txt
```

Looking at `recommendations.txt` you see the customer ID and then a set of numbers. These numbers are the stock item IDs that Mahout has recommended for that customer based on the other user ratings made.

To make sense of the results, I've quickly coded up a Java program that reads in a customer ID and extracts the recommendations:

```java
package chapter10;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.HashMap;
import java.util.Map;

public class ShowRecommendations {
    static {
        try {
            Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    private Map<Integer, String> recommendations = new HashMap<Integer,
String>();

    public ShowRecommendations(int customerid) {
        initMap(); // load the recommendations in to the hash map
        try {
            String rec = recommendations.get(new Integer(customerid));
            String output = generateRecommendation(rec);
            System.out.println("Customer: " + customerid + "\n" +
output);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private String generateRecommendation(String input) throws
SQLException {
        StringBuilder sb = new StringBuilder();
        System.out.println("Working on " + input);
        String tempstring = input.substring(1,input.length()-1);
        String[] products = tempstring.split(",");
        System.out.println("products = " + products.length);
        Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost/mahoutratings", "root", "");
        PreparedStatement pstmt = con
                .prepareStatement("SELECT albumname, albumartist FROM
stockitems WHERE id=?");
        ResultSet rs = null;
        for (int i = 0; i < products.length; i++) {
            String[] itemSplit = products[i].split(":");
```

```java
                pstmt.setInt(1, Integer.parseInt(itemSplit[0]));
                rs = pstmt.executeQuery();
                if (rs.next()) {
                    sb.append("Album: " + rs.getString("albumname") + " by "
                            + rs.getString("albumartist") + " rating: "
                            + itemSplit[1] + "\n");
                }
            }

            rs.close();
            pstmt.close();
            con.close();

            return sb.toString();
        }

        private void initMap() {
            try {
                BufferedReader in = new BufferedReader(new FileReader(
                        "/Users/Jason/mahouttest.txt"));
                String str;
                while ((str = in.readLine()) != null) {
                    String[] split = str.split("[ ]+");
                    System.out.println("adding: " + split[0] + "=" +
split[1]);
                    recommendations.put(Integer.parseInt(split[0]),
split[1]);
                }
                in.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public static void main(String[] args) {
            if (args.length < 1) {
                System.out.println("Usage: ShowRecommendations
[customerid]");
            } else {
                ShowRecommendations sr = new ShowRecommendations(Integer.
parseInt(args[0]));
            }
        }

}
```

A quick test on customer ID 9 returns the following output:

```
Customer: 9
Album: AlbumName_739 by AlbumArtist_739 rating: 5.0
Album: AlbumName_550 by AlbumArtist_550 rating: 5.0
Album: AlbumName_546 by AlbumArtist_546 rating: 5.0
```

```
Album: AlbumName_11 by AlbumArtist_11 rating: 5.0
Album: AlbumName_527 by AlbumArtist_527 rating: 5.0
Album: AlbumName_523 by AlbumArtist_523 rating: 5.0
Album: AlbumName_514 by AlbumArtist_514 rating: 5.0
Album: AlbumName_511 by AlbumArtist_511 rating: 5.0
Album: AlbumName_508 by AlbumArtist_508 rating: 5.0
Album: AlbumName_498 by AlbumArtist_498 rating: 5.0
```

The code could easily be modified to store the recommendations back to the database table, so it could be easily used in the e-commerce code for a website.

Extending This Project Further

There's plenty of scope for this project to be used in a real-world situation. The more data you generate, the better the recommendations you can give. Although you could perform the Mahout functions via code within a program, I prefer to use them away from the web application. This choice is more about personal preference than anything else, but I have some good reasons. I don't want the database or the web application to be slowed down by heavy memory processing. As you've probably gathered, I prefer a more batch-oriented approach by extracting the data from the database, running the recommendation algorithm, and then sending it back to the database for the web application to access.

To run this routine periodically, set up a UNIX `cron` job (setting up a `cron` job is covered in the section "Scheduling Batch Jobs" at the end of this chapter). Depending on the volume of ratings you get, this could be run once an hour, twice a day, or once a day. There's nothing wrong with trying a few variations of time difference to find out which works best for you.

Mahout Conclusions

The Mahout tutorial gave you a robust recommendation engine without the need for programming. Using Sqoop to extract data from a database and then using Hadoop to perform the Mahout processing in parallel gave you good recommendations on a large amount of data in a short space of time. The Mahout libraries give a lot of options of memory caching to improve processing performance, but it's still worth taking the processing away from the main application and performing it in batches.

Mining Sales Data

For all the Big Data problems in the world, there's a large lean toward the social media side of things. It's mostly text and can be mined with relative ease. Back in the real world where commerce is based on numbers, hashtag counts might

not be the most important thing on a CEO's mind. Finding out which customers are worth retaining, however, probably is.

This walkthrough looks at developing some code to perform calculations and then moves the code into Hadoop. Finally, you move the data through Pig, which is a MapReduce scripting-like language. You can download the code and the data from `http://www.wiley.com/go/machinelearning`.

Welcome to My Coffee Shop!

To be fair, it's an odd coffee shop because it sells only one sort of coffee: lattes. It's been open for just more than a year, and I have a good set of data for my 20,000 customers (it's a large coffee shop). Okay, I don't have a coffee shop, but you can use your imagination for this exercise.

The data is made up of a customer ID and the sales for 13 months. Here's what the data looks like for the first ten customers:

```
1,3,11,6,10,7,10,12,9,7,6,10,14,5
2,1,13,10,0,5,12,1,13,1,3,7,5,14
3,14,2,3,8,10,12,13,6,13,2,1,7,2
4,14,1,11,14,11,1,12,10,0,1,14,5,9
5,1,3,11,14,3,8,10,9,7,3,0,5,6
6,3,3,1,14,5,0,4,8,9,11,8,0,5
7,8,1,11,8,13,2,13,4,6,2,7,14,14
8,13,6,8,12,8,10,14,0,13,6,14,2,9
9,11,1,13,0,11,13,6,3,10,8,8,5,1
10,1,10,2,0,5,0,4,12,12,10,6,6,9
```

It's a standard CSV file full of integers. I've been neat in my data collection over time. What I want to do now is answer some questions that have been on my mind.

- How many lattes do my customers purchase?
- What is the sales drop for each customer?
- What is the duration of the sales drop?

Armed with these stakeholder questions, you can put some actions into place. Fortunately, these questions are simple to answer with a little bit of simple math:

- How many lattes do my customers purchase?

  Calculate the mean sales on months 1-12.

- What is the sales drop for each customer?

  Subtract the month 13 sales from the mean.

- What is the duration of the sales drop?

  Calculate the number of months in which sales were below 40 percent of the mean.

If you had a handful of customers, you could do this with a calculator or spreadsheet. Even with 20,000 customers, a spreadsheet would do the job fine. If my coffee shop goes worldwide and scales up to 20 million customers, then I'd have a job on my hands, and that's where these tools become useful.

Going Small Scale

You need to start small and develop some code that answers the questions posed without processing all the data. Create a small training and testing data set using the UNIX `head` command:

```
head –n20 salesdata.csv > training.csv
```

With small sets of data, you can write the core of your logic, test it, and then port it to a Hadoop MapReduce job later.

Writing the Core Methods

In the stakeholder list of requirements, there are three methods you need to code up. Add a file reader to pull the CSV file in and loop through each customer record:

```java
package chapter10;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class CoreMethods {

    public CoreMethods() {
        try {
            BufferedReader in = new BufferedReader(new FileReader(
                    tmpdata.csv"));
            String str;
            while ((str = in.readLine()) != null) {
                String[] split = str.split(",");
                double[] values = new double[12];
                // 0 = userid
                // 1 - 13 = months sales. Jan - Jan, Mar - Mar etc
                for (int i = 1; i <= 12; i++) {
                    values[i-1] = Double.parseDouble(split[i]);
                }

                double mean = getMean(values);
                System.out.println("User id: " + split[0]);
                System.out.println("\tMean: " + mean);
                System.out.println("\tMonth 13 Sales Drop = " +
```

```
calcSalesDrop(Double.parseDouble(split[13]), mean));
                System.out.println("\tMonths 40% below mean: " +
monthsBelow(values, mean));
            }
            in.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private int calcSalesDrop(double lastMonth, double mean) {
        return (int)(lastMonth - mean) < 0 ? 0 : (int)(lastMonth -
mean);
    }

    private int monthsBelow(double[] data, double mean) {
        int count = 0;
        for(double a : data) {
            if((a < (mean * 0.40))) count++;
        }
        return count;
    }

    private double getMean(double[] data) {
        double sum = 0.0;
        for(double a : data) {
            sum += a;
        }
        return sum/data.length;
    }

    public static void main(String[] args) {
        CoreMethods coreMethods = new CoreMethods();
    }
}
```

As you can see, there are three private methods. The getMean() method takes the array of the sales data from month 1 to 12. The calcSalesDrop() method takes the value of the last month of sales (month 13) and subtracts the mean. If it's a negative number, then it just returns zero. Last, the monthsBelow() method iterates through each of the 12 months, and anything below 40 percent of the mean is added to the count variable.

Programmatically the methods are simple math functions, but they serve their purposes perfectly.

Running the program with the test data, you get the following output:

```
User id: 1
    Mean: 8.75
    Month 13 Sales Drop = 0
    Months 40% below mean: 1
```

```
User id: 2
    Mean: 5.916666666666667
    Month 13 Sales Drop = 8
    Months 40% below mean: 4
User id: 3
    Mean: 7.583333333333333
    Month 13 Sales Drop = 0
    Months 40% below mean: 4
User id: 4
    Mean: 7.833333333333333
    Month 13 Sales Drop = 1
    Months 40% below mean: 4
User id: 5
    Mean: 6.166666666666667
    Month 13 Sales Drop = 0
    Months 40% below mean: 2
User id: 6
    Mean: 5.5
    Month 13 Sales Drop = 0
    Months 40% below mean: 3
```

If this were a small coffee shop with a few hundred customers, then that would do fine. No need to complicate matters with Hadoop clusters or anything like that—it's computing power you just don't need. On the chance that the coffee shop acquires a global chain of shops and the customer count increases to millions of customers, then you have to start looking at alternatives.

The next section looks at scaling it up using the Hadoop framework.

Using Hadoop and MapReduce

You can reuse the core methods you created in the last program within the Hadoop framework. In terms of modification, it's a case of adding those three methods to within the map phase of the MapReduce job. The main difference between this code and the MapReduce job you created for the hashtag mining is that you don't require a reducer.

In this instance, the mapper is going to work through each line of the sales data and output the results. The job configuration does not have any reducers set, so the output is the same number of records that goes in.

This time around, you add the job configuration in the same Java class as the mapper, just to keep everything together.

```java
package chapter10;

import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
```

```java
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;

public class CoreMethodsMapper {

    public static class Map extends MapReduceBase implements
            Mapper<LongWritable, Text, Text, Text> {
        private Text userid = new Text();
        private Text userinfo = new Text();

        public void map(LongWritable key, Text value,
                OutputCollector<Text, Text> output, Reporter reporter)
                throws IOException {
            String[] split = value.toString().split(",");
            double[] datavalues = new double[12];

            for (int i = 1; i <= 12; i++) {
                datavalues[i - 1] = Double.parseDouble(split[i]);
            }

            double mean = getMean(datavalues);

            StringBuilder sb = new StringBuilder()
                    .append(mean + "\t")
                    .append(calcSalesDrop(Double.parseDouble(split[13]),
mean)
                        + "\t").append(monthsBelow(datavalues,
mean));

            userid.set(split[0]); // define our output key, the user id
            userinfo.set(sb.toString()); // define the output data

            output.collect(userid, userinfo);
        }

        private int calcSalesDrop(double lastMonth, double mean) {
            return (int) (lastMonth - mean) < 0 ? 0 : (int) (lastMonth -
mean);
        }

        private int monthsBelow(double[] data, double mean) {
            int count = 0;
```

```
            for (double a : data) {
                if ((a < (mean * 0.40)))
                    count++;
            }
            return count;
        }

        private double getMean(double[] data) {
            double sum = 0.0;
            for (double a : data) {
                sum += a;
            }
            return sum / data.length;
        }

    }

    public static void main(String args[]) throws IOException {
        JobConf conf = new JobConf(CoreMethodsMapper.class);
        conf.setJobName("CoreMethods Sales Data");
        conf.setNumReduceTasks(0); // no reducers!

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(Text.class);

        conf.setMapperClass(Map.class); // the map class within this
code

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);

    }
}
```

The main method contains the job configuration; notice that there's a method called `setNumReduceTasks()` and it's set to zero. You're telling Hadoop not to perform any reducers and just write out what the mapper is sending.

Copy the entire 20,000-record sales data set into the `input` directory.

To run the job, you have to export the `jar` file with the class file inside and use Hadoop to run it. (If you need a reminder on how to create the `jar` files and run them with Hadoop, there are examples earlier in this chapter.)

```
hadoop jar coremethodstest.jar chapter10.CoreMethodsMapper input output
```

The output is sent to the output folder, and within the part-00000 file you see all the records in order with their mean, sales drop, and months below average:

```
head -n 20 output/part-00000
1    8.75    0    1
2    5.916666666666667    8    4
3    7.583333333333333    0    4
4    7.833333333333333    1    4
5    6.166666666666667    0    2
6    5.5    0    3
7    7.416666666666667    6    3
8    8.833333333333334    0    2
9    7.416666666666667    0    2
10    5.666666666666667    3    4
11    7.75    0    3
12    6.75    6    2
13    7.333333333333333    0    1
14    8.666666666666666    0    1
15    7.833333333333333    0    3
16    5.583333333333333    0    1
17    8.0    0    2
18    7.0    0    2
19    7.333333333333333    0    1
20    8.416666666666666    0    1
```

The output is tab delimited, so you can easily import it into a database or a spreadsheet if you need to.

Hadoop can be used for many applications, and, although the main emphasis is on the MapReduce programming model, sometimes there's no need for the reducer. The mapper acts as a very handy parallel work distributor.

One of the issues with the Hadoop framework is that there's programming knowledge required to construct the mapper and reducer code. So, what happens if there aren't any programmers in your organization that would perhaps struggle with crafting a MapReduce program? Then you need to think about a way of scripting a MapReduce job in a more abstract way that does not require deep programming skill.

Using Pig to Mine Sales Data

Coding MapReduce jobs can be a bit of a chore, as there's a lot of coding that goes into a basic MapReduce job. It is worth noting that you're not limited to using Java to create the jobs; with the Hadoop streaming facility you can use languages such as Ruby, Python, and R for processing the data. All those languages require good scripters or programmers to code solutions. With Pig, though, you can produce MapReduce models with a basic scripting language that looks very much like SQL.

Why the name Pig? It's derived from the Pig Latin language. Pig abstracts the MapReduce model and converts it into a higher-level language that's a lot easier to understand. I must stress, though, that Pig is not SQL; it just looks a bit like it.

Installing Pig

The Pig downloads are available from `www.apache.org/dyn/closer.cgi/pig`. After you have downloaded a release, you need to unzip and then untar it to your chosen directory.

For these examples, I use Pig in local mode (on a single machine with no parallel processing) and later use the batch mode where I save Pig scripts in their own files. You can use Pig in MapReduce mode and that uses the Hadoop cluster to connect and access the HDFS file system.

Starting the Pig Console

To start the console in local mode, you run the following from the command line:

```
pig -x local
```

After it's loaded, you see Pig's command prompt, `grunt>`, and you're ready to get started.

Before you can do anything, you need to load some data. Use the sales data that you've previously been working on.

Loading Data into Pig

The `LOAD` command reads in the data. In its most basic form it is a very greedy command:

```
grunt> SD = LOAD 'salesdata.csv ' USING PigStorage(' ,');
```

This loads the sales data into a Pig variable called `SD` (that's the name I gave it and not a Pig-defined one). The `PigStorage` function is a storage type that Pig uses as a loading and storing utility. It acts as a parser to the input records. We're supplying the `PigStorage` with the comma to parse on.

To make sure the data has loaded into Pig, you can run the `DUMP` command to see what the input looks like.

```
grunt> DUMP SD;
```

You see the output sent to the console; here are the last few lines of the sales data:

```
(19985,3,5,1,13,13,8,6,14,4,9,7,2,6)
(19986,2,9,3,13,5,1,12,7,12,0,12,7,12)
```

```
(19987,7,11,11,6,13,3,2,5,6,11,4,7,9)
(19988,6,4,2,7,13,3,7,11,13,2,11,12,12)
(19989,12,0,3,0,14,7,0,5,6,8,9,1,5)
(19990,6,2,10,9,1,12,9,8,10,2,2,8,8)
(19991,2,11,14,4,14,10,10,8,13,8,1,9,1)
(19992,9,12,1,14,12,3,0,7,12,2,0,5,13)
(19993,1,2,13,8,6,9,10,14,9,2,2,14,2)
(19994,9,4,4,2,11,9,10,4,2,3,13,9,0)
(19995,5,2,1,1,14,4,9,12,4,13,6,7,14)
(19996,3,6,6,13,8,1,14,10,5,0,7,9,14)
(19997,4,8,13,3,14,11,4,2,11,12,13,8,1)
(19998,8,10,5,8,12,9,7,8,14,8,14,7,12)
(19999,10,13,14,4,10,14,2,9,11,14,9,14,8)
(20000,2,4,7,2,9,10,8,13,7,1,5,9,3)
```

Notice how you've not told Pig what the schema is; it's just assuming that there's something there, regardless of whether it's numbers or text. Pig is very greedy on these terms; it guesses its way around and basically consumes any data that's loaded in. To define what data the `PigStorage` is to expect, you can define a schema:

```
grunt> SD = LOAD 'salesdata.csv' USING PigStorage(',') AS (custid:int,
m1:int, m2:int, m3:int, m4:int, m5:int, m6:int, m7:int, m8:int, m9:int,
m10:int, m11:int, m12:int, m13:int);
```

All the values are integers and the schema is basic; it's a customer ID and then the values for the 13 months of sales. There are identifiers for the values too: `custid`, `m1`, `m2`, and so on. If relation names are not defined then you can reference by position using the dollar sign: $0, $1, $2, and so on.

When a schema is presented in the LOAD statement, then Pig tries to conform the best it can. If it comes across data that doesn't match the schema, it either returns a null value or throws an error.

Data Types in Pig

For now you're using integer data types, but the other data types that Pig supports are listed in Tables 10-1 and 10-2 for reference. Further in the walkthrough, I bring in more data types when I cover joining tables of data together.

If a schema is not present, then Pig defaults to the type `bytearray` and attempts to convert the data, depending on the context in which you are using that data.

Table 10-1: Simple Data Types

DATA TYPE	DESCRIPTION
chararray	A character array string formatted in Unicode UTF-8
bytearray	Byte array (blob)
int	Signed 32-bit integer
long	Signed 64-bit integer
float	32-bit floating point
double	64-bit floating point

Table 10-2: Complex Data Types

DATA TYPE	DESCRIPTION
tuple	An ordered set of fields.
	Example: (1,2,8,2,3,0)
bag	A collection of tuples.
	Example: {(1,2),(4,2),(9,8)}
map	A set of key value pairs.
	Example: [mykey#myvalue]

Running the Sales Data as a Pig Script

With a bit of Pig theory under your belt, you can concentrate on the sales data again. Up to now you've done the major processing of any data in Java either as a plain program or using the Hadoop framework to run MapReduce on larger volumes. Quickly review what you need to do with the data:

- How many lattes do my customers purchase?

  Calculate the mean sales on months 1-12.

- What is the sales drop for each customer?

  Subtract the month 13 sales from the mean.

- What is the duration of the sales drop?

  Calculate the number of months in which sales were below 40 percent of the mean.

It's easier to create scripts in a text editor and run from the command line. If you're still within the Pig shell, then exit using the quit command and you return to the command prompt.

Open your text editor of choice. Call the file salesdata.pig; it will contain your script. The first two items on the list are straightforward and the last requires some thought.

```
SD = LOAD 'salesdata.csv' USING PigStorage(',') AS (custid:int, m1:int,
m2:int, m3:int, m4:int, m5:int, m6:int, m7:int, m8:int, m9:int, m10:int,
m11:int, m12:int, m13:int);

AVGDATA =   FOREACH SD GENERATE custid,
            AVG({m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12}),
            AVG({m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12})-m13;

DUMP AVGDATA;
```

The script is basic, but as a first pass it already fulfills the first type requirements. On the first line, you import the data using a set schema with the customer ID and then the months. The imported data is saved in a variable called SD.

You create a new variable called AVGDATA by iterating through each line of SD and storing the customer ID and the average of months 1 through 12 sales, and then calculating the sales drop. Last, you output the new calculations to the console.

Save the script in your text editor and exit. Make sure that the data is in the same directory as the Pig script and run the following command:

```
pig -x local salesdata.pig
```

You see Pig start processing the script and the data. If all is well, you see all 20,000 records output to the console with the customer ID, average sales, and the sales drop.

```
(19989,5.416666666666667,0.41666666666666696)
(19990,6.583333333333333,-1.416666666666667)
(19991,8.666666666666666,7.666666666666666)
(19992,6.416666666666667,-6.583333333333333)
(19993,7.5,5.5)
(19994,6.666666666666667,6.666666666666667)
(19995,6.5,-7.5)
(19996,6.833333333333333,-7.166666666666667)
(19997,8.583333333333334,7.583333333333334)
(19998,9.166666666666666,-2.833333333333334)
(19999,10.333333333333334,2.333333333333334)
(20000,6.416666666666667,3.416666666666667)
```

The data is stored as tuples. You can see the parenthesis in the output data. Note, this is just output to the console and hasn't been stored. I cover that in a moment. There's still one requirement left to fix.

I'm creating a custom function to calculate the months below the average sales. Pig lets you create user-defined functions (UDF) in Java (and other scripting languages), so the more complex evaluations can be crafted to your requirements.

Look at the Java class that will do the work. I'm going to reuse some of the methods in the core and Hadoop jobs that I created earlier to calculate the mean and figure out how many months' sales were below the mean.

```java
import java.io.IOException;
import org.apache.pig.EvalFunc;
```

```
import org.apache.pig.data.Tuple;

public class PigCalcMonthsBelow extends EvalFunc<Integer> {

        @Override
        public Integer exec(Tuple tuple) throws IOException {
            int[] months = new int[12];
            for(int count = 1; count <= 12; count++) {
            months[count-1] = (Integer)tuple.get(count);

    }
        return monthsBelow(months, getMean(months));
}

    private int monthsBelow(int[] data, double mean) {
        int count = 0;
        for(double a : data) {
        if((a < (mean * 0.40))) count++;
}
        return count;
}

    private double getMean(int[] data) {
        double sum = 0.0;
        for(double a : data) {
        sum += a;
}
        return sum/data.length;
        }
}
```

It's a basic class, but as you can see there is some Pig-related code in there with the `exec` function, which does the work for you. (This is the method that is used when the Pig script calls the function.)

After this class is exported to a `jar` file, then you can use it within your Pig script. The `REGISTER` command loads the `jar` file into Pig for you. To call your custom function, you can use the name of the class including the full package name.

```
REGISTER salesudp.jar

SD = LOAD 'salesdata.csv' USING PigStorage(',') AS (custid:int, m1:int,
m2:int, m3:int, m4:int, m5:int, m6:int, m7:int, m8:int, m9:int, m10:int,
m11:int, m12:int, m13:int);

AVGDATA =  FOREACH SD GENERATE custid,
            AVG({m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12}),
            AVG({m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12})-m13,
            PigCalcMonthsBelow(*);
DUMP(AVGDATA);
```

In this instance, the whole tuple of your data has been passed into the `exec` function of the Java class. It iterates the month data and creates a `double[]` array

and also stores the month 13 value. Then it calculates the mean and the months below 40 percent of the average sales and passes that number back to Pig.

Storing the Output Data

So far, the only output you've done is to the console. This is far from helpful when other people want to see the data. The STORE function writes the output data to a file. It's very similar to the LOAD function that you used earlier:

```
STORE DATA INTO 'myoutputfile' USING PigStorage(',');
```

This command would store the tuple data in DATA into a comma-separated text file called myoutputfile. So, to complete your task. you amend your Pig script accordingly:

```
REGISTER salesudp.jar

SD = LOAD 'salesdata.csv' USING PigStorage(',') AS (custid:int, m1:int,
m2:int, m3:int, m4:int, m5:int, m6:int, m7:int, m8:int, m9:int, m10:int,
m11:int, m12:int, m13:int);

AVGDATA =   FOREACH SD GENERATE custid,
            AVG({m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12}),
            AVG({m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12})-m13,
            PigCalcMonthsBelow(*);

STORE AVGDATA INTO 'salesoutput' USING PigStorage(',');
```

When the job is run this time, you see Pig go through its usual routines and process the script. Instead of dumping the entire output to the console this time, you see the following:

```
HadoopVersion   PigVersion    UserId     StartedAt      FinishedAt
Features
1.2.1    0.12.0    Jason    2014-01-26 07:56:12    2014-01-26 07:56:15
UNKNOWN

Success!

Job Stats (time in seconds):
JobId    Alias    Feature    Outputs
job_local1303731742_0001    AVGDATA,SD    MAP_ONLY    file:///Users/
Jason/cmepart2/salesoutput,

Input(s):
Successfully read records from: "file:///Users/Jason/cmepart2/salesdata.
csv"

Output(s):
Successfully stored records in: "file:///Users/Jason/cmepart2/
```

```
salesoutput"

Job DAG:
job_local1303731742_0001
```

This gives the status of the job and, most importantly, the state of the output. In this instance, it's been successful. The output lives in a directory using the name you specified in the STORE command. Change into that directory and you see a Hadoop-like output named file part-m-00000 that contains the output.

Joining the Customer Details Table

So far, you've been working on one table. The customer ID is hardly the politest way of reporting customer information back to the stakeholders. Under some circumstances you would not want to be sharing personal information with anyone, but I'm using this scenario solely as a way of illustration to join two tables of information together with Pig (for more information on PII please look at Chapter 2).

Within Pig, you can join tables of information together very much like a relational database would.

Assume that I have a customer table within the company database or that I could export the customers out of the customer relationship management (CRM) system. The schema is basic; it's just a customer ID, e-mail address, first name, and last name. (I've used www.fakenamegenerator.com to generate my customer data; none of it is real.)

```
10,LouieThornton@cuvox.de,Louie,Thornton
11,NicholasWong@dayrep.com,Nicholas,Wong
12,HarrietBrennan@teleworm.us,Harriet,Brennan
13,AimeeHill@armyspy.com,Aimee,Hill
14,LaraBoyle@rhyta.com,Lara,Boyle
15,AmelieHowell@superrito.com,Amelie,Howell
16,GeorgiaAbbott@gustr.com,Georgia,Abbott
17,BradleyShah@einrot.com,Bradley,Shah
18,EvieRoss@teleworm.us,Evie,Ross
19,EllaOSullivan@einrot.com,Ella,O'Sullivan
20,ReeceEvans@cuvox.de,Reece,Evans
21,HenryPalmer@rhyta.com,Henry,Palmer
22,KayleighMoss@teleworm.us,Kayleigh,Moss
23,RobertHobbs@einrot.com,Robert,Hobbs
24,LoganGraham@superrito.com,Logan,Graham
```

Within the Pig script, you need to load both sets of data: the customers and the sales data:

```
CONSUMER = LOAD 'customer.csv' USING PigStorage(',') AS (custid:int,
```

```
email:chararray, firstname:chararray, lastname:chararray);

SD = LOAD 'salesdata.csv' USING PigStorage(',') AS (custid:int, m1:int,
m2:int, m3:int, m4:int, m5:int, m6:int, m7:int, m8:int, m9:int, m10:int,
m11:int, m12:int, m13:int);
```

You can see that the schema for the customer data LOAD statement has the data type chararray for the e-mail, first name, and last name. The sales data is loaded into a different variable reference, so there are now two data sets defined. They're just not joined together.

To join two data sets together in Pig is a simple matter of using the JOIN command. It takes two data sets with a common ID (in our case the customer ID) and joins them into a new data set:

```
FULLDATA = JOIN CONSUMER BY (custid), SD BY (custid);
```

If you dumped out the FULLDATA tuple after joining the two data sets, the output would look like the following:

```
(19994,FreddieWoodward@teleworm.us,Freddie,Woodw
ard,19994,9,4,4,2,11,9,10,4,2,3,13,9,0)
(19995,LillyMahmood@cuvox.de,Lilly,Mahm
ood,19995,5,2,1,1,14,4,9,12,4,13,6,7,14)
(19996,DavidHudson@cuvox.de,David,Hud
son,19996,3,6,6,13,8,1,14,10,5,0,7,9,14)
(19997,AliciaSmart@teleworm.us,Alicia,Sm
art,19997,4,8,13,3,14,11,4,2,11,12,13,8,1)
(19998,EmmaGrant@teleworm.us,Emma,Gr
ant,19998,8,10,5,8,12,9,7,8,14,8,14,7,12)
(19999,JoshGallagher@superrito.com,Josh,Gallag
her,19999,10,13,14,4,10,14,2,9,11,14,9,14,8)
(20000,HenryColeman@armyspy.com,Henry,Cole
man,20000,2,4,7,2,9,10,8,13,7,1,5,9,3)
```

Piecing It All Together

With all the knowledge you've acquired about Pig and creating custom functions with UDFs, you can now look at completing the original task:

```
REGISTER salesudp.jar

SD = LOAD 'salesdata.csv' USING PigStorage(',') AS (custid:int, m1:int,
m2:int, m3:int, m4:int, m5:int, m6:int, m7:int, m8:int, m9:int, m10:int,
m11:int, m12:int, m13:int);

CONSUMER = LOAD 'customers.csv' USING PigStorage(',') AS (custid:int,
email:chararray, firstname:chararray, lastname:chararray);

AVGDATA =   FOREACH SD GENERATE custid,
            AVG({m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12}),
            AVG({m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12})-m13,
```

```
        PigCalcMonthsBelow(*);

FULLDATA = JOIN CONSUMER BY (custid), AVGDATA BY (custid);

STORE FULLDATA INTO 'salesoutput' USING PigStorage(',');
```

First, the custom `jar` file is loaded into Pig; you then load the two data sets into two separate variables (`SD` for the sales data and `CONSUMER` for the customer data). You join the two data sets together using the customer ID as the key, creating a new data set called `FULLDATA`, then iterate through the data to do your calculations for the average sales, the sales drop, and the months below the average. Finally, you write the results into a new comma-separated file.

Once Pig has run you can inspect the output in the `salesoutput` folder.

```
1,EloiseBradley@dayrep.com,Eloise,Bradley,1,8.75,3.75,1
2,ChloeJohnston@einrot.com,Chloe,Johns
ton,2,5.916666666666667,-8.083333333333332,4
3,TomClark@rhyta.com,Tom,Clark,3,7.583333333333333,5.583333333333333,4
4,AliciaArmstrong@armyspy.com,Alicia,Armstr
ong,4,7.833333333333333,-1.166666666666667,4
5,RobertLong@superrito.com,Robert,L
ong,5,6.166666666666667,0.16666666666666696,2
6,OliviaOsborne@cuvox.de,Olivia,Osborne,6,5.5,0.5,3
7,EllieChandler@dayrep.com,Ellie,Chand
ler,7,7.416666666666667,-6.583333333333333,3
8,SophieDean@gustr.com,Sophie,D
ean,8,8.833333333333334,-0.16666666666666607,2
9,DeclanBruce@gustr.com,Declan,Br
uce,9,7.416666666666667,6.416666666666667,2
10,LouieThornton@cuvox.de,Louie,Thorn
ton,10,5.666666666666667,-3.333333333333333,4
```

What this section aimed to do was give you a broad overview of what could be achieved with a set of data-processing tools in a real-world context. Word count examples are all good, but they only scratch the surface of what's possible.

I've tried to avoid explaining how the core of each technology works, because some of those subjects require books in themselves. Instead, I offered a hands-on "get-it-working" approach that can be used in your projects quickly. To further your learning on Hadoop, Pig, Mahout, and Sqoop, have a look at the "Further Reading" appendix at the end of this book.

You now have a framework for giving recommendations, exporting data from existing data sources, and mining sales data. You can easily modify these tutorials for your own means. Just sit down, plan, and think about what question you're trying to answer. Have a proper talk with stakeholders in the company to find out what benefit the data could bring to them. Sometimes the answers and ideas are surprising.

Scheduling Batch Jobs

Within this chapter you've dealt with batch processing but not with repeating the task over and over again. The obvious choice for scheduling something to run is by using a scheduled task, Launch Daemon job, or `cron` job.

To set up a `cron` job on a UNIX system or Mac OS X, you need to amend the `crontab` file. From the command line, run the following command:

```
crontab –e
```

Depending on the operating system of your machine, if a default text editor hasn't been defined, then you are prompted for one. After the text editor has opened, then you can create your `cron` job.

A `cron` job is specified as a time when the job is scheduled to be run and this is represented as white-space-separated fields comprising the minutes, hours, day of month, month, and day of week, followed by the command to run. For example, to run a job every day at 11:30 a.m., the `cron` job syntax would look like the following:

```
30 11 * * *  echo "It's 11:30am"
```

After the text editor saves the `cron` job line, then `crontab` stores the schedule and runs the process at the desired time.

To put this in context of your hashtags example, perhaps you want to check every hour what top-trending tags are from Spring XD (remember it's streaming the data into HDFS for you already), so you can write a shell script to run the task:

```
#!/usr/bin/env bash
# runhashtag.sh – Hadoop shell script for Spring XD hashtags
hadoop jar hashtagmining.jar HashtagJob /xd/fashiontweets /fashionoutput
hadoop fs -getmerge /fashionoutput fashionoutput.txt
sort –k2nr fashionoutput.txt | head -20 > hourlytrends.txt
```

The following `crontab` entry would run the shell script every hour on the hour and output the top 20 results to a text file:

```
0 * * * * svc –o /path/to/script/runhashtag.sh
```

If you want to learn more about `crontab`, you can either read the manual pages on your UNIX distribution (type **man cron** at the command line) or consult the Wikipedia page on `cron` at `http://en.wikipedia.org/wiki/Cron`.

Although this is a simple approach, you must be careful. If the volumes of data become great, you might reach a stage where the processing is taking longer to produce results than the time difference you leave in between the `cron` job.

It's worth adding mutual exclusion to your job and having it exit if an earlier incarnation of itself is running, skipping those hours where earlier jobs have

taken too long to complete. It might also be worth having the job "give up" and exit before it completes, if it takes much too long (perhaps 3 hours).

Summary

Hadoop is an important technology, and it's worth putting some time aside to craft some MapReduce programs that might be of use in your organization. You also have to think about the openly available data that's available to you. Weather data is a gold mine if you are in the retail business, and with a carefully crafted algorithm you could consume the weather data and compare it with sales data. Do sales increase in certain weather conditions? If the weather can be predicted five days out, then can you do the same with the potential sales?

If crafting MapReduce seems like a lot of work, then think about using Pig instead. It might be just as quick to solve simple problems against large sets of data. Remember that Pig is designed to work with data flow, so when complexity creeps in, it might be worth looking at a coded solution.

Machine learning sometimes isn't so much about designing heavy-duty algorithms as it is a case of finding the right tools to do the simple job.

With a small amount of exposure to Hadoop, Sqoop, Mahout, and Pig, there's a broad set of tools for extracting, processing, and learning from existing and newly acquired data. Coupling this with the different types of machine learning outlined in the chapters of this book will give you a good grounding in what's possible.

I used Hadoop version 1 because it gives a good grounding on how MapReduce jobs are put together. In a real-world context you would be using Hadoop version 2 because it performs better and also has the added feature of YARN (Yet Another Resource Negotiator) for job deployment. It's also worth noting that in memory systems like Spark are gaining in popularity; read on to Chapter 11 for more information about Spark.

Apache Spark

The Apache Spark project was created by the AMPLab at UC Berkeley as a data analytics cluster computing framework. This chapter is a quick overview of the Scala language and its use within the Spark framework. The chapter also looks at the external libraries for machine learning, SQL-like queries, and streaming data with Spark.

Spark: A Hadoop Replacement?

The debate about whether Spark is a Hadoop replacement might rage on longer than some would like. One of the problems with Hadoop is the same thing that made it famous: MapReduce. The programming model can take time to master for certain tasks. If it's a case of straight totaling up frequencies of data, then MapReduce is fine, but after you get past that point, you're left with some hard decisions to make.

Hadoop2 gets beyond the issue of using Hadoop only for MapReduce. With the introduction of YARN (Yet Another Resource Negotiator) Hadoop acts as an operating system for data with YARN controlling resources against the cluster. These resources weren't limited to MapReduce jobs; they could be any job that could be executed. An excellent example of this was the deployment of JBoss application server containers in the book *Apache Hadoop YARN* by Arun

C. Murthy and Vinod Kumar Vavilapalli (Addison-Wesley Professional, 2014; see the "Further Reading" section at the end of this book for more details).

The Spark project doesn't rely on MapReduce, which gives it a speed advantage. The claim is that it's 100 times faster than in-memory Hadoop and 10 times faster on disk. If speed is an issue for you, then Spark is certainly up there on the list of things to look at. As for the argument that it's a replacement for Hadoop. . . . well, I'll leave that for the technical press to suss out. It all boils down to the fundamental element of what question you are trying to answer. The tools are just that—tools.

Java, Scala, or Python?

Spark jobs can be written in Java, Scala, or Python. As usual, which language you use tends to be a matter of personal preference. Because it is written in the JVM language Scala, the primary language for Spark is Scala and, with the inclusion of a command-line tool, this makes life easier when you want to get answers more quickly.

The application programming interface (API) is available to Java developers, but there's more work required in getting programs ready. The Python language is also supported, and it, like Scala, comes with a command-line interpreter for Spark jobs.

The bottom line is, if you want to work on the command line, then you should use Scala or Python. If the thought of Scala scares you, read on and I'll attempt to help make you feel better. On the other hand, if you want to skip this part, then you can proceed to the "Downloading and Installing Spark" section.

As you progress through this chapter, you might come to the conclusion that Java is too bulky for quickly writing Spark jobs. Remember, it's a matter of personal preference, so you should use the language in which you're most comfortable.

Scala Crash Course

If you are a Java programmer and you want to move to Scala, this section covers the essentials in a few digestible pages.

For a more detailed look at the Scala language, try the site documentation at `www.scala-lang.org/documentation`.

Installing Scala

Scala is classed as a Java Virtual Machine (JVM) language, because it uses the JVM to execute its bytecodes. There is a download that you need to perform

before you get into the basics. If you are happy to use Spark on its own, then you don't need to download Scala, because the libraries are in the distribution. On the other hand, if you want to write some Scala programs to run on Spark afterward then it's prudent to download the Scala distribution file and install it. You can find the downloads at `www.scala-lang.org/download/all.html`.

Pick a fairly recent version for your operating system (nothing too cutting edge) and download it. After it has downloaded, find a place for it to live and then uncompress the archive (usually a `.tgz` file for Mac OS X/Linux or a Zip file for Windows users).

Ensure that the path to the `bin` directory is set, so you can access the programs from your command line. When you have all that done, you can proceed.

Packages

Package imports behave in the same way as Java imports, but instead of using the `*` as you do in Java, you use `_` in Scala.

```
import java.util._
import org.my.thing._
```

Scala is based on the JVM, so you can import and use Java libraries and third-party libraries.

Data Types

Scala supports the following seven numeric data types. They are classes. Scala doesn't support primitive data types like Java does.

- Byte
- Char
- Short
- Int
- Long
- Float
- Double
- Boolean

Because they are all classes, you can use the class methods to manipulate them in the same way as Java:

```
scala> 1000.toString
res10: String = 1000
scala> 1000.toDouble
```

```
res11: Double = 1000.0
scala> 1000.toInt
res13: Int = 1000
```

Declarations of constants and variables are defined as `val` or `var`; it's not a big deal which one you use.

Classes

Scala source files can contain as many classes as you want in the same way as Java:

```
class MyClass {
    var myval = 0
}
```

Like Plain Old Java Objects (POJOs), where you create a public variable name and then create two getter and setter methods, Scala is doing the same. The variable (`myval` in the example) is accessed like so:

```
val newclass = new Myclass
newclass.myval = 42
println(newclass.myval)
```

The primary constructor in Scala classes is public. All variables have to be initialized within Scala code; it's not an optional thing.

Within classes you can define methods in the same way as you can in Java. So, a basic void method would look like the following:

```
def myMethod() {
}
```

You can also add return types as you do in Java. So, assuming you want to return an integer data type from your method, you'd type:

```
def myMethod(): Int = {
    Return 42
}
```

The final example shows how you pass values into the function. Add two numbers together and return the answer, like so:

```
def addNums(a: Int, b: Int) : Int = {
return a + b
}
```

Calling Functions

Function calls in Scala are very much the same as in Java. Using the dot notation against the object you can call the method name. If no values are to be passed

(like `.addNumber(1,2)`, for example), then the parentheses are optional. Unlike Java, though, there are no static methods in Scala.

Operators

The operators in Scala operate in the same way as the majority of the Java operators.

Control Structures

Scala supports the usual mix of `if`, `while`, and `for` control structures.

for Loops

`for` loops can work on any collection. For example:

```
val filesInDir = (new java.io.File("/home/user/").listFiles
for(thisFile <- filesInDir)
    println(thisFile)
```

You also have ranges at your disposal, which is simpler than the Java equivalent:

```
for(i <- 1 until 10)
    println(i)
for(i <- 1 to 10)
    println(i)
```

You can also filter within the `for` loop. The following is an example with the output from my home directory. (I've made the actual commands bold.)

```
scala> val fileshere = (new java.io.File("/home/jason/").listFiles)
fileshere: Array[java.io.File] = Array(/home/jason/twitterstream.jar,
/home/jason/.ssh, /home/jason/.profile, /home/jason/spring-shell.log,
/home/jason/testyarn.jar, /home/jason/twitterstreamtransformer.xml,
/home/jason/.bashrc, /home/jason/worldbrain.txt, /home/jason/.bash_
logout, /home/jason/.spark_history, /home/jason/.bash_history)

scala> for(file <- fileshere if file.getName.endsWith(".xml"))
print(file)
/home/jason/twitterstreamtransformer.xml
```

while Loops

`while` loops operate in the same way as Java `while` loops:

```
while(value != 100) {
    value += 1
}
```

if Statements

As you'd expect, `if` statements in Scala are similar to Java `if` statements. Scala supports ternary operators as Java does.

```
Java:
boolean debug = server.equals("localhost") ? true : false;

Scala:
var debug = if (server.equals("localhost")) true else false;
```

Downloading and Installing Spark

There are a few ways to download and use Spark. The easiest way is to use the prebuilt packages that are available from the Spark website. Before you download one, check which version of Hadoop you (or your organization) is running as it will determine the download you want. At the time of writing, Spark was available for

- Hadoop 1—(Hortonworks HDP1 and Cloudera CDH3)
- Hadoop 2—(Hortonworks HDP2 and Cloudera CDH5)
- Cloudera CDH4

You are not required to exclusively run Hortonworks or Cloudera. Spark works fine on a download of Hadoop 1 or 2 from the Apache site.

After you have downloaded the file for your Hadoop version, you can install it by moving the downloaded file to the directory you want to install it to. The file is a `.tgz` file, so you can unarchive it in one command:

```
tar xvzf spark-1.0.0-bin-hadoop2.tgz
```

The contents of the file will unarchive and be ready for use.

A Quick Intro to Spark

The interactive shell in Scala gives you a quick and easy way to see what Spark can do in a very short span of time. If you read the really short introduction to Scala earlier in the chapter, then you should cope well with the following examples.

Starting the Shell

From the directory where you installed Spark, type the following command to launch the shell:

```
./bin/spark-shell
```

You see the following output while Spark boots up:

```
jason@cloudatics:/usr/local/spark$ bin/spark-shell
Spark assembly has been built with Hive, including Datanucleus jars on
classpath
14/07/05 09:28:44 INFO SecurityManager: Using Spark's default log4j
profile: org/apache/spark/log4j-defaults.properties
14/07/05 09:28:44 INFO SecurityManager: Changing view acls to: jason
14/07/05 09:28:44 INFO SecurityManager: SecurityManager: authentication
disabled; ui acls disabled; users with view permissions: Set(jason)
14/07/05 09:28:44 INFO HttpServer: Starting HTTP Server
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.0.0
      /_/

Using Scala version 2.10.4 (OpenJDK 64-Bit Server VM, Java 1.6.0_31)
Type in expressions to have them evaluated.
Type :help for more information.
14/07/05 09:28:51 INFO SecurityManager: Changing view acls to: jason
14/07/05 09:28:51 INFO SecurityManager: SecurityManager: authentication
disabled; ui acls disabled; users with view permissions: Set(jason)
14/07/05 09:28:52 INFO Slf4jLogger: Slf4jLogger started
14/07/05 09:28:52 INFO Remoting: Starting remoting
14/07/05 09:28:52 INFO Remoting: Remoting started; listening on
addresses :[akka.tcp://spark@cloudatics.com:38921]
14/07/05 09:28:52 INFO Remoting: Remoting now listens on addresses:
[akka.tcp://spark@cloudatics.com:38921]
14/07/05 09:28:52 INFO SparkEnv: Registering MapOutputTracker
14/07/05 09:28:52 INFO SparkEnv: Registering BlockManagerMaster
14/07/05 09:28:52 INFO DiskBlockManager: Created local directory at /
tmp/spark-local-20140705092852-2f00
14/07/05 09:28:52 INFO MemoryStore: MemoryStore started with capacity
297.0 MB.
14/07/05 09:28:53 INFO ConnectionManager: Bound socket to port 48513
with id = ConnectionManagerId(cloudatics.com,48513)
14/07/05 09:28:53 INFO BlockManagerMaster: Trying to register
BlockManager
```

```
14/07/05 09:28:53 INFO BlockManagerInfo: Registering block manager
cloudatics.com:48513 with 297.0 MB RAM
14/07/05 09:28:53 INFO BlockManagerMaster: Registered BlockManager
14/07/05 09:28:53 INFO HttpServer: Starting HTTP Server
14/07/05 09:28:53 INFO HttpBroadcast: Broadcast server started at
http://176.67.170.176:41692
14/07/05 09:28:53 INFO HttpFileServer: HTTP File server directory is /
tmp/spark-c6a5efed-113d-4dd9-9cd0-7618d901f389
14/07/05 09:28:53 INFO HttpServer: Starting HTTP Server
14/07/05 09:28:53 INFO SparkUI: Started SparkUI at http://cloudatics.
com:4040
14/07/05 09:28:54 WARN NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
14/07/05 09:28:54 INFO Executor: Using REPL class URI:
http://176.67.170.176:39192
14/07/05 09:28:54 INFO SparkILoop: Created spark context..
Spark context available as sc.

scala>
```

I'm showing all the messages, because there are a few interesting things I want to show you in a moment. For now, though, you should see the `scala>` prompt at the bottom, which means you're ready.

Data Sources

Spark supports the same input file systems as Hadoop. If your data store is supported by the `InputFormat` method in Hadoop, then you can read it into Spark without too much effort.

The obvious ones that come to mind are the local filesystem, Amazon S3 buckets, HBase, Cassandra, and files already on a Hadoop Distributed File System (HDFS). As you see in the next section, you're using the `sc.textFile` method to load in data. This isn't limited to specific files; you can use that method to process wildcards on files, zipped files, and directories as well.

Testing Spark

Find a text file with which to test Spark and follow along with the rest of this section. I'm using a file from the local filesystem for this example.

Load the Text File

First, load the text file. From the Scala command line, type

```
scala> var textF = sc.textFile("/home/jason/worldbrain.txt")
```

Basically you're storing the contents of the text file into a Scala variable called `textF`. Spark responds with output along the lines of the following:

```
14/07/05 09:40:08 INFO MemoryStore: ensureFreeSpace(146579) called with
curMem=0, maxMem=311387750
14/07/05 09:40:08 INFO MemoryStore: Block broadcast_0 stored as values
to memory (estimated size 143.1 KB, free 296.8 MB)
textF: org.apache.spark.rdd.RDD[String] = MappedRDD[1] at textFile at
<console>:12

scala>
```

Spark uses a concept called Resilient Distributed Datasets (RDDs), so in the output you can see that you now have a MappedRDD containing strings. With the text file loaded, you can start to inspect it and get some results.

Make Some Quick Inspections

With the data loaded, you can do some quick inspections. First, how many elements of data do you have in the RDD?

```
scala> textF.count()
```

You get the output:

```
14/07/05 09:58:01 INFO SparkContext: Job finished: count at
<console>:15, took 0.036546484 s
res4: Long = 469

scala>
```

This count represents the number of elements in the RDD and not the number of lines in the text file. You can pull the first element from the RDD:

```
scala> textF.first()
And the output appears:
14/07/05 10:00:20 INFO SparkContext: Job finished: first at
<console>:15, took 0.007266678 s
res5: String = THE papers and addresses I have collected in this little
book are submitted as contributions,......
scala>
```

As you can see from the output, these results are returning very quickly as the RDD is based in memory.

Filter Text from the RDD

With the `.filter` function, you can start to inspect specific things within the text file. Assuming that you want to see how many times the word "statistical" occurs in the document, you can run the following:

```
scala> textF.filter(line => line.contains("statistical")).count()
```

One line in Scala and the Spark filter iterate the RDD and inspect the lines. Because you've appended the count at the end, you get the following result:

```
14/07/05 10:16:55 INFO SparkContext: Job finished: count at
<console>:15, took 0.042640019 s
res7: Long = 2
```

So, there are two mentions of the word "statistical" in the entire document, and all completed in 0.04 seconds. If required, you could save this output to another Scala array:

```
scala> var filtered = textF.filter(line => line.contains("statistical"))
filtered: org.apache.spark.rdd.RDD[String] = FilteredRDD[4] at filter at
<console>:14
scala> filtered.count
14/07/08 09:20:16 INFO SparkContext: Job finished: count at
<console>:17, took 0.067395793 s
res4: Long = 2
```

With the aid of concise commands in Scala and the power of in-memory processing of distributed datasets (the RDDs), you have a neat system to get large amounts of crunching done quickly.

Spark Monitor

Earlier, I mentioned that when you start Spark a few things are being run. One of them is the web-based monitor. If you point your browser to `http://<yourdomain>:4040`, you should get the website shown in Figure 11-1, assuming Spark is still running.

As far as Spark is concerned, every line you run is a job, so Spark logs it accordingly, giving its duration and outcome. There's also information on the storage and runtime environment.

You can click on each of the stages to see full details of the job and its execution output. If one of your jobs is causing trouble, then it's handy to look here first and get a bird's eye view of things.
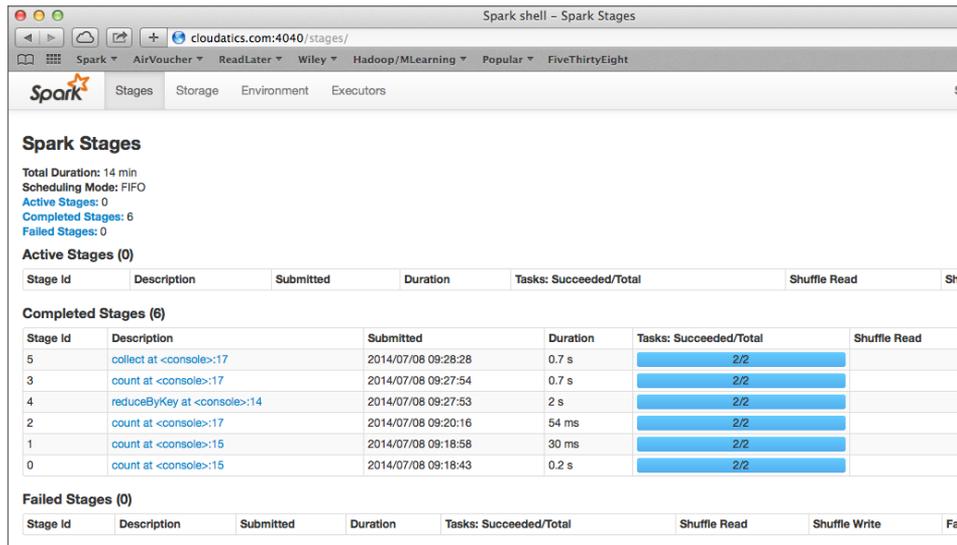
Figure 11-1: Spark web console

Comparing Hadoop MapReduce to Spark

Recall from Chapter 10 that it required a certain amount of effort to create a MapReduce program in Java to work with Hadoop.

The basic Java code boilerplate is usually along the lines of the map and reduce phases:

```
public static class Map extends MapReduceBase implements
          Mapper<LongWritable, Text, Text, IntWritable> {
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();

      public void map(LongWritable key, Text value,
              OutputCollector<Text, IntWritable> output, Reporter
reporter)
              throws IOException {
          // ususally emit something to the reducer here....
      }
    }

    public static class Reduce extends MapReduceBase implements
          Reducer<Text, IntWritable, Text, IntWritable> {
```

```
        public void reduce(Text key, Iterator<IntWritable> values,
                OutputCollector<Text, IntWritable> output, Reporter
reporter)
                throws IOException {
            // reducer would add the value +1 for example

        }
    }
```

Then a job definition enables it to run within the Hadoop framework:

```
public static void main(String[] args) throws IOException {
        JobConf conf = new JobConf(BlankHadoopJob.class);
        conf.setJobName("BlankHadoopJob");

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        conf.setMapperClass(Map.class);
        conf.setCombinerClass(Reduce.class);
        conf.setReducerClass(Reduce.class);

        conf.setInputFormat(TextInputFormat.class);
        conf.setOutputFormat(TextOutputFormat.class);

        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        JobClient.runJob(conf);

    }
```

Some developers complain about the amount of code you need to write in Java to get MapReduce working. It's never been an issue for me (as I have templates set up), but when I show you how it works in Spark, you'll realize why they were complaining.

In Spark, you can put together a quick word count MapReduce routine that demonstrates how easy it is to do. Using the same text file you used earlier, you can run a MapReduce process from the Spark shell. First load the text file:

```
scala> var textF  = sc.textFile("/home/jason/worldbrain.txt")
```

Then create a new variable with the results of the MapReduce:

```
scala> var mapred = textF.flatMap(line => line.split(" ")).map(word =>
(word, 1)).reduceByKey((a,b) => a+b)
```

Then output the results:

```
scala> mapred.collect
```

You see a block of the output appear in the shell:

```
14/07/08 09:28:29 INFO SparkContext: Job finished: collect at
<console>:17, took 0.78279487 s
res7: Array[(String, Int)] = Array((professors,,1), (mattered,1),
(intimately,1), (better.,3), (someone,3), (House,1), (manifestly,1),
(order,13), (socialism.,1), (apprehension,4), (conclusively,1),
(gowns,2), (behind,3), (Out",,1), (merge,1), (wasn't,1), (been,125),
(Judea;,1), (gap,5), (underrate,1), (aspects;,1), (knows,8),
(informative,15), (divorced,1), (are,259), (records,4), (2.,1),
(Western,3), (politician,,1), (room—and,1), (newspapers,,3),
(picture.,1), (interchange—from,1), (prete
```

To save the results, use the `.saveAsTextFile` method on the RDD to output as text:

```
scala> mapred.saveAsTextFile("/home/jason/testoutput")
```

Spark, in the same way as Hadoop, saves the files in a directory (I called this one `testoutput`). Within it you see the `part-00000` files:

```
-rw-r--r-- 1 1234 1234 52961 Jul  8 13:47 part-00000
-rw-r--r-- 1 1234 1234 52861 Jul  8 13:47 part-00001
-rw-r--r-- 1 1234 1234     0 Jul  8 13:47 _SUCCESS
```

The output of those files contains the basic word count:

```
(lags—throughout,1)
(however,9)
(cry.,1)
(eminent,2)
(dangerous,5)
(varieties,1)
(History.,1)
(behind,,1)
(late,3)
(nineteenth,6)
(helpless,1)
(throwing,2)
(aesthetic,4)
(leapt,2)
```

In three lines you performed a basic MapReduce program on some raw text. Notice that I didn't remove odd characters and convert everything to lowercase, but essentially it gave us the word count output.

Writing Standalone Programs with Spark

The brief introduction to Spark has shown you that it's fast, and you can perform certain tasks with ease. The shell is useful for inspecting datasets and getting a basic set of answers, counts, or frequencies from the data. There are times, though, when full programs are required to be written. As previously discussed, Spark programs can be written in Scala, Java, or Python, because there's a supporting API for each of them. This section concentrates on Scala and Java.

NOTE The Scala program requires the Scala libraries and compiler. Instructions on how to install them are provided earlier in this chapter.

Spark Programs in Scala

Scala applications are very similar to Java applications in their construction. With the Scala Build Tool (`sbt`), you can manage the building of everything in one place, too.

It's worth treating programs as small projects, so keeping the code in its own directory makes maintenance more manageable.

Installing SBT

The Scala Build Tool is a separate download from the main Scala libraries, so you need to download and install that as well. The downloads are available from `www.scala-sbt.org/download.html`. Download the file that's appropriate for your operating system and unarchive it. Make sure the binary is in your path, so you can execute the `sbt` file.

The Scala Program Code

Create a project called `ScalaMRExample`, then create the `src` and `main` directories within it:

```
mkdir –p ScalaMRExample/src/main
```

Keeping the MapReduce example in mind, the Spark application, in a file named "ScalaMRExample.scala" would look like so:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object ScalaMRExample {
  def main(args: Array[String]) {
    val configuration = new SparkConf().setAppName("Scala MapReduce
Example")
    val sc = new SparkContext(conf)
    val textFile = "/home/jason/worldbrain.txt"
    val textSc = sc.textFile(textFile)
    val mapred = textFile.flatMap(line => line.split(" ")).map(word =>
(word, 1)).reduceByKey((a,b) => a+b)

    mapred.collect

    mapred.saveAsTextFile("wboutput")

  }
}
```

You must adjust the filenames and paths for your environment, of course. The program imports the required Spark libraries and then sets up the object name and the main method to run. As you can see, it's very similar to Java in those respects.

To load the system properties and the active classpath, you create a `SparkConf` object and pass in the name of your application, in this case `ScalaMapReduce Example`. The remainder of the program is the same as what has been previously covered. Load in the text file, run the simple MapReduce, output the results, and then save the output results to a location.

The Scala Build Tool File

The Scala Build Tool is similar to Java build tools such as Ant and Maven. You need a build file in the `ScalaMRExample` directory that tells Scala about the project, its dependencies, and where to get them:

```
name := "ScalaMRExample"

version := "0.1"

scalaVersion := "2.10.4"
```

```
libraryDependencies += "org.apache.spark" %% "spark-core" % "1.0.0"

resolvers += "Akka Repository" at "http://repo.akka.io/releases/"
```

Save this file as `scalamrexample.sbt`, and then you can use `sbt` to package up the project. Inspect the directory structure to make sure everything is in place. In Linux/MacOS X, you can use the `find` command:

```
Jason-Bells-MacBook-Pro:ScalaMRExample Jason$ find . -print
.
./scalamrexample.sbt
./src
./src/main
./src/main/scala
./src/main/scala/ScalaMRExample.scala
```

When you're happy with the file listing (it should look like the preceding code), then you can package it up using `sbt`. Make sure you are in the top-level ScalaMRExample.

`sbt package`

The first run might take some time, as `sbt` might have to download external libraries. When that's happened, though, the code compiles and the package is created:

```
Jason-Bells-MacBook-Pro:ScalaMRExample Jason$ sbt package[info] Set
current project to ScalaMRExample (in build file:/Users/Jason/work/
scala/ScalaMRExample/)
[info] Compiling 1 Scala source to /Users/Jason/work/scala/
ScalaMRExample/target/scala-2.10/classes...
[info] Packaging /Users/Jason/work/scala/ScalaMRExample/target/scala-
2.10/scalamrexample_2.10-1.0.jar ...
[info] Done packaging.
[success] Total time: 8 s, completed 09-Jul-2014 17:25:49
```

You see the newly compiled package in the `target/scala-2.10` directory:

```
Jason-Bells-MacBook-Pro:ScalaMRExample Jason$ cd target/scala-2.10/
Jason-Bells-MacBook-Pro:scala-2.10 Jason$ ls -l
total 16
drwxr-xr-x  7 Jason  staff   238  9 Jul 17:25 classes
-rw-r--r--  1 Jason  staff  4495  9 Jul 17:25 scalamrexample_2.10-
1.0.jar
Jason-Bells-MacBook-Pro:scala-2.10 Jason$
```

Executing the Spark Project

It's the `jar` file that you need to use with Spark. Assuming you are in the same directory where the `jar` file resides, use the `spark-submit` script to run the file:

```
/usr/local/spark/bin/spark-submit --class "ScalaMRExample" \
--master local[4] scalamrexample_2.10-1.0.jar
```

The `--master local[4]` option tells Spark to create four local nodes when it starts up. If all goes according the plan, you see Spark whirr into action and perform the task:

```
14/07/09 19:32:42 INFO TaskSetManager: Finished TID 5 in 922 ms on
localhost (progress: 2/2)
14/07/09 19:32:42 INFO TaskSchedulerImpl: Removed TaskSet 2.0, whose
tasks have all completed, from pool
14/07/09 19:32:42 INFO DAGScheduler: Completed ResultTask(2, 1)
14/07/09 19:32:42 INFO DAGScheduler: Stage 2 (saveAsTextFile at
ScalaMRExample.scala:15) finished in 0.924 s
14/07/09 19:32:42 INFO SparkContext: Job finished: saveAsTextFile at
ScalaMRExample.scala:15, took 1.014446651 s
```

As you used `wboutput` as the target directory to save the text output to, you see that in your directory structure. You see that using Scala means that creating MapReduce-based programs becomes a simple matter, compared to the Hadoop way of doing things. I'm not saying that one is better than the other; it's just another way of doing things. Take a look at the Java way of creating Spark programs.

Spark Programs in Java

The concept of creating a Spark program written in Java is very much the same as the Scala example just covered. The main difference is the language and the build tool.

The Spark API for Java requires a little more thought than the Scala examples. You can't just rattle out a four-line program to perform a MapReduce task.

```
import scala.Tuple2;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
```

```java
import org.apache.spark.api.java.function.PairFunction;

import java.util.Arrays;
import java.util.List;
import java.util.regex.Pattern;

public final class JavaMRExample {
  private static final Pattern spacePattern = Pattern.compile(" ");

  public static void main(String[] args) throws Exception {

    SparkConf configuration = new SparkConf().
setAppName("JavaMRExample");
    JavaSparkContext ctx = new JavaSparkContext(configuration);
    JavaRDD<String> linesOfText = ctx.textFile("/home/jason/worldbrain.
txt", 1);

    JavaRDD<String> findWords = linesOfText.flatMap(new
FlatMapFunction<String, String>() {
      @Override
      public Iterable<String> call(String thisString) {
        return Arrays.asList(spacePattern.split(thisString));
      }
    });

    JavaPairRDD<String, Integer> getTheOnes = findWords.mapToPair(new
PairFunction<String, String, Integer>() {
      @Override
      public Tuple2<String, Integer> call(String thisString) {
        return new Tuple2<String, Integer>(thisString, 1);
      }
    });

    JavaPairRDD<String, Integer> finalCounts = getTheOnes.
reduceByKey(new Function2<Integer, Integer, Integer>() {
      @Override
      public Integer call(Integer i1, Integer i2) {
        return i1 + i2;
      }
    });

    List<Tuple2<String, Integer>> thisOutput = finalCounts.collect();
    for (Tuple2<?,?> tuple : thisOutput) {
      System.out.println(tuple._1() + ": " + tuple._2());
    }
    ctx.stop();
  }
}
```

There's a lot more code to put together than in the Scala MapReduce example program. The Java Spark API requires a little more thought in execution:

- Splitting the words
- Assigning a count of 1 for each word
- Adding up the counts for each unique word
- Generating the output

Using Maven to Build the Project

Throughout the book, I've used Eclipse to generate the projects and handle the build for me. Maven is a build tool (sbt was pretty much inspired by the Maven build process in my opinion) and, along with Ant, is the mainstay of Java build tools. If you don't have Maven installed, you can download it from http://maven.apache.org and unarchive the file.

For every project, you need a Maven build file; this is called pom.xml.

```xml
<project>
  <groupId>com.mlbook</groupId>
  <artifactId>javamrexample</artifactId>
  <modelVersion>4.0.0</modelVersion>
  <name>Java MR Example</name>
  <packaging>jar</packaging>
  <version>1.0</version>
  <repositories>
    <repository>
      <id>Akka repository</id>
      <url>http://repo.akka.io/releases</url>
    </repository>
  </repositories>
  <dependencies>
    <dependency>
      <groupId>org.apache.spark</groupId>
      <artifactId>spark-core_2.10</artifactId>
      <version>1.0.0</version>
    </dependency>
  </dependencies>
</project>
```

This gives you the basic outline of the project and which repositories to pull any required dependencies from. For Spark projects, you need to have the Spark API in the dependencies declaration.

Creating Packages in Maven

To create the package, you run Maven from the command line:

```
mvn package
```

The Maven build tool looks after the downloading of the dependencies, creates the class files, and then packages the `jar` file with the required classes. After it's built, a directory called `target` is created and you see the `jar` file there.

```
Jason-Bells-MacBook-Pro:JavaMRExample Jason$ cd target/
Jason-Bells-MacBook-Pro:target Jason$ ls -l
total 24
drwxr-xr-x  10 Jason  staff   340  9 Jul 18:19 classes
-rw-r--r--   1 Jason  staff  8679  9 Jul 18:19 javamrexample-1.0.jar
drwxr-xr-x   3 Jason  staff   102  9 Jul 18:04 maven-archiver
Jason-Bells-MacBook-Pro:target Jason$
```

To run the project with Spark, you need to use the `spark-submit` program like you did in the previous Scala example:

```
jason@cloudatics:~$ /usr/local/spark/bin/spark-submit --class
"JavaMRExample" \
--master local[4] javamrexample-1.0.jar
```

Spark executes the `jar` file, and you see the MapReduce output in the console:

```
14/07/09 20:37:08 INFO DAGScheduler: Stage 0 (collect at JavaMRExample.
java:44) finished in 0.784 s
14/07/09 20:37:08 INFO SparkContext: Job finished: collect at
JavaMRExample.java:44, took 2.47317507 s
young: 29
mattered: 1
intimately: 1
journeys.: 1
Let: 7
House: 1
(Socialism),: 1
instance,: 1
superannuated.: 1
proportions,: 1
Contributed: 1
secure: 4
incoherent.: 1
everyone.: 1
gowns: 2
well-informed: 1
```

Spark Program Summary

You've now seen two methods for quickly creating and building projects for the Spark framework. As previously mentioned, the decision about whether to use Java or Scala really depends on your comfort level. For production work, you really don't want to be using experimental code in a language with which you're not 100percent comfortable.

So, with the basics of Spark and creating standalone applications, you can now move forward and look at some of the libraries that are available for use within Spark.

Spark SQL

One of the great discussions in the Hadoop world has been "Can I run SQL (Structured Query Language)-like queries?" There's still something intrinsic in people who deal with data and their use of SQL. It's easy to see why; SQL is nice and language friendly. The Pig scripting language in Hadoop was close to getting to English-like MapReduce jobs, but the lure of true SQL queries is just too much for some, so they ask if it exists.

SparkSQL is just that—SQL-like queries within the Spark framework. It's just limited to SQL, so you can run Scala queries or even HiveSQL queries within the Spark framework. The data can be held within an external datastore like Apache Hive or loaded into memory as an RDD and queried that way.

> **NOTE** The SQL parser in SparkSQL will be useful for most people, but it's not as advanced as some might expect in a full database system. If you need that full flexibility, then you are advised to look at HiveSQL as an alternative. For the remainder of the section I concentrate on the SparkSQL parser by way of an introduction.

Basic Concepts

With the `SparkContext` in place, you can easily create a SparkSQL instance and associate that with the current context. This can be done within the shell in Scala or as a standalone program with Scala, Java, or Python. The program method is easier if you have maintenance of code in mind.

The basic code to incorporate the SparkSQL libraries is as follows:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
```

```scala
import org.apache.spark.SparkConf

object SparkSQLExample {
  def main(args: Array[String]) {
    val configuration = new SparkConf().setAppName("SparkSQL Example")
    val sc = new SparkContext(configuration)
    val sql = new org.apache.spark.sql.SQLContext(sc)

    // program continues.

  }
}
```

You need to add the SparkSQL library to the required dependencies in your `sbt` build file. I've created a new build file for the code:

```
name := "SparkSQLExample"

version := "1.0"

scalaVersion := "2.10.4"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.0.0"

libraryDependencies += "org.apache.spark" %% "spark-sql" % "1.0.0"

resolvers += "Akka Repository" at http://repo.akka.io/releases/
```

Don't forget to ensure there is a blank line between each text line. Otherwise, the build tool throws an error during the build.

Using SparkSQL with RDDs

It's time to create a real-world example of the SparkSQL system using the internal RDD as a storage system. With the data being held, in memory it will be fast. In addition, you'll open up the Scala language a little bit to show you an application with more than one `.scala` file in the build.

Generating Data

This example uses a `.csv` file with the following information:

- Unique Id
- First name
- Surname
- UK Postcode

■ Date of Birth (month/day/year format)

■ Latitude

■ Longitude

This could represent some form of customer database. You can either generate your own data or use a service such as fakenamegenerator.com to create a dataset for you. If you do use that service, ensure that you do not have the field names on the top line of the `.csv` file; delete it if necessary.

A sample of the data looks like this:

```
72215385-fad0-45fd-b932-a3d4fa07fb6d,William,Winter,DG3
7AL,1/4/1969,55.282416,-3.831666
71efb06f-63a7-4312-abe2-1fc4c8bd52e5,Billy,Noble,DD6
7FU,9/11/1969,55.597429,-3.170968
eac7b8f6-2d71-49a7-8e4d-a780826ce893,Kayleigh,Atkins,DG8
9ES,1/20/1932,54.73341,-4.407141
337d498d-9bed-4663-b688-6ec2651bcd9d,Emma,Perry,IM3
2GE,4/16/1935,54.173625,-4.480233
c3b0cc54-1f2f-416f-9b91-17a0700a4dc1,Liam,Carr,EX32
8AA,1/3/1991,50.682015,-3.119895
ea6acc38-1b69-424f-b885-09b078f6eaf2,Alex,Dodd,PH2
0QJ,6/24/1993,55.982172,-3.592573
```

With the data in place, you can turn your attention to the code.

The Scala Application

I'm going to split this application up logically. One Scala object to run the main program and Spark configuration and another to do the query work. The main program is going to output some log information as well, making it easier to keep track of what's going on while the program is being run. Take a look at the basic breakdown with the configuration:

```
import java.io._

import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

import org.apache.spark.sql.SQLContext

object SparkSQLExample {

  def main(args: Array[String]) {
```

```
    val conf = new SparkConf().setAppName("SparkSQLExample").
setMaster("local")

    conf.set("spark.eventLog.enabled", "true")
    conf.set("spark.eventLog.dir", "file:///home/jason/sparksqltest/
myeventlog/eventlog.out")


    val logger = new PrintWriter(new File("logging/sparksql.out"))

    val sc = new SparkContext(conf)
    val sql = SparkQueries.importDataToRDD(sc)

    logger.write("Sending Query " + System.currentTimeMillis() + "\n")
    SparkQueries.getUserLocations(sql, sc)
    System.exit(0)
  }
}
```

As in the previous example, you create a `SparkConfiguration` object and give the application a name. You're also adding the master node of the Spark deployment (in this instance `local` as it's on the local machine).

You've enabled the logging as well and given the name of the directory and file to write the event log to. You could extend the configuration and give the Spark scheduler a priority schedule for running the application in favor of others in the queue, but that's not a concern right now. You will also see the object `SparkQueries` being called to import the data to the RDD and run another method called `getUserLocations` to run a SQL query.

Building the Queries

I suggest that you create a separate object for the queries to keep everything clean in terms of code. There's one object to run the job and one for the worker, very much like the Java MapReduce jobs.

As I said at the start of the section, you're going to use the RDD to hold the data. This example creates a case class to define the schema object. If you think of a POJO, it's very similar in definition.

```
case class Customer(guid: String, firstname: String, lastname: String,
postcode: String, dob: String, latitude: Double, longitude: Double)
```

You now need a way of importing the data from the `.csv` file into the Customer object. Previously in this chapter, you used the `sc.textFile()` method to pull in the data and store it as an RDD as an array datatype. With the new Customer object, you can map the values from each row of the `.csv` into the object:

```
def importDataToRDD(sc: SparkContext) : SQLContext = {
    val sqlContext = new SQLContext(sc)
    val customers = sc.textFile("customers.csv").map(_.split(",")).
map(c=> Customer(c(0), c(1), c(2), c(3), c(4), c(5), c(6).toDouble,
```

```
c(7).toDouble))
    customers.registerAsTable("customers")
    cacheTable("customers")
    sqlContext
  }
```

All the customer data is now in memory, so you can create a query to look at the data. The SparkSQL dialect is very similar to standard ANSI SQL. In this example, I create separate functions for each of my queries, mainly to keep the code readable and clean, and I pass in the Spark context and SQL context in the parameters:

```
def getUserLocations(sqlContext: SQLContext, sc: SparkContext) {
    val query = "SELECT lastname, firstname, latitude, longitude FROM
customers"
    sc.setJobDescription(query)
    val result = sqlContext.sql(cmd).collect.foreach(println)
  }
```

The full object code looks like this:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

import org.apache.spark.sql.SQLContext

case class Customer(guid: String,
    firstname: String,
    lastname: String,
    postcode: String,
    dob: String,
    latitude: Double,
    longitude: Double)

//This code just copied from shark context testing
object SparkQueries {

  def importDataToRDD(sc: SparkContext) : SQLContext = {
    val sqlContext = new SQLContext(sc)
    import sqlContext._
    val customers = sc.textFile("customers.csv").map(_.split(",")).
map(c=> Customer(c(0), c(1), c(2), c(3), c(4), c(5).toDouble, c(6).
toDouble))
    customers.registerAsTable("customers")
    cacheTable("customers")
    sqlContext
  }

  def getUserLocations(sqlContext: SQLContext, sc: SparkContext) {
```

```
    val query = "SELECT lastname, firstname, latitude, longitude FROM
customers"
    sc.setJobDescription(query)
    val result = sqlContext.sql(query).collect.foreach(println)
  }
}
```

With the two object files complete, you can build the project:

```
Jason-Bells-MacBook-Pro:SparkSQLExample$ sbt package
```

```
[info] Set current project to SparkSQLExample (in build file:/Users/
Jason/work/scala/SparkSQLExample/)
[info] Compiling 2 Scala sources to /Users/Jason/work/scala/
SparkSQLExample/target/scala-2.10/classes...
[info] Packaging /Users/Jason/work/scala/SparkSQLExample/target/scala-
2.10/sparksqlexample_2.10-1.0.jar ...
[info] Done packaging.
[success] Total time: 9 s, completed 12-Jul-2014 14:25:05
```

Running the Project

Put the required `jar` file and the `.csv` data in a directory called `sparksqldemo`:

```
jason@cloudatics:~/sparksqldemo$ ls -l
total 292
-rw------- 1 jason jason 269781 Jul 12 16:51 customers.csv
drwxr-xr-x 2 jason jason   4096 Jul 12 16:47 logging
drwxr-xr-x 3 jason jason   4096 Jul 12 16:47 myeventlog
-rw-r--r-- 1 jason jason  12673 Jul 12 16:50 sparksqlexample_2.10-
1.0.jar
jason@cloudatics:~/sparksqldemo$
```

To run the program under Spark, use the `spark-submit` program:

```
/usr/local/spark/bin/spark-submit --class "SparkSQLExample" --master
local[4] sparksqlexample_2.10-1.0.jar
```

There's a lot of console output when Spark runs the program, but there are some interesting things to look out for. First of all, you see SparkSQL importing the data into memory:

```
14/07/12 17:00:44 INFO StringColumnBuilder: Compressor for [guid]: org.
apache.spark.sql.columnar.compression.PassThrough$Encoder@2ce10a23,
ratio: 1.0
14/07/12 17:00:44 INFO StringColumnBuilder: Compressor for [firstname]:
org.apache.spark.sql.columnar.compression.PassThrough$Encoder@6a6096d9,
```

```
ratio: 1.0
14/07/12 17:00:44 INFO StringColumnBuilder: Compressor for [lastname]:
org.apache.spark.sql.columnar.compression.PassThrough$Encoder@6b4fb71e,
ratio: 1.0
14/07/12 17:00:44 INFO StringColumnBuilder: Compressor for [postcode]:
org.apache.spark.sql.columnar.compression.PassThrough$Encoder@3e7f499c,
ratio: 1.0
14/07/12 17:00:44 INFO StringColumnBuilder: Compressor for [dob]: org.
apache.spark.sql.columnar.compression.PassThrough$Encoder@16b9c0d2,
ratio: 1.0
14/07/12 17:00:44 INFO FloatColumnBuilder: Compressor for [latitude]:
org.apache.spark.sql.columnar.compression.PassThrough$Encoder@7d2226a5,
ratio: 1.0
14/07/12 17:00:44 INFO FloatColumnBuilder: Compressor for [longitude]:
org.apache.spark.sql.columnar.compression.PassThrough$Encoder@4d175ad1,
ratio: 1.0
```

The query plan then shows the query that's running. The first line tells you the fields that are being returned. (In this example, they are `lastname`, `firstname`, `latitude`, and `longitude`).

```
== Query Plan ==
Project [lastname#2:2,firstname#1:1,latitude#5:5,longitude#6:6]
 InMemoryColumnarTableScan
[guid#0,firstname#1,lastname#2,postcode#3,dob#4,latitude#5,longitude#6],
(ExistingRdd
[guid#0,firstname#1,lastname#2,postcode#3,dob#4,latitude#5,longitude#6],
MapPartitionsRDD[4] at mapPartitions at basicOperators.scala:174),
false)
14/07/12 17:00:44 INFO TaskSchedulerImpl: Adding task set 1.0 with 1
tasks
14/07/12 17:00:44 INFO TaskSetManager: Starting task 1.0:0 as TID 1 on
executor localhost: localhost (PROCESS_LOCAL)
14/07/12 17:00:44 INFO TaskSetManager: Serialized task 1.0:0 as 2883
bytes in 0 ms
14/07/12 17:00:44 INFO Executor: Running task ID 1
14/07/12 17:00:44 INFO BlockManager: Found block broadcast_0 locally
14/07/12 17:00:44 INFO BlockManager: Found block rdd_7_0 locally
14/07/12 17:00:44 INFO Executor: Serialized size of result for 1 is
162709
14/07/12 17:00:44 INFO Executor: Sending result for 1 directly to driver
14/07/12 17:00:44 INFO Executor: Finished task ID 1
14/07/12 17:00:44 INFO TaskSetManager: Finished TID 1 in 543 ms on
localhost (progress: 1/1)
14/07/12 17:00:44 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose
tasks have all completed, from pool
14/07/12 17:00:44 INFO DAGScheduler: Completed ResultTask(1, 0)
14/07/12 17:00:44 INFO DAGScheduler: Stage 1 (collect at SparkQueries.
scala:30) finished in 0.808 s
```

```
14/07/12 17:00:44 INFO SparkContext: Job finished: collect at
SparkQueries.scala:30, took 0.849306003 s
[Thorpe,Maya,52.541897,1.624524]
[Cox,Ethan,52.67349,-2.078559]
[Glover,Imogen,52.47305,-0.963058]
[Smith,Sarah,50.54306,-3.720899]
[Reid,Christopher,50.855225,-0.599943]
[Iqbal,Oliver,50.808296,-2.644681]
[Little,Evie,52.01529,-1.95964]
[Flynn,Finley,51.381012,-3.258463]
```

Using Object-Based Queries

You're not just restricted to SQL-based queries. If you are used to object mapping, you might be more interested in object-based queries. Assuming that I have the customer objects in the RDD called Customers, I could easily query all the firstnames with "George" as follows:

```
val georges = customers.where('firstname="George").select('lastname)
```

A Java SparkSQL Example

As you can imagine, the Java version of the code is similar to the Scala version. Using a POJO for the customer class, you import the data into the RDD:

```java
import java.io.Serializable;
import java.util.List;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.Function;

import org.apache.spark.sql.api.java.JavaSQLContext;
import org.apache.spark.sql.api.java.JavaSchemaRDD;
import org.apache.spark.sql.api.java.Row;

public class JavaSQLExample {
    public static class Customer implements Serializable {
        private String guid;
        private String firstname;
        private String lastname;
        private String postcode;
        private String dob;
        private Float latitude;
        private Float longitude;
```

```java
public String getGuid() {
    return guid;
}

public void setGuid(String guid) {
    this.guid = guid;
}

public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}

public String getPostcode() {
    return postcode;
}

public void setPostcode(String postcode) {
    this.postcode = postcode;
}

public String getDob() {
    return dob;
}

public void setDob(String dob) {
    this.dob = dob;
}

public Float getLatitude() {
    return latitude;
}

public void setLatitude(Float latitude) {
    this.latitude = latitude;
}

public Float getLongitude() {
    return longitude;
```

```
        }

        public void setLongitude(Float longitude) {
            this.longitude = longitude;
        }
    }

    public static void main(String[] args) throws Exception {
        SparkConf sparkConf = new SparkConf().
setAppName("JavaSQLExample");
        JavaSparkContext ctx = new JavaSparkContext(sparkConf);
        JavaSQLContext sqlCtx = new JavaSQLContext(ctx);

        System.out.println("Load csv file into RDD");

        JavaRDD<Customer> customers = ctx.textFile("customers.csv").map(
          new Function<String, Customer>() {
            public Customer call(String line) throws Exception {
              String[] split = line.split(",");

              Customer customer = new Customer();
                customer.setGuid(split[0]);
                customer.setFirstname(split[1]);
                customer.setLastname(split[2]);
                customer.setPostcode(split[3]);
                customer.setDob(split[4]);
                customer.setLatitude(Float.parseFloat(split[5]));
                customer.setLongitude(Float.parseFloat(split[6]));

              return customer;
            }
          });

        JavaSchemaRDD customerSchema = sqlCtx.applySchema(customers,
Customer.class);
        customerSchema.registerAsTable("customers");

        JavaSchemaRDD georges = sqlCtx.sql("SELECT lastname FROM
customers WHERE firstname='George'");

        List<String> georgeList = georges.map(new Function<Row,
String>() {
          public String call(Row row) {
            return "Lastname: " + row.getString(0);
          }
        }).collect();

        for (String lastname: georgeList) {
```

```
            System.out.println(lastname);
        }
    }
}
```

To run the example, just run Maven to package up the `jar` file and then run with the `spark-submit` tool.

```
$ /usr/local/spark/bin/spark-submit --class "JavaSQLExample" --master
local[4] javasqlexample-1.0.jar
```

Spark generates a lot of output while it starts up and imports the `.csv` file into memory. Finally the query runs and you see the results output to the console:

```
14/08/25 23:03:19 INFO SparkContext: Job finished: collect at
JavaSQLExample.java:110, took 1.664090864 s
Lastname: Dawson
Lastname: Power
Lastname: Stewart
Lastname: Stewart
Lastname: Young
Lastname: Cartwright
Lastname: Carr
Lastname: Chandler
Lastname: Begum
```

Wrapping Up SparkSQL

If you, or any of your team, have good knowledge of SQL, then it makes sense to look at SparkSQL, especially if you are trying to extract data from the RDDs. Although there are limitations of the SparkSQL language range, it covers the needs of most people in terms of SELECT and JOINS and so on.

Spark Streaming

Chapter 9 discusses using Spring XD as a mechanism for ingesting data from various sources. Spark has a streaming ingestion engine, too, by way of Spark Streaming.

Basic Concepts

Spark Streaming can ingest data from a range of sources, such as ZeroMQ, Kafka, Flume, Twitter (more on that in a moment), and raw TCP sockets. As with

Spring XD, after data has entered the system, you have the option to process and manipulate the data coming in and then store it to an outbound location.

Spark Streaming divides data into batches for processing, rather than handling one piece of data at a time as Spring XD does. Then Spark Streaming processes and hands those batches to the requested output. Spark calls them "micro batches."

Try creating a really basic example of a stream. You can use the raw TCP socket to emit some data and Spark Streaming to ingest it. Spark Streaming uses the concept of a DStream—a discretized stream—which is a continuous stream of data coming in for processing.

Creating Your First Stream with Scala

`StreamingContext` replaces the standard `SparkContext` as the main entry point. The code listing is pretty simple; it listens to the raw socket on port 9898 on localhost and then does a quick word count on the data coming in.

```
import org.apache.spark.SparkConf
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel

object TCPIngester {
  def main(args: Array[String]) {

    val sparkConf = new SparkConf().setAppName("TCPIngester")
    val ssc = new StreamingContext(sparkConf, Seconds(1))

    val incomingLines = ssc.socketTextStream("localhost", 9898,
StorageLevel.MEMORY_AND_DISK_SER)
    val words = lines.flatMap(_.split(" "))
    val counts = words.map(w => (w, 1)).reduceByKey(_ + _)
    counts.print()

    ssc.start()
    ssc.awaitTermination()
  }
}
```

The code is very similar to the previous word count examples in this chapter. `StreamingContext` takes the configuration and also defines the amount of time to wait between processing the batches of data. In this example, you've asked for one second. The last two lines, though, are new, and they are required to ensure that Spark Streaming runs and monitors the stream.

The Build File

Use the following build file to ensure that the streaming example compiles and packages up properly:

```
name := "StreamingExample"

version := "1.0"

scalaVersion := "2.10.4"

libraryDependencies += "org.apache.spark" %% "spark-core" % "1.0.0"

libraryDependencies += "org.apache.spark" % "spark-streaming_2.10" %
"1.0.0"

resolvers += "Akka Repository" at "http://repo.akka.io/releases/"
```

Then use the sbt tool to build the project. It might take a few minutes for the streaming libraries and its dependencies to download.

```
Jason-Bells-MacBook-Pro:SparkStreaming Jason$ sbt package
[info] Set current project to StreamingExample (in build file:/Users/
Jason/work/scala/SparkStreaming/)
[info] Updating {file:/Users/Jason/work/scala/SparkStreaming/}
sparkstreaming...
[info] Resolving org.fusesource.jansi#jansi;1.4 ...
[info] Done updating.
[info] Packaging /Users/Jason/work/scala/SparkStreaming/target/scala-
2.10/streamingexample_2.10-1.0.jar ...
[info] Done packaging.
[success] Total time: 9 s, completed 13-Jul-2014 10:48:58
```

With everything compiled and packaged up you can test the project.

Testing the Project

There are two parts to this test. One is the Spark Streaming program you've just created, and the second is finding some data to stream in.

The example uses netcat to send the data out and uses the same port number (9898), to which the project will be listening. Start netcat first; otherwise, the Spark Streaming program will fail to bind to the host and port number and then throw an exception.

It's best to have two terminal windows open to perform this test. In a terminal window, start up netcat:

```
nc -lp 10000
```

There'll be no output to say it's started, it will just sit there waiting for input. In the other terminal window, you need to start the Spark Streaming project. Ensure that you have two local nodes running (as denoted by `local[2]` in the command line); otherwise, you won't get the reduced output:

```
/usr/local/spark/bin/spark-submit --class "TCPIngester" --master
local[2] streamingexample_2.-1.0.jar
```

It's worth having the two terminal windows side by side, so you don't miss the output. Type some text into the terminal running netcat, and you see the stream ingesting the information:

```
14/07/13 15:53:42 INFO BlockManagerInfo: Added input-0-1405256022600 in
memory on cloudatics.com:55995 (size: 18.0 B, free: 297.0 MB)
```

You also see the reduced output:

```
-----------------------------------------
Time: 1405258766000 ms
-----------------------------------------
(this,1)
(is,1)
(another,1)
(a,2)
(and,2)
(wibble,3)
```

Saving the Output

It's easy to extend this program further, adding the `saveAsTextFiles` method:

```
counts.saveAsTextFiles("mystream_",".txt")
```

After you've rebuilt the package with `sbt` and run the project again, you see the output written every second as a directory with an output file. With the stream processor running every second, it's sometimes hard to see what file the output is in, but using the `find` command can easily solve that.

```
jason@cloudatics:~/streamtestout$ ls -l
total 108
drwxr-xr-x 2 jason jason 4096 Jul 13 16:48 mystream_-1405259285000..txt
drwxr-xr-x 2 jason jason 4096 Jul 13 16:48 mystream_-1405259286000..txt
```

```
drwxr-xr-x 2 jason jason 4096 Jul 13 16:48 mystream_-1405259287000...txt
drwxr-xr-x 2 jason jason 4096 Jul 13 16:48 mystream_-1405259288000...txt
.
.
. (and so on ....)
.
drwxr-xr-x 2 jason jason 4096 Jul 13 16:48 mystream_-1405259309000...txt
drwxr-xr-x 2 jason jason 4096 Jul 13 16:48 mystream_-1405259310000...txt
drwxr-xr-x 2 jason jason 4096 Jul 13 16:48 mystream_-1405259311000...txt

jason@cloudatics:~/streamtestout$ find . -type f -exec grep "ipsum" {}
\; -print
(ipsum.,1)
(ipsum,1)
./mystream_-1405259294000...txt/part-00000
```

Creating Your First Stream with Java

The Java example works in the same way logically as the Scala example:

```java
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.function.FlatMapFunction;
import org.apache.spark.api.java.function.Function2;
import org.apache.spark.api.java.function.PairFunction;
import org.apache.spark.api.java.StorageLevels;
import org.apache.spark.streaming.Duration;
import org.apache.spark.streaming.api.java.JavaDStream;
import org.apache.spark.streaming.api.java.JavaPairDStream;
import org.apache.spark.streaming.api.java.JavaReceiverInputDStream;
import org.apache.spark.streaming.api.java.JavaStreamingContext;
import scala.Tuple2;
import com.google.common.collect.Lists;

import java.util.regex.Pattern;

public final class JavaStreaming {
  private static final Pattern SPACE = Pattern.compile(" ");

  public static void main(String[] args) {

    SparkConf sparkConf = new SparkConf().setAppName("JavaStreaming");
    JavaStreamingContext jsc = new JavaStreamingContext(sparkConf, new
Duration(1000));

    JavaReceiverInputDStream<String> incomingLines = ssc.
socketTextStream(
```

```
            "127.0.0.1", 10000, StorageLevels.MEMORY_AND_DISK_SER);

    JavaDStream<String> words = incomingLines.flatMap(new
FlatMapFunction<String, String>() {
      @Override
      public Iterable<String> call(String incomingLine) {
        return Lists.newArrayList(SPACE.split(incomingLine));
      }
    });

    JavaPairDStream<String, Integer> counts = words.mapToPair(
      new PairFunction<String, String, Integer>() {
        @Override
        public Tuple2<String, Integer> call(String s) {
          return new Tuple2<String, Integer>(s, 1);
        }
      }).reduceByKey(new Function2<Integer, Integer, Integer>() {
        @Override
        public Integer call(Integer i1, Integer i2) {
          return i1 + i2;
        }
      });

    counts.print();
    jsc.start();
    jsc.awaitTermination();
  }
}
```

The pom.xml Maven build file has the dependencies for the Spark Streaming libraries and Google Guava libraries:

```
<project>
  <groupId>com.mlbook</groupId>
  <artifactId>javastreaming</artifactId>
  <modelVersion>4.0.0</modelVersion>
  <name>Java Streaming Example</name>
  <packaging>jar</packaging>
  <version>1.0</version>
  <repositories>
    <repository>
      <id>Akka repository</id>
      <url>http://repo.akka.io/releases</url>
    </repository>
  </repositories>
  <dependencies>
    <dependency>
      <groupId>org.apache.spark</groupId>
```

```
        <artifactId>spark-core_2.10</artifactId>
        <version>1.0.0</version>
    </dependency>
      <dependency>
          <groupId>org.apache.spark</groupId>
          <artifactId>spark-streaming_2.10</artifactId>
          <version>1.0.0</version>
      </dependency>
    <dependency>
        <groupId>com.google.guava</groupId>
        <artifactId>guava</artifactId>
        <version>17.0</version>
    </dependency>
  </dependencies>
</project>
```

Running and testing the project is the same as the Scala version previously covered; all you need to do is run netcat and then ensure you're using the `jar` file that Maven has created with the following command:

```
/usr/local/spark/bin/spark-submit --class "JavaStreaming" --master
local[2] javastreaming-1.0.jar
```

MLib: The Machine Learning Library

The Spark Machine Learning libraries provide Spark with some of the concepts covered earlier in the book with Weka. MLib provides the following,

- Support Vector Machines
- Linear Least Squares
- Decision Trees
- Naïve Bayes
- K-means Clustering

Dependencies

This is where things get a little involved. MLib is dependent on the ScalaNLP library called Breeze, which is dependent on netlib-java and jbias. If you're just going to want the use of linear calculations, then it might be a better idea to use other existing libraries, such as the Weka framework or the Apache Commons Math libraries.

Netlib-java requires low-level libraries to be installed. I'm going to concentrate on Linux-based systems (Debian and Ubuntu especially).

```
sudo apt-get install libatlas3gf-base
```

This installs the libfortran3 library as well. The netlib-java library is a required dependency to be added to the build file of any MLib project you are working on; it's not included in the Spark system.

```
"com.github.fommil.netlib" % "all" % "1.1.1" pomOnly()
```

The next two sections cover a couple of examples.

Decision Trees

Chapter 3 covers decision trees and uses Weka to produce a system to decide the best place to put a CD based on previous sales patterns. The walkthrough of the theory examines the notion of entropy-based classification.

Try walking through the code block by block. First, you need to import some data from a file (or a datastore) and convert it into vector classes. Be careful, as the vectors in MLib are different than the Java and Scala vectors. If you run into problems, make sure that you are importing the right ones.

```
val inputData = sc.textFile("/home/jason/mlibtestdata.csv")
val mldata = data.map { line =>
  val parts = line.split(',').map(_.toDouble)
  LabeledPoint(parts(0), Vectors.dense(parts.tail))
}
```

Then run the algorithm to determine the decision tree:

```
val model = DecisionTree.train(mldata, Classification, Entropy, 5)
```

The `.train` method takes the loaded data, the algorithm (in this case `Classification`), an impurity type (`Entropy`, but you could have chosen `Gini`), and the maximum depth of the tree.

You can evaluate the classification and look at the predictions the algorithm has calculated. Finally, you output the training error rate (if there is one):

```
val labelAndPreds = mldata.map { point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
}
val trainErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble /
parsedData.count
println("Training Error = " + trainErr)
```

If you extend this basic code further, you could store the final model so it can be reused.

Clustering

The k-means clustering algorithm is very useful to construct within Spark MLib. It's so easy that it could be run in the Spark shell to test it.

Like the decision tree example, you load in some data and then define the algorithm you're going to use:

```
val inputData = sc.textFile("/home/jason/mlibkmeans/trainingdata.txt")
val parsedData = inputData.map(s => Vectors.dense(s.split(' ').map(_.
toDouble)))
```

In this case, the k-means object trains the data, and you need to define the number of clusters and the number of iterations you want the algorithm to run:

```
val numberOfClusters = 4
val numberOfIterations = 40
val clusters = KMeans.train(parsedData, numberOfClusters,
numberOfIterations)
```

You can finally show the "Within Set Sum of Squared Errors" based on the trained k-means algorithm:

```
val WSSSE = clusters.computeCost(parsedData)
println("WSSSE = " + WSSSE)
```

Summary

This chapter is a whistle-stop tour of the core Spark system and a number of the projects that work along with it. At the time of writing, Spark had just hit version 1.0, but it's still considered cutting edge in terms of the landscape of machine learning and data tools.

That hasn't stopped folks from calling it the Hadoop replacement, but at the end of the day it comes down to finding the tools that work for you, not against you. If you have a good resource of Scala developers, then perhaps Spark is a good fit.

One thing to keep in mind with these sorts of cutting-edge projects is that the codebase is liable to change from version to version, and it's worth keeping on top of the mailing list discussions to see how the landscape is emerging.

Machine Learning with R

When you're in a room of data scientists, statisticians, and math types, you'll hear one letter crop up again and again: the letter *R*. R is a programming language, and it's basically command-line driven. If you used the Spark shell in Chapter 11, then you're already familiar with the shell concept; R is the same. In addition to being used in the command-line shell, R can be written in code form and run.

Why am I telling you all this? Well, on top of the programming skills that get mentioned, you might also be asked, "Do you do R?" After this chapter, you'll hopefully have a starting point to reply, "Yes!"

Installing R

The R language comes ready to use for a number of operating systems. The download page at `http://www.r-project.org` has a number of mirror sites, so pick a mirror that's closest to you. From the mirror, choose the download for your operating system.

Mac OSX

The current version of R (3.1.1 at time of writing) comes in two separate download types: one for users running Snow Leopard and the other for Mavericks.

315

The latter is built on XCode5 compiler binaries. Download the file and open it to install. It installs the R binaries to the `/Applications` folder.

Windows

The `.exe` download for Windows provides binaries for running on 32- or 64-bit machines. The base package download will provide you with everything you need to get started.

Linux

Binary downloads are available for Debian, Ubuntu, Red Hat, and SUSE Linux distributions. If you want to save some time (and effort) and you're running Debian or Ubuntu, then you can use `apt-get` to install the r-base and r-base-dev packages. Ensure that the repository package base is up-to-date first. For users of the RedHat family of distributions use the command `sudo yum install R`.

Your First Run

When you run R, you're presented with the basic R shell, as shown in Figure 12-1. This is the main place where the work is done. It's sparse, but it does the job fine.

Use this to
quit the shell
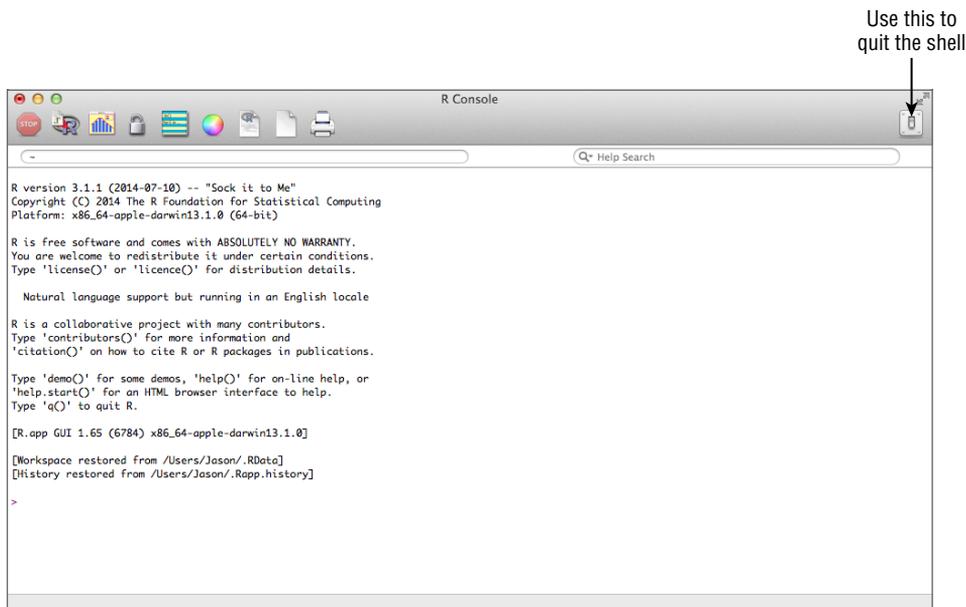
Figure 12-1: The R Shell

If at any time you want help on a topic, you can use the `help` command. For example, if you want to know about Standard Deviation, just type **`help(sd),`** and R opens a new window with the information (see Figure 12-2).
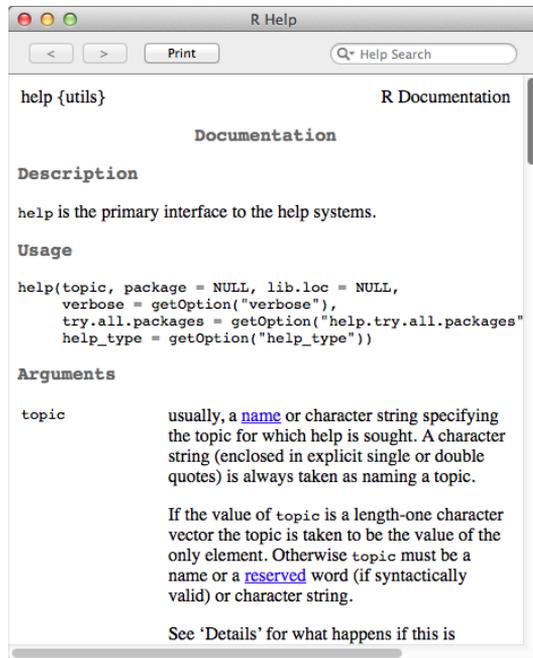
Figure 12-2: R's help system

You can quit the shell by either clicking the light switch on the top right of the program window (refer to Figure 12-1) or by typing **quit()** on the command line.

For basic needs, the R shell is fine and does the job well. For an actual development environment, you have to install some more software such as R-Studio.

Installing R-Studio

The R-Studio project (see Figure 12-3) is a commercial integrated development environment (IDE) for R. It comes in an open source community edition that is free to use. To download R-Studio IDE, visit `http://www.rstudio.com/ products/rstudio/download` and select your operating system type. Make sure that the R base binary is installed as described in the preceding section before you download R-Studio.
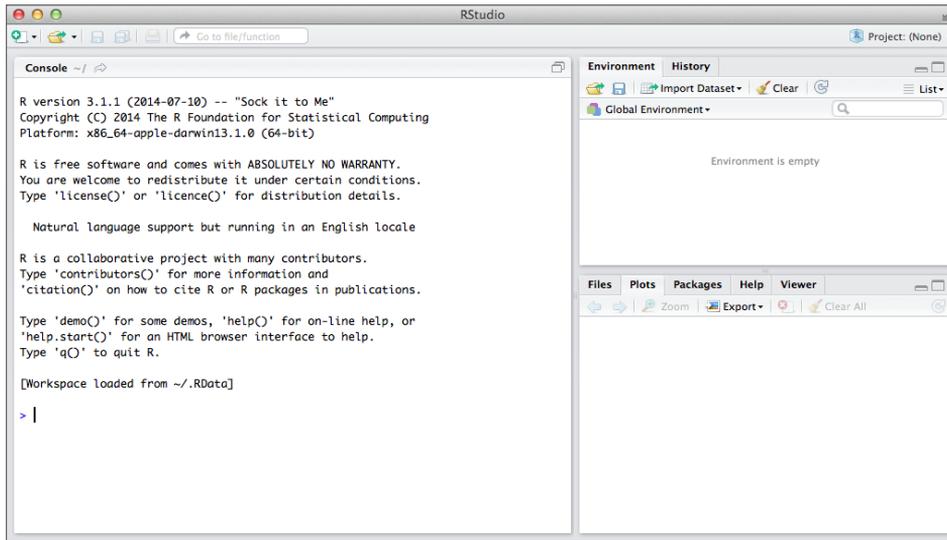
Figure 12-3: R-Studio

The R Basics

To run through the R basics, I'm going to use the standard R development environment. The command-line prompt is a simple greater than sign (>).

You can perform calculations on the command line, so adding numbers together is a trivial process, like so:

```
> 1+2
[1] 3
>
```

To get proper use from R, though, you need to think a little more programmatically.

Variables and Vectors

R supports variables as you would expect. To assign them, you can either use the equal sign (=) or the less than sign and a hyphen together (<-):

```
> myage = 21
> myageagain <- 21
> myage
[1] 21
```

```
> myageagain
[1] 21
>
```

Variables can also store string variables and other data types. The one you'll use most are numeric values.

Lists of data are held in arrays, called *vectors* in R, and are defined with the `c()` function.

```
> lotterynums <- c(2,7,20,35,36,42)
> lotterynums
[1]  2  7 20 35 36 42
```

Vectors can also hold strings. Using the `length()` function tells you how many elements are in the array.

```
> kc <- c("Robert", "Adrian", "Tony", "Bill", "Pat", "Trey")
> kc
[1] "Robert" "Adrian" "Tony"   "Bill"   "Pat"    "Trey"
> length(kc)
[1] 6
>
```

To show specific values in the array, you can use the variable name and the element you want to show.

```
> kc[5]
[1] "Pat"
```

Matrices

Now that you know how vector lists of numbers work, you can convert them into a matrix. To define a matrix, you take the data and then define how many rows and columns you require.

```
> mymatrix <- matrix(c(1,2,3,4,5,6,7,8,9,10), nrow=2, ncol=5,
byrow=TRUE)
> mymatrix
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
>
```

You can then retrieve data based on the row and column position.

```
> # by row, col
> mymatrix[2,4]
[1] 9
```

```
> # entire row
> mymatrix[2,]
[1]  6  7  8  9 10
> # entire col
> mymatrix[,4]
[1] 4 9
>
```

Instead of numeric row and column names, you can define text label names to make things more readable.

```
> dimnames(mymatrix) <- list(c("row1","row2"),c("c1","c2","c3","c4"
,"c5"))
> mymatrix
      c1 c2 c3 c4 c5
row1  1  2  3  4  5
row2  6  7  8  9 10
>
```

You can reference data by row and column by using the row and column names you've just defined.

```
> mymatrix["row2", "c5"]
[1] 10
```

Lists

A *list* is a vector containing other objects. This can be a mixture of objects (numeric, Boolean, and strings, for example) or other vectors within the list.

```
> nums <- c(1,2,3,4,5)
> strings <- c("hello", "world", "again")
> bools <- (TRUE, FALSE)
Error: unexpected ',' in "bools <- (TRUE,"
> bools <- c(TRUE, FALSE)
> mylist <- list(bools, strings, nums)
> mylist
[[1]]
[1]  TRUE FALSE

[[2]]
[1] "hello" "world" "again"

[[3]]
[1] 1 2 3 4 5
```

To retrieve the strings on their own, you can slice the list accordingly with the [] notation.

```
> mylist[2]
[[1]]
[1] "hello" "world" "again"
```

To reference a member of the listed object directly, you have to use a double squared bracket. You can modify the member within the list as well.

```
> mylist[[2]][1]
[1] "hello"
> mylist[[2]][1] <- "goodbye"
> mylist
[[1]]
[1]  TRUE FALSE

[[2]]
[1] "goodbye" "world"   "again"

[[3]]
[1] 1 2 3 4 5

>
```

Data Frames

Data frames are basically lists of vectors. The column count is the same in the vectors. R comes with some predefined data frames to play with. Using the head() function, you can see the top few lines of the data frame. This saves the entire contents of the frame being shown in the command line.

```
> data(USArrests)
> head(USArrests)
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California     9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

You can reference data with the row and column positioning like you did with the matrices.

```
> USArrests["New York",]
         Murder Assault UrbanPop Rape
New York   11.1     254       86 26.1
> USArrests["New York", "Assault"]
[1] 254
```

Installing Packages

R comes with a comprehensive selection of packages that are available to download. You can see the Comprehensive R Archive Network (usually referred to as "CRAN") packages that are available on the R website at `www.r-project.org/`. They are a broad spectrum of statistics, data-processing, and other tools.

To install the packages, you use the `install.packages()` function from the R command line. It takes care of everything for you.

For example, to install the tools for Approximate Bayesian Computation (ABC), you install the "abc" package:

```
>install.packages("abc")
```

Some packages might require dependencies to be installed first, so it's prudent to use the `dependencies` flag to ensure they are installed, too.

```
>
also installing the dependencies 'SparseM', 'quantreg', 'locfit'

trying URL 'http://cran.rstudio.com/bin/macosx/mavericks/contrib/3.1/
SparseM_1.03.tgz'
Content type 'application/x-gzip' length 825491 bytes (806 Kb)
opened URL
==================================================
downloaded 806 Kb

trying URL 'http://cran.rstudio.com/bin/macosx/mavericks/contrib/3.1/
quantreg_5.05.tgz'
Content type 'application/x-gzip' length 1846783 bytes (1.8 Mb)
opened URL
==================================================
downloaded 1.8 Mb

trying URL 'http://cran.rstudio.com/bin/macosx/mavericks/contrib/3.1/
locfit_1.5-9.1.tgz'
Content type 'application/x-gzip' length 597404 bytes (583 Kb)
opened URL
==================================================
downloaded 583 Kb

trying URL 'http://cran.rstudio.com/bin/macosx/mavericks/contrib/3.1/
abc_2.0.tgz'
Content type 'application/x-gzip' length 5303210 bytes (5.1 Mb)
opened URL
==================================================
downloaded 5.1 Mb
```

```
The downloaded binary packages are in
/var/folders/b5/fz_57qk522nd6vqk2pd4lytr0000gn/T//RtmpOqHaEV/downloaded_
packages
```

To use the library after it's installed, you call it with the `library()` function. It initializes and gives notice of the dependencies it has also loaded.

```
> library(abc)
Loading required package: nnet
Loading required package: quantreg
Loading required package: SparseM

Attaching package: 'SparseM'

The following object is masked from 'package:base':

    backsolve

Loading required package: MASS
Loading required package: locfit
locfit 1.5-9.1 2013-03-22
```

Loading in Data

With the basic notions of variables, lists, and vectors in place, it's time to look at getting some data loaded into R.

CSV Files

The `read.csv` function reads a `.csv` file and loads it into a data frame.

```
> trans <- read.csv('vdata.csv', header=TRUE, sep=',')
> head(trans)
  wheels chassis pax vtype
1      4       2   4   Car
2      9      20  25   Bus
3      5      14  18   Bus
4      5       2   1   Car
5      9      17  25   Bus
6      1       1   1  Bike
>
```

If your `.csv` file has the column names in the first line, then use `header=TRUE`; otherwise, set it to `FALSE`. The separator is defined with the `sep` keyword, and you define whatever delimiter you want. If you have missing values, then it's wise to use the `fill` flag as well to ensure that your data will have the correct number of elements in each row.

MySQL Queries

Installing the RMySQL package gives you access to MySQL databases. You can pull queries into R so they can be processed.

```
>install.packages("RMySQL", dependencies=TRUE)
```

If you are working on a Windows-based system then the library requires building for the source files. Two environment variables are required for the library to compile:

```
> Sys.setenv(PKG_CPPFLAGS = "-I/path/to/mysql/include/dir")
> Sys.setenv(PKG_LIBS = "-L/path/to/library/dir -lmysqlclient")
> install.packages("RMySQL", type = "source")
```

As with Java code, you need to define a connection to the database before you can query it.

```
>con <- dbConnect(MySQL(), user="myuser", password="mypass",
dbname="mydb", host="localhost")
```

From there, after you have a connection, you can see what tables are in the database.

```
>dbListTables(con)
```

You then query a table. The data from the query is returned as a data frame.

```
>dta <- dbGetQuery(con, "SELECT * FROM mytable")
```

There are a number of other databases supported in R, including SQLite3, Postgresql, and Oracle.

Creating Random Sample Data

Perhaps you don't have any data to load or you just want to have a random sample of numbers to play with. Using the `sample` function, you can create a handy vector of numbers.

```
> sam <- sample.int(1000, 20, replace=TRUE)
> sam
 [1]   32 192 783 654 250 261 150 687 619 332 549 225 545 175 508 782 237
748 334 804
```

Plotting Data

R supports basic plots of your data. They can take a little amount of getting use to with regard to the syntax, so the following sections provide a short primer.

Bar Charts

How many bar charts did you draw at school? I drew far more than I care to remember, but R makes it easy for me now. (See Figure 12-4.)

```
> sam <- sample.int(1000, 20, replace=TRUE)
> sam
 [1]  32 192 783 654 250 261 150 687 619 332 549 225 545 175 508 782 237
748 334 804
> barplot(sam, main="My first plot", horiz=TRUE)
```
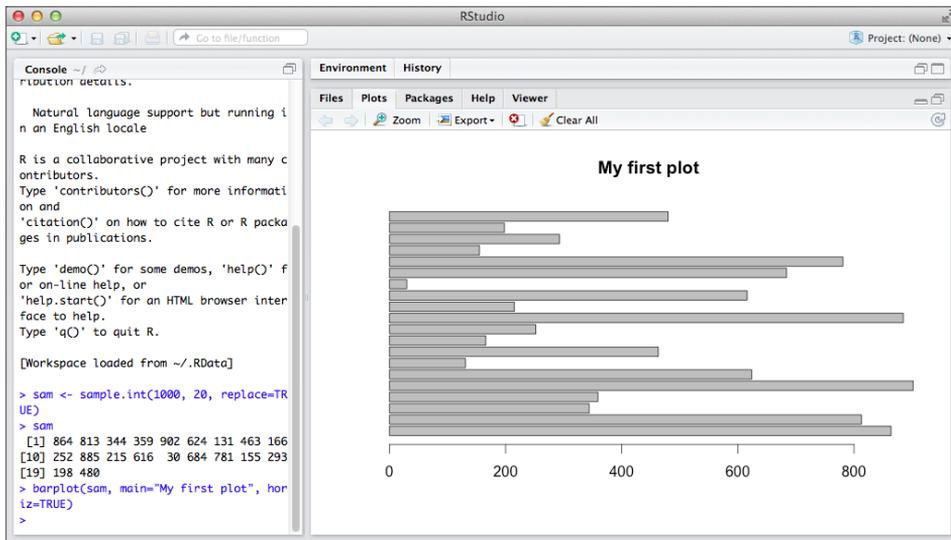

Figure 12-4: Horizontal bar chart

If you remove the `horiz` option, you get the bars travelling in a vertical direction, as shown in Figure 12-5.

```
> barplot(sam, main="My first plot")
```

Pie Charts

The pie charts in R are basic (see Figure 12-6), but they get the job done. It's just a case of giving the pie chart values and labels. You can easily expand on this if necessary.

```
> pie(sam, main="First Pie Chart", labels=sam)
```
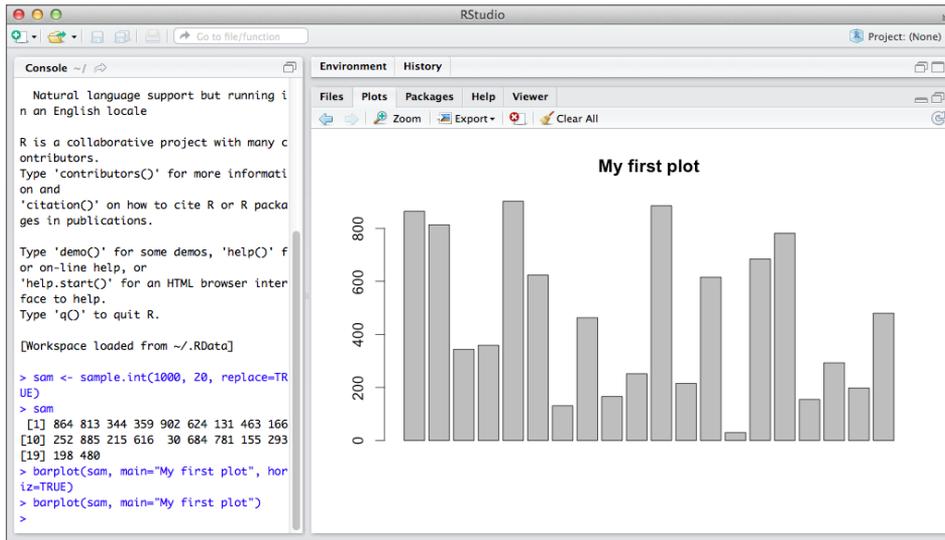
Figure 12-5: Vertical bar chart
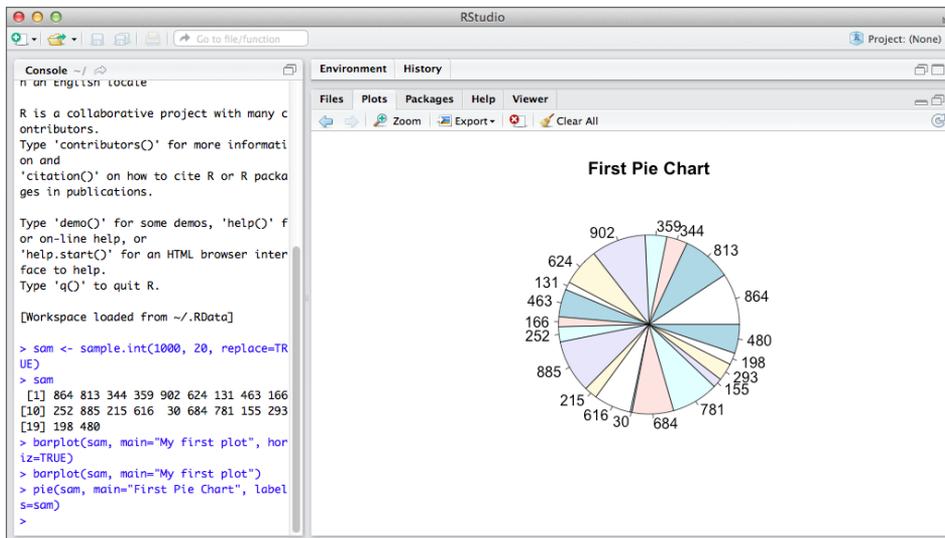
Figure 12-6: Simple pie chart

Dot Plots

The dot plot function (see Figure 12-7) is a simple case of specifying a vector. You can also group the dot plot into specific sections if required.

```
> dotchart(sam, main="My Dot Chart", labels="Value", xlab="Frequency")
```
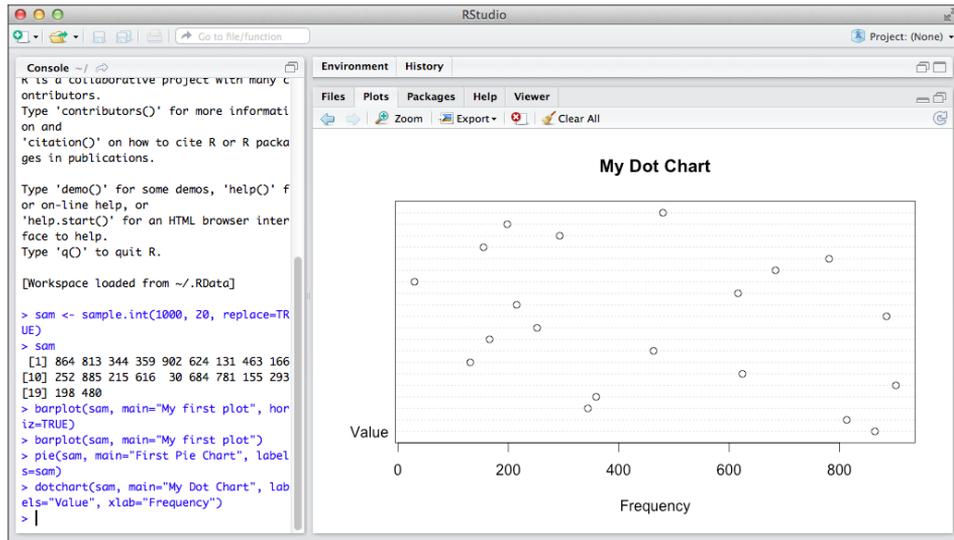
Figure 12-7: Simple dot plot

Line Charts

With two vectors of numbers, you can create a line chart, as shown in Figure 12-8.

```
> sam1 <- sample.int(10, 12, replace=TRUE)
> sam1
 [1]  5 10  4  8  2  2  2  4  2  5  2  6
> sam2 <- sam1
> plot(sam1, sam2)
> lines(sam1, sam2, type="l")
```

Simple Statistics

R is about statistics; that's what it's built for. Unlike Java, Scala, or Python, R's syntax is a little unforgiving, but after a few sessions it becomes more natural.

Try creating a simple vector of numbers, and then you can work through some functions. Start with the basics.

```
> s <- sample(100, 12, replace=TRUE)
> # get a basic summary of the vector: lowest value, 1st quartile,
median, mean, 3rd quartile and maximum value
> summary(s)
```

```
    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    1.00    9.25   28.50   37.58   58.25   97.00
> # just get the minimum
> min(s)
[1] 1
> # get the maximum value
> max(s)
[1] 97
> # get the average
> mean(s)
[1] 37.58333
> # get the median
> median(s)
> # get the standard deviation
> sd(s)
[1] 31.57807
> # use the table function to see the frequency of the data
> table(s)
s
 1   5   7 10 22 25 32 55 57 62 78 97
 1   1   1  1  1  1  1  1  1  1  1  1
```

Obviously, you can reassign these function results as new variables or vectors. This gives you the basic outline of how the summaries work.
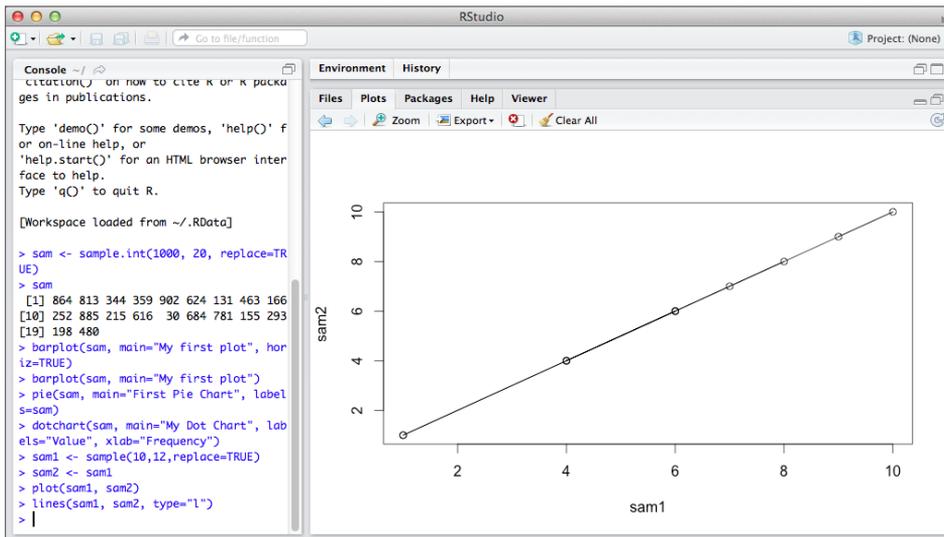
Figure 12-8: Simple line chart

Simple Linear Regression

This section gives an example of simple linear regression in R. It will give you a good idea of how things are put together. Here's the story: You have profit made based on the number of seconds that the sales team is on a call. If you know the profit made, can you calculate how long the call took?

Creating the Data

First, create two separate vectors: one for the number of seconds in the call (`secondsCall`) and another for the amount of profit that was made (`dollarProfit`).

```
> # setup the data
> secondsCall <- c(23,28,39,48,64,75,88,96,97,109,118,149,150,156,165)
> dollarProfit <- c(1,2,3,3,4,4,5,6,6,7,8,8,9,10,10)
```

The Initial Graph

You can create a simple plot for those values (see Figure 12-9) by using the `plot` command.

```
> # create a simple plot
> plot(secondsCall, dollarProfit)
```
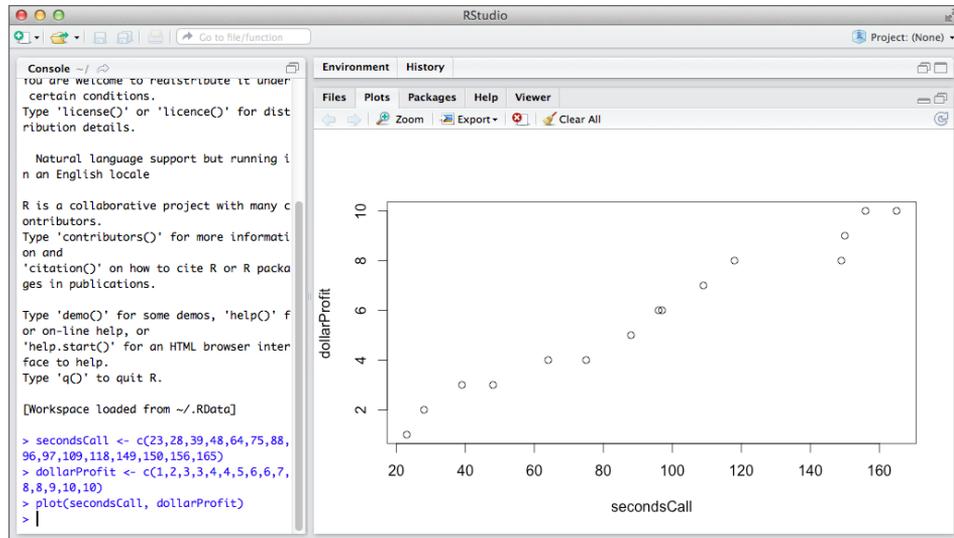

Figure 12-9: Seconds/dollar plot

Regression with the Linear Model

Within R, there is a command that will do the linear model for you: `lm`. You can define the model and save it as a variable. The order of variables is dependent (`secondsCall`), followed by a tilde symbol (`~`), and finally the independent variables (`dollarProfit`).

```
> # define the linear model
> model <- lm(secondsCall ~ dollarProfit)
> model

Call:
lm(formula = secondsCall ~ dollarProfit)

Coefficients:
 (Intercept)  dollarProfit
      0.6226       16.2286

>
```

So, you now know the intercept (0.6226) and the dollar profit amount of 16.22. You can expand on the model information using the `summary` command.

```
> # expend the summary of the model
> summary(model)

Call:
lm(formula = secondsCall ~ dollarProfit)

Residuals:
    Min      1Q  Median      3Q     Max
-12.451  -5.151  -1.308   4.734  18.549

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.6226     4.8981   0.127    0.901
dollarProfit 16.2286     0.7681  21.129 1.9e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.306 on 13 degrees of freedom
Multiple R-squared:  0.9717, Adjusted R-squared:  0.9695
F-statistic: 446.4 on 1 and 13 DF,  p-value: 1.898e-11
```

So, you have a basic model that gives you a regression equation of *secondsCall = 0.6226 + 16.2268 * profit amount*. To put the regression line on the plot, use the `abline` function.

```
> abline(model)
```

Making a Prediction

Assume that someone made a $5 profit, and you want to know the duration of the call based on the model you've just created. Using the `predict` command, you can make the prediction.

```
> # make a basic prediction of someone making $5.
> predict(model, newdata=data.frame(dollarProfit=5))
       1
81.76568
>
```

The prediction is that the person was on a call for 81 seconds. You can extend that by adding different `interval` types, which will give you the upper and lower prediction amounts based on the model.

```
> predict(model, newdata=data.frame(dollarProfit=5), interval="pred")
       fit      lwr      upr
1 81.76568 63.19372 100.3376
> predict(model, newdata=data.frame(dollarProfit=5),
interval="confidence")
       fit      lwr      upr
1 81.76568 76.97553 86.55583
```

Basic Sentiment Analysis

Chapter 9, where you ingested tweet data and showed the positive and negative scoring, covers basic sentiment analysis. The same is achievable in R with some basic coding. The text to rate could be anything from simple sentences typed in to reading in a Twitter stream or a file.

Functions to Load in Word Lists

You need two sets of text files: one with the positive words and one with the negative words. You can write two quick functions to load the text files and save them to two separate lists.

```
LoadPosWordSet<-function(){
 iu.pos = scan("positive-words.txt", what='character', comment.char=";")
 pos.words = c(iu.pos)
 return(pos.words)
}
```

Then you do the same for the negative word list:

```
LoadNegWordSet<-function(){
 iu.neg = scan("negative-words.txt", what='character', comment.char=";")
 neg.words = c(iu.neg)
 return(neg.words)
}
```

Writing a Function to Score Sentiment

You have a function that takes in a sentence and two word lists (positive and negative sentiment words). So now you can test it.

```
GetScore<-function(sentence, pos.words, neg.words) {
 sentence = gsub('[[:punct:]]', '', sentence)
 sentence = gsub('[[:cntrl:]]', '', sentence)
 sentence = gsub('\\d+', '', sentence)

 sentence = tolower(sentence)

 word.list = str_split(sentence, '\\s+')
 words = unlist(word.list)

 pos.matches = match(words, pos.words)
 neg.matches = match(words, neg.words)

 pos.matches = !is.na(pos.matches)
 neg.matches = !is.na(neg.matches)
 score = sum(pos.matches) - sum(neg.matches)

 return(score)
}
```

The first thing that happens is the sentence is cleaned up with punctuation, control characters, and numbers removed. That should give you just a sentence of words; you then convert it into all lowercase letters.

You split the sentence into a list of words and find out how many times the words match in the positive word list. You also do the same with the negative word list.

With a positive score and negative score, you take the negative away from the positive to get the final score.

You can save the functions as an R source file. You can either create it in a text editor or use the R-Studio editor to create the source file. For the purpose of this example, I save it all in a file called sentiment.r.

Testing the Function

To test the sentiment code, you first need to load the code and the required library into R:

```
>install.packages("stringr")
>library(stringr)
>source('sentiment.r')

> pos.words <- LoadPosWordSet()
Read 2006 items

> neg.words <- LoadNegWordSet()
Read 4783 items
```

So, you have 2,006 positive words and 4,783 negative words loaded. By using the `GetScore` method, you can get a score now on some text, and you can make up some to test. For example, here's a positive one:

```
> testscore<-GetScore("This concert is the best thing I've been to!",
pos.words, neg.words)
> testscore
[1] 1
```

As you can see the sentiment analysis gave a score of +1, so it's positive. Try a negative sentence:

```
> testscore2<-GetScore("That's bad real bad, horrible", pos.words, neg.
words)
> testscore2
[1] -3
```

With a negative string, you get a score of -3. With this basic function, you could process a list of sentences and create a bar graph of the scoring.

Apriori Association Rules

With a set of transactions, you can run a basic Apriori algorithm. The R base system requires a package called `arules` to be installed before use.

Installing the ARules Package

Before you get started you have to install the `arules` package:

```
> install.packages("arules", dependencies=TRUE)
also installing the dependencies 'colorspace', 'TSP', 'gclus',
'scatterplot3d', 'vcd', 'seriation', 'igraph', 'pmml', 'XML',
'arulesViz', 'testthat'
The downloaded binary packages are in /var/folders/b5/fz_57qk522nd6vqk2p
d4lytr0000gn/T//Rtmpgp3zNQ/downloaded_packages
> library(arules)
Loading required package: Matrix

Attaching package: 'arules'

The following objects are masked from 'package:base':
%in%, write

>
```

The Training Data

I have prepared a basic `.csv` file with the basket ID and one item per line. You can see that there are repeating basket IDs to show that the basket contains multiple items. I've called my file `transactions.csv`.

```
1001,Fries
1001,Coffee
1001,Milk
1002,Coffee
1002,Fries
1003,Coffee
1003,Coke
1003,Eraser
1004,Coffee
1004,Fries
1004,Cookies
1005,Milk
1006,Coffee
1006,Milk
1007,Coffee
1007,Fries
1008,Fries
1008,Coke
```

Importing the Transaction Data

With the `read.transactions` method, you load the `.csv` data into a `transactions` object. It works in a similar way to the `read.csv` function you saw at the start of the chapter.

I'm setting the `rm.duplicates` flag to `FALSE`, because I don't want basket items to be removed.

```
> transactions <- read.transactions(file="transactions.csv",
rm.duplicates=FALSE, format="single", sep=",", cols=c(1,2))
> transactions
transactions in sparse format with
 8 transactions (rows) and
 6 items (columns)
>
```

As you can see, the `transaction` object knows there are eight transactions with six items. You can see the relative frequency of items graphically by using the `itemFrequencyPlot` function, which generates a graph like the one in Figure 12-10.

```
> itemFrequencyPlot(transactions)
```


Figure 12-10: Transaction frequencies

Running the Apriori Algorithm

The function to run the algorithm is done in one line. You supply the transaction objects and a set of parameters. When you run the algorithm, you're presented with the resulting output. So, with a support of 0.5 and a confidence of 0.8 (the system is 80% confident), you get one association rule:

```
> minedbasketrules <- apriori(transactions, parameter=list(sup=0.5,
conf=0.8, target="rules"))

parameter specification:
 confidence minval smax arem  aval originalSupport support minlen maxlen
target   ext
       0.8    0.1    1 none FALSE            TRUE     0.5      1     10
rules FALSE

algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09)        (c) 1996-2004   Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[6 item(s), 8 transaction(s)] done [0.00s].
sorting and recoding items ... [2 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
writing ... [1 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
>
```

Inspecting the Results

Have a look at the one rule and see what it is. You use the `inspect` command to look at the result:

```
> inspect(minedbasketrules)
  lhs        rhs       support confidence      lift
1 {Fries} => {Coffee}     0.5         0.8 1.066667
>
```

There's an 80 percent chance that if someone buys fries, they'll also buy a coffee. If you had thousands or tens of thousands of transactions, then you would raise the confidence level and also be able to see more rules appearing.

Accessing R from Java

Like the power of the Weka workbench can be accessed from a Java program, so too can R code. With the `rJava` bridge, you can run R within Java code and Java within R code.

Installing the rJava Package

Originally the `rJava` package was split along with the JRI package; installing them was a technical process at the time. Fortunately, that has all been replaced with a single binary that covers both packages.

```
> install.packages("rJava")
trying URL 'http://cran.rstudio.com/bin/macosx/mavericks/contrib/3.1/
rJava_0.9-6.tgz'
Content type 'application/x-gzip' length 600621 bytes (586 Kb)
opened URL
=================================================
downloaded 586 Kb


The downloaded binary packages are in
/var/folders/b5/fz_57qk522nd6vqk2pd4lytr0000gn/T//Rtmpgp3zNQ/downloaded_
packages
>
```

The rJava package uses JNI to talk to Java libraries. From the point of view of working within R, things might seem a little cumbersome, but they do work fine.

Your First Java Code in R

Open your R console or R-Studio. Assuming you've installed the package as described earlier in this chapter, you can do the following:

```
> library(rJava)
> .jinit()
> stringobj <- .jnew("java/lang/String", "This is a string as a Java
object, in R!")
> stringobj
[1] "Java-Object{This is a string as a Java object, in R!}"
```

After the library is loaded, you need to initialize the rJava system with the `.jinit()` method. You then create a new variable in R that is going to contain

a Java string object. The `.jnew()` method creates a new String object and populates the string. Notice that you have to put the full Java package name in with a slashed notation and not the dotted one.

If you want to find the location of the word "Java" in the string, you use Java's `indexOf` method. You can call it with rJava by executing the following:

```
[1] "Java-Object{This is a string as a Java object, in R!}"
> .jcall(stringobj, "I", "indexOf", "Java")
[1] 22
```

The command looks involved. The first parameter is the existing object you previously created. The next is the return type `from` method; because the `indexOf` method returns an integer, you use the `"I"` in the calling method. Next is the method name—`"indexOf"`—and last is the thing you're looking for, `"Java"`. You see the result on the line underneath.

For the full package information for the rJava interface, have a look at the method list at `http://rforge.net/doc/packages/rJava/00Index.html`.

Calling R from Java Programs

The interface for calling R from Java is called JRI. The files required to do this are all in the library that was installed from R. There are two components that your Java project requires: the `jar` file (called `JRI.jar`) and the native library file (the name changes depending on the operating system you are using—on Mac OS X, it's called `libjri.jnilib`).

```
Jason-Bells-MacBook-Pro:jri Jason$ pwd
/Library/Frameworks/R.framework/Resources/library/rJava/jri
Jason-Bells-MacBook-Pro:jri Jason$ ls -l
total 256
-rw-r--r--  1 Jason  admin  31384 24 Apr 16:02 JRI.jar
-rw-r--r--  1 Jason  admin  10272 24 Apr 16:02 JRIEngine.jar
-rw-r--r--  1 Jason  admin  32354 24 Apr 16:02 REngine.jar
drwxr-xr-x  8 Jason  admin    272 24 Apr 16:02 examples
-rwxr-xr-x  1 Jason  admin  47500 24 Apr 16:02 libjri.jnilib
-rwxr-xr-x  1 Jason  admin    833 24 Apr 16:02 run
Jason-Bells-MacBook-Pro:jri Jason$
```

You'll set up a basic Eclipse project and then you can see how the parts fit together. I developed the example on the Mac OS X operating system, but the variations on the other operating systems, such as Windows or Linux, are not that different.

Setting Up an Eclipse Project

Create a Java project and call it `JRITest`. Go to the properties, click the Java Build Path, and add an external `jar` file. Now look for the `JRI.jar` file, which

is normally located in the `/Library/Frameworks/R.framework/Resources/ library/rJava/jri` folder. (See Figure 12-11.)
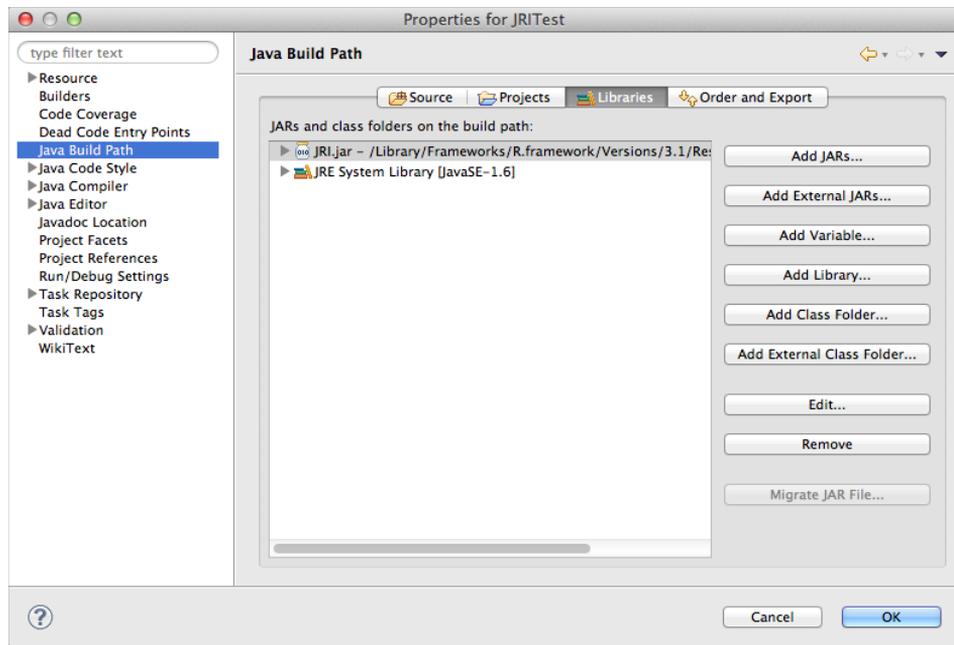
Figure 12-11: Adding the JRI.jar file to the project

To make sure the R engine is working within Java, you're going to create a small test file to initialize the engine, load the built-in `iris` dataset, and iterate through an evaluation.

Creating the Java/R Class

To create a new class, select File ➪ New ➪ Class and call the new file **TestR.java**.

```java
import java.util.Enumeration;

import org.rosuda.JRI.REXP;
import org.rosuda.JRI.RVector;
import org.rosuda.JRI.Rengine;

public class TestR {

    public static void main(String[] args) {
        Rengine rEngine = new Rengine(new String[] { "--vanilla" },
false, null);
        System.out.println("Waiting for R to create the engine.");

        if (!rEngine.waitForR()) {
```

```
            System.out.println("Cannot load R engine.");
            return;
        }

        rEngine.eval("data(iris)", false);
        REXP exp = rEngine.eval("iris");
        RVector vector = exp.asVector();
        System.out.println("Outputting data:");
        for (Enumeration e = vector.getNames().elements();
e.hasMoreElements();) {
            System.out.println(e.nextElement());
        }
    }
}
```

The first thing that happens within the `main` method is to start up an R engine. Nothing works until this step is complete.

Next, you pass an R command to load the `iris` data using the `REngine.eval` function. Last, you initialize an R vector within Java, convert the `iris` data to a vector, and then iterate the output.

Running the Example

If you attempt to run the class now, you will get an error from Eclipse, because the R runtime library isn't linked to the project. You get the following error if the library isn't linked:

```
Cannot find JRI native library!
Please make sure that the JRI native library is in a directory listed in
java.library.path.

java.lang.UnsatisfiedLinkError: no jri in java.library.path
    at java.lang.ClassLoader.loadLibrary(ClassLoader.java:1764)
    at java.lang.Runtime.loadLibrary0(Runtime.java:823)
    at java.lang.System.loadLibrary(System.java:1044)
    at org.rosuda.JRI.Rengine.<clinit>(Rengine.java:19)
    at TestR.main(TestR.java:10)
```

To set that up, you need to look at the run configurations for the project. Select Run ▷ Run Configurations and then click the `TestR` class. On the Arguments tab, you need to add a `-D` flag to the virtual machine arguments, as shown in Figure 12-12.

If you get an error about the `R_HOME` path not being set, then reopen the run configuration and click the Environments tab, as shown in Figure 12-13. If you use windows locate the `R.dll` file on your system and add it to your Path.
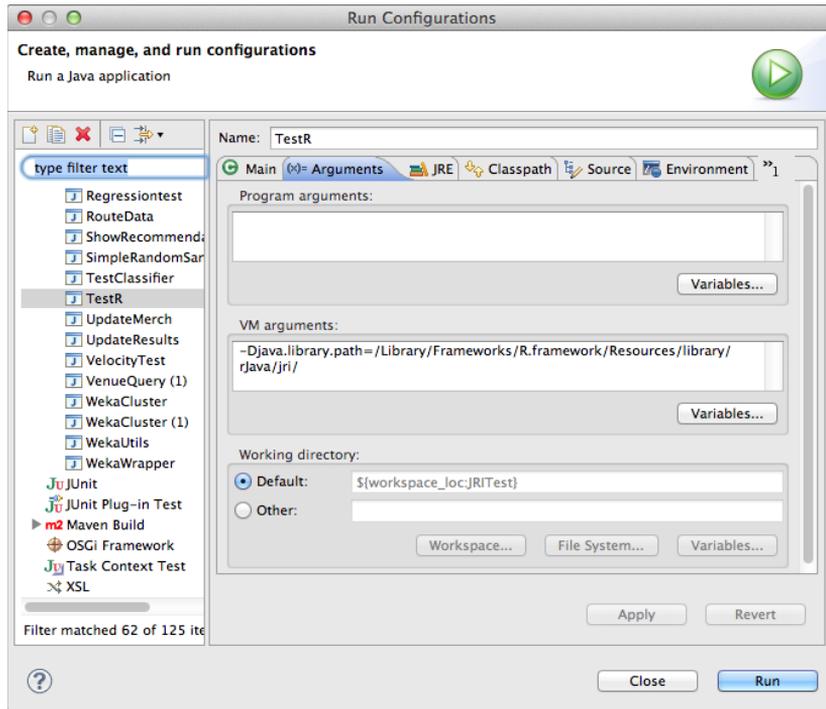
Figure 12-12: Adding the JRI library path
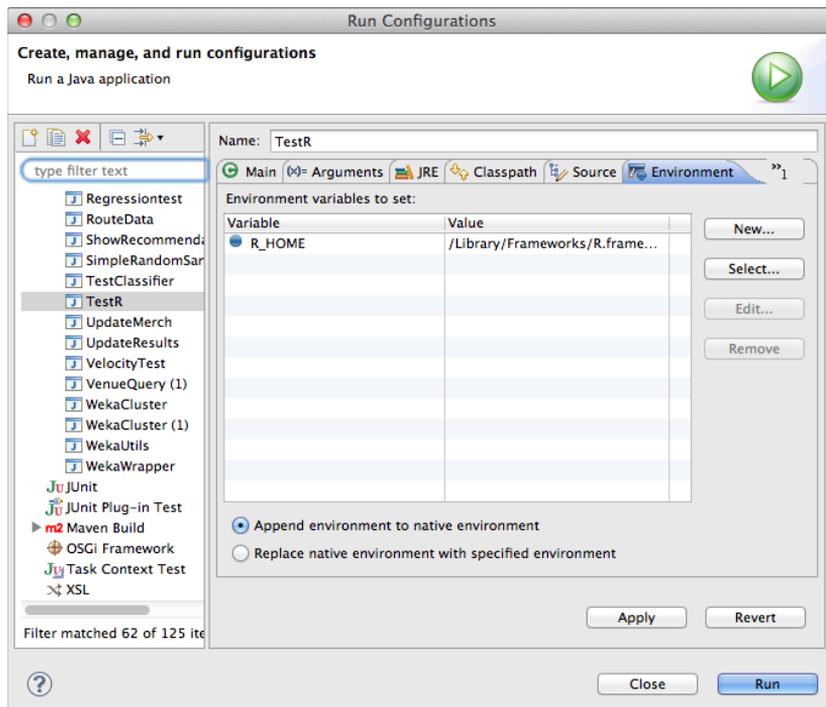
Figure 12-13: Adding the environment R_HOME path

Click Run and try again. This time you should see the correct output.

```
Waiting for R to create the engine.
Outputting names:
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width
Species
```

Extending Your R Implementations

The code you've just walked through gives you the basic framework for getting R functions and commands working from a Java program. The R examples you've seen in this chapter could be easily converted to a Java program using this method if you want. If you have an R expert on your team, then it might be prudent to have that person write the R functions first and then port them to Java.

R and Hadoop

If you are of a delicate nature, then you might want to look away from this section. It is possible to run R jobs with Hadoop. When you look back at the date when the likes of R and Weka were first developed, the notion of the MapReduce paradigm existed, but it was not on the top of the list of most data crunchers.

Over time, though, the volumes of data have increased dramatically. What hasn't progressed as much are the older tools. With large volumes of data, the memory implications make R difficult to use on a day-to-day basis. For small work, it's fine; when used for larger, more memory-intensive work, things might eventually break.

The RHadoop Project

Revolution Analytics developed the RHadoop libraries, which enable you to use Hadoop to scale R jobs. This gives you full MapReduce capabilities on one or more Hadoop nodes.

The RHadoop project requires you to have Hadoop 1.0.2 or later (Cloudera CDH3). R installs on each node that you are intending to use with a copy of the rmr2 package installed as well.

Before you can download and install the rmr2 package you are required to have the following R packages installed:

- Rcpp
- RJSONIO (>= 0.8-2)

- Bitops
- Digest
- Functional
- reshape2
- stringr
- plyr
- caTools (>= 1.16)

To install the rmr2 package, you have to manually download it from Revolution Analytics' Github repository (`https://github.com/RevolutionAnalytics/rmr2`) and run the following command from a terminal shell:

```
$ R CMD INSTALL rmr2_<the_version_number>.tar.gz . rmr2
```

The rmr2 package is not available on the CRAN package manager at present.

Finally the environment variable for HADOOP_HOME needs to be set and the location of the streaming `jar` file set for HADOOP_STREAMING.

A Sample Map Reduce Job in RHadoop

I'm going to break down a basic word count job in R using the RHadoop packages. I'm assuming you've already copied some text data into the Hadoop Distributed File System (HDFS).

The Map Phase

The first thing you do is load the rmr2 library package into memory. The map splits the incoming lines by a space character, which gives you a list of words. You're emitting a value of 1 for each word.

```
library(rmr2)

map <- function(x,inputLines) {
  words.list <- strsplit(inputLines, '\\s')
  words <- unlist(words.list)
  return( keyval(words, 1) )
}
```

The Reduce Phase

The reduce phase is a function that takes a word and its counts and reduces them to a word and a sum of the counts, This is then passed back to the calling function.

```
## reduce function
reduce <- function(word, counts) {
  keyval(word, sum(counts))
}
```

Setting Up the WordCount Job

You create a function to take in the input and output detail to the job and set up the basic map and reduce functions—the ones we've just created.

```
wordcount <- function (inputObject, outputObject=NULL) {
  mapreduce(input= inputObject, output= outputObject, input.
format="text", map=map, reduce=reduce)
}
```

Running the Job

The first thing you do is delete any references to the existing output data in HDFS. Hadoop fails if the output directory already exists.

Next, create your job with the input and output paths and then set a variable object to the output of the wordcount function you created earlier:

```
system("hadoop fs -rmr wordcount/outputdata")

hdfs.root <- 'wordcount'
hdfs.data <- file.path(hdfs.root, 'inputdata')
hdfs.out <- file.path(hdfs.root, 'outputdata')
out <- wordcount(hdfs.data, hdfs.out)
```

Checking the Results

You extract the output data from HDFS and then create a dataframe that reads in the results. Two column names are created for the top of the output: inputWord and frequency.

```
results <- from.dfs(out)

## check top 30 frequent words
results.df <- as.data.frame(results, stringsAsFactors=F)
colnames(results.df) <- c('inputWord', 'frequency')
head(results.df[order(results.df$count, decreasing=T), ], 10)
```

NOTE **Although I've concentrated on the RHadoop library, it's not the only one that exists. It's also worth checking out the RHIPE software libraries for R published by the Department of Statistics at Purdue University. You can find the website with all the details at** `www.datadr.org/install.html`**.**

Connecting to Social Media with R

I know—another mention of the Twitter application program interface (API). This sort of data is important and not just for sentiment analysis. The majority of news, social graphs, and conversations happen on these platforms, and it's increasingly important to keep on top of developments. This goes for R, too; being able to connect to Twitter is important.

In the previous examples you used Twitter's development account to create the authorization keys for the application. You're essentially forcing your application to use these pre-existing keys. That works fine, but you have to approach things a little differently for the twitteR library.

Make a note of the consumer key and secret as you'll need those, but there are couple of other things you need to confirm.

First of all, make sure there's no callback URL. You confirm this on the Settings tab of your application. Make sure the Allow This Application to Be Used to Sign In to Twitter check box is set to true as well. Don't forget to update the settings for them to take effect. Refresh the page until you see the settings are correct.

Now that the Twitter side of things is set up you can look at some R code to help you log in to Twitter.

You need to open a text editor and use the following code:

```
library(twitteR)
cred <- OAuthFactory$new(consumerKey="xxxxxxxxxxxxx",
 consumerSecret="xxxxxxxxxxxx",
 requestURL="http://api.twitter.com/oauth/request_token",
 accessURL="http://api.twitter.com/oauth/access_token",
 authURL="http://api.twitter.com/oauth/authorize")
download.file(url="http://curl.haxx.se/ca/cacert.pem", destfile="cacert.
pem")
cred$handshake(cainfo="cacert.pem")
```

Replace the consumer key and secret values with the ones you have for your application. Save the file as `twitterconnect.r` and then quit the text editor.

Back in the R command line, you load the source file, and it runs as soon as it's loaded.

```
>source(twitterconnect.r')
```

You see the twitteR library load the dependencies and then attempt to connect to Twitter:

```
> source("twitterconnect.r")
Loading required package: ROAuth
Loading required package: RCurl
Loading required package: bitops
Loading required package: digest
Loading required package: rjson
trying URL 'http://curl.haxx.se/ca/cacert.pem'
Content type 'text/plain' length 251338 bytes (245 Kb)
opened URL
==================================================
downloaded 245 Kb
To enable the connection, please direct your web browser to:

http://api.twitter.com/oauth/authorize?oauth_token=cv88UGfPrAJnraJPqdGje
g5QJEUMk185jOUncJhDk

When complete, record the PIN given to you and provide it here:
```

The main thing to look out for is the connecting URL with the oauth_token key. Copy the whole URL and paste it into a browser. You go to the Twitter site where you're asked to authorize your application request. When you accept this you'd normally return to the application, but because you've disabled that feature, you get a personal identification number (PIN) instead.

Back at the R command line, the program you have written is currently waiting for input—the PIN—so type that in the R window.

As soon as the PIN is entered you're returned to the R prompt. You need to register the oauth with the twitteR library, which you do with the following command:

```
>registerTwitterOAuth(cred)
```

The R command line responds with TRUE when the credentials are registered. After that's done you can run a quick test:

```
searchTwitter("#bigdata")
```

You start to see results come through to the command line:

```
[[1]]
[1] "eriksmits: #BigData could generate millions of new jobs http://t.
co/w1FGdxjBI9 via @FortuneMagazine, is a Java Hadoop developer key for
creating value?"
[[2]]
[1] "MobileBIAus: RT @BI_Television: RT @DavidAFrankel: Big Data
Collides with Market Research http://t.co/M36yXMWJbg #bigdata
#analytics"
```

```
[[3]]
[1] "alibaba_aus: @PracticalEcomm discusses how the use of #BigData can
combat #ecommerce fraud http://t.co/fmd9m0wWeH"
[[4]]
[1] "alankayvr: RT @ventanaresearch: It's not too late! Join us in S.F.
for the 2013 Technology Leadership Summit - sessions on #BigData #Cloud
&amp; more http. . ."
[[5]]
[1] "DelrayMom: MT @Loyalty360: White Paper 6 #Tips for turning
#BigData into key #insights, http://t.co/yuNRWxzXfZ, @SAS, #mktg #data
#contentmarketing"
```

It's worth saving the credentials so you don't have to keep re-authorizing the access tokens via Twitter. You can save them to a file:

```
>save(cred,file="credentials.RData")
```

The next time you want to use the Twitter credentials again, you can do it in two lines:

```
>load("credentials.RData")
>registerTwitterOAuth(cred)
```

Summary

R is a complex piece of software, but it's well-loved by data scientists (new and old ones alike) and also it's become the de facto statistics software for the open source generation.

There are hundreds of well-developed and documented libraries for R within the CRAN package library.

Development does come at a cost; it can be memory intensive for large and complex jobs. It doesn't handle huge amounts of data well, but this is improved by the work done by Revolution Analytics and Purdue University bringing Hadoop processing power to the R engine.

If you still prefer the comfort of your "normal" programming language, such as Java, then use the JRI/RJava combinations; they work very well. Think about real-time analytics to your Java web servlets, for example—very powerful indeed.

Regardless of whether you're processing social media feeds, evaluating e-commerce shopping baskets, or reading sensor data from temperature gauges, look at R as an alternative for processing the data.

SpringXD Quick Start

You can download SpringXD from the Pivotal site at `http://projects.spring`
`.io/spring-xd`.

Installing Manually

Use the following steps to install SpringXD:

1. Create a new directory:

   ```
   mkdir /opt/springxd
   ```

2. Unzip the contents of the downloaded zip file.

3. Set the environment variable to the directory:

   ```
   export XD_HOME=/opt/springxd/spring-xd-1.0.0.M7/
   ```

Starting SpringXD

Start SpringXD by doing the following:

1. Change the directory to the location in which you installed SpringXD:

   ```
   cd /opt/springxd/spring-xd-1.0.0.M7/xd
   ```

2. Start the SpringXD server with the following command:

   ```
   ./bin/xd-singlenode &
   ```

Creating a Stream

Assuming that the SpringXD server is up and running, launch the SpringXD shell in another terminal window:

```
../shell/bin/xd-shell
```

From the SpringXD `xd:>` command prompt, create a stream like so:

```
xd:>stream create --name ticker --definition "time | log" –deploy
```

Go back to your SpringXD server terminal window and check the output of the stream to confirm it's working properly.

```
16:39:06,323  INFO task-scheduler-1 sink.ticker:155 - 2014-06-30
16:39:06
16:39:07,326  INFO task-scheduler-4 sink.ticker:155 - 2014-06-30
16:39:07
16:39:08,330  INFO task-scheduler-1 sink.ticker:155 - 2014-06-30
16:39:08
16:39:09,331  INFO task-scheduler-1 sink.ticker:155 - 2014-06-30
16:39:09
16:39:10,333  INFO task-scheduler-1 sink.ticker:155 - 2014-06-30
16:39:10
16:39:11,339  INFO task-scheduler-7 sink.ticker:155 - 2014-06-30
16:39:11
16:39:12,341  INFO task-scheduler-8 sink.ticker:155 - 2014-06-30
16:39:12
```

Adding a Twitter Application Key

If you want to use SpringXD to ingest data from the Twitter application programming interface (API), you first need to have created an application on the Twitter developer site (`http://dev.twitter.com`).

In SpringXD you can store the Twitter credentials in a properties file. You can find a template file in the `./xd/config/modules/directory`. Older builds of SpringXD use the `twitter.properties` file directory and newer builds use the `modules.yml` file.

To add a Twitter stream API key, add the following lines to the appropriate file with your text editor (I used vi in this example):

```
vi modules.yml
```

Uncomment the following lines:

```
twitter:
consumerKey: wSs8crbV3Wa. . . .
consumerSecret: yKKv7UqutMsvXyMuQks. . . .
accessToken: 1248789104-k5QurElGEHMabclPw. . . .
accessTokenSecret: ceqZkUPUYqv391qoKlFx6YGAe. . . .
```

Add the consumer key, secret and access tokens at their appropriate lines in the config file. If you want to use another set of credentials, you can define them within the stream. Changes to these files take effect only after you restart the SpringXD server.

Hadoop 1.x Quick Start

Chapter 10 covers Hadoop, and the example is based on Hadoop's MapReduce version 1 engine (MR1). The instructions in this appendix are for the MR1 engine. You might want to consider the Hadoop 2 version, which has performance improvements; installation is very similar to the following steps.

A NOTE FOR WINDOWS USERS

The Apache open source version of Hadoop is not really suitable for Windows operating systems. It's possible to run via Cygwin, but it is slow and clunky. If you are on Windows, you probably want to look at the Hortonworks open source Hadoop distribution. Hortonworks has developed a version of Hadoop for Windows that runs natively.

If you are running Mac OS X or a Unix operating system (for example, Linux) then the following instructions apply.

Downloading and Installing Hadoop

You can download Hadoop from one of the Apache mirror sites. I used `http://mirrors.ukfast.co.uk/sites/ftp.apache.org/hadoop/common/`.

Look for the `hadoop-1.2.x` release and download the `.tar.gz`, where x is the highest number. There are extra links to "stable1," "stable2," and "current," that help guide you to the correct version.

```
tar xvzf hadoop-x.x.x-bin.tar.gz
```

In the configuration directory (called `conf`), edit the `hadoop-env.sh` file and edit the `JAVA_HOME` line:

```
export JAVA_HOME=/path/to/wherever/your-java/is
```

Also (this is a part that's very rarely mentioned), if your SSH configuration has a different port set up (the default port is 22, but the paranoid among us run it on another port), then you also need to comment out the `HADOOP_SSH_OPS` and add a line that reads:

```
export HADOOP_SSH_OPS="-p <my ssh port>"
```

TIP **Obviously you should change** `<my ssh port>` **to the actual number on which your** SSHD **accepts connections.**

Next, if you have not already done so, create a passwordless RSA key on your machine:

```
ssh-keygen -t rsa -P ''
```

That command should save the key under your home directory in `~/.ssh/id_rsa`.

Now, append that file to the authorized keys file in that same directory on the same machine:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Add Hadoop's bin path to your working path. Consider also adding this line to your `~/.bash_profile` so that it executes each time you log in.

```
export PATH=$PATH:/path/to/hadoop/bin
```

The last thing to do is to define Hadoop's filesystem settings in `conf/core-site.xml`.

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

Formatting the HDFS Filesystem

From the command line, run the following:

```
hadoop namenode -format
```

When executing this command, you're prompted with Y or N to proceed; make sure you enter uppercase Y and not lowercase y. It's case sensitive.

Starting and Stopping Hadoop

To start Hadoop, run:

```
start-all.sh
```

To stop it, use:

```
stop-all.sh
```

Process List of a Basic Job

Start up the server:

```
start-all.sh
```

Copy the data you want to process to the HDFS filesystem:

```
hadoop fs -put mydata.txt mydata.txt
```

Run the Hadoop job. There are two parameters: the input file or directory in HDFS and the directory to output the results (to ensure it doesn't already exist on HDFS):

```
hadoop jar /path/to/hadoop-*-examples.jar wordcount mydata.txt output
```

Hadoop then processes the job. To see the results, you need to copy the result data from the HDFS filesystem back to your local filesystem.

```
hadoop fs --getmerge output output.txt
```

Stop the server:

```
stop-all.sh
```

Useful Unix Commands

Regardless of the operating system you use on a daily basis, there's nothing wrong with learning some handy Unix commands. The commands covered in this section will help you on the day-to-day tasks of quickly testing, parsing, and searching through your text data.

If you're a Windows user, you can still join in by downloading Cygwin, which is a Unix shell command interpreter. Cygwin is a shell that sits on top of the Windows Command application and behaves like it's a Unix install.

Some of these commands will appear from time to time in the chapters of this book, especially in Chapters 9 and 10, so it's worth reviewing them now. Experiment with them and study the output so you have an idea of what to expect.

Using Sample Data

Before you get started with the tools, you need some sample data. With a text editor, type out the following lines and separate each value with a tab.

```
987    1391548780   hhh bbb
988    1391548781   sda jjj
989    1391548782   asd asd
990    1391548783   gjh jkl
991    1391548784   abc abc
```

```
992     1391548785    ghj gjh
993     1391548785    hhh bbb
994     1391548785    sda jjj
995     1391548786    asd asd
996     1391548787    gjh jkl
997     1391548787    abc abc
998     1391548787    ghj gjh
```

Name the text file `text.txt`, and then you can follow along with the following commands and see the output. The sample data is basically comprised of a unique id, a timestamp, and some text. It's the sort of thing you would see within a database table but is output as text. This example uses a tab delimiter, but you might find data with commas, semicolons, and other characters used as a delimiter.

Showing the Contents: cat, more, and less

The `cat` command concatenates and prints the contents of one or more files to the console output.

Example Command

```
cat text.txt text2.txt text3.txt
```

Expected Output

```
987     1391548780    hhh bbb
988     1391548781    sda jjj
989     1391548782    asd asd
990     1391548783    gjh jkl
991     1391548784    abc abc
992     1391548785    ghj gjh
993     1391548785    hhh bbb
994     1391548785    sda jjj
995     1391548786    asd asd
996     1391548787    gjh jkl
997     1391548787    abc abc
998     1391548787    ghj gjh
```

Adding `-b` to the options gives you the line numbers, too.

```
$ cat -b sample.txt
     1 987     1391548780    hhh bbb
     2 988     1391548781    sda jjj
     3 989     1391548782    asd asd
```

```
 4 990    1391548783    gjh jkl
 5 991    1391548784    abc abc
 6 992    1391548785    ghj gjh
 7 993    1391548785    hhh bbb
 8 994    1391548785    sda jjj
 9 995    1391548786    asd asd
10 996    1391548787    gjh jkl
11 997    1391548787    abc abc
12 998    1391548787    ghj gjh
```

If too much content is showing in the console for you to keep up with, you can add the `more` command after the `cat` command:

```
$ cat sample | more
```

Alternatively the `less` command gives you a controlled environment for viewing the contents of files but does not let you edit them.

Filtering Content: grep

For matching patterns within text, `grep` is your friend. It's one of those utilities you'll use again and again after you get used to it. The syntax is very basic: `grep [options] [pattern to find] [name of file(s)].`

Example Command for Finding Text

```
$grep 'bbb bbb' sample.txt
```

Example Output

To invert the output, find the lines that don't match the pattern then add the `–v` flag before the pattern. Table C-1 shows other handy option flags you can use with `grep`.

Table C-1: grep Option Flags

FLAG	EFFECT ON OUTPUT
-c	Outputs the number of times the pattern was matched in the file
-v	Inverts the output to show lines that don't match the pattern
-i	Ignores the case on the input line so "BBB" and "bbb" would match
-n	Displays the line number on which the pattern match occurs
-l	Lists the filenames where the pattern matches

The pattern matching can be taken a step further by introducing Perl-like patterns with the −P option. There are numerous books on the subject of Perl and Regular Expressions.

```
$ grep -e '[1-5]\tsda' sample.txt
988    139154878    sda jjj
994    1391548785   sda jjj
```

Sorting Data: sort

The sort command takes the input file and sorts in ascending or descending order. By default, sort assumes that everything is a string. (You'll read about number values in a moment.)

Example Command for Basic Sorting

```
$sort sample.txt
```

Example Output

```
987    1391548780   hhh bbb
988    1391548781   sda jjj
989    1391548782   asd asd
990    1391548783   gjh jkl
991    1391548784   abc abc
992    1391548785   ghj gjh
993    1391548785   hhh bbb
994    1391548785   sda jjj
995    1391548786   asd asd
996    1391548787   gjh jkl
997    1391548787   abc abc
998    1391548787   ghj gjh
```

The −r flag outputs the results in descending order.

```
$ sort -r sample.txt
998    1391548787   ghj gjh
997    1391548787   abc abc
996    1391548787   gjh jkl
995    1391548786   asd asd
994    1391548785   sda jjj
993    1391548785   hhh bbb
992    1391548785   ghj gjh
991    1391548784   abc abc
990    1391548783   gjh jkl
989    1391548782   asd asd
988    1391548781   sda jjj
987    1391548780   hhh bbb
```

So far the example has covered sorting strings. Consider the following list of numbers:

```
10
18
1
20
17
15
103
110
12
22
21
201
```

Running a default `sort` command would sort the number values as strings, so the output would be correct, but it probably would not be what you hoped for.

```
$ sort sample2.txt
1
10
103
110
12
15
17
18
20
201
21
22
```

The `-n` option treats the input data as numeric.

```
$ sort -n sample2.txt
1
10
12
15
17
18
20
21
22
103
110
201
```

The final useful option is the -k flag, which splits the input data into columns and lets you sort on a specific column. In the sample data, the first three-letter text column is the third, so by using the option -k3 you sort on the text column.

```
$ sort -k3 sample.txt
991    1391548784    abc abc
997    1391548787    abc abc
989    1391548782    asd asd
995    1391548786    asd asd
992    1391548785    ghj gjh
998    1391548787    ghj gjh
990    1391548783    gjh jkl
996    1391548787    gjh jkl
987    1391548780    hhh bbb
993    1391548785    hhh bbb
988    1391548781    sda jjj
994    1391548785    sda jjj
```

When combining the sort option flags, you can output pretty much anything you want. For example, you could reverse sort the second column as a numeric value using the following code:

```
$ sort -k2nr sample.txt
996    1391548787    gjh jkl
997    1391548787    abc abc
998    1391548787    ghj gjh
995    1391548786    asd asd
992    1391548785    ghj gjh
993    1391548785    hhh bbb
994    1391548785    sda jjj
991    1391548784    abc abc
990    1391548783    gjh jkl
989    1391548782    asd asd
988    1391548781    sda jjj
987    1391548780    hhh bbb
```

The combination of the -k, -n, and -r flags is suitable for most applications when you're running through basic delimited data.

Finding Unique Occurrences: uniq

Sometimes data will have repeat lines in them due to a data entry error or a slightly error-prone database query. Before raising your voice at the database operator (I have to look in the mirror to talk to my DBA), you can use the uniq command to extract all the unique lines.

```
$ uniq sample.txt
987    1391548780    hhh bbb
```

```
988    1391548781   sda jjj
989    1391548782   asd asd
990    1391548783   gjh jkl
991    1391548784   abc abc
992    1391548785   ghj gjh
993    1391548785   hhh bbb
994    1391548785   sda jjj
995    1391548786   asd asd
996    1391548787   gjh jkl
997    1391548787   abc abc
998    1391548787   ghj gjh
```

Using `uniq` in combination with the `-c` flag, the output will contain the number of repeats that were found within the file. If you do not need to see the counts then you can use the `-u` flag in the `sort` command.

Showing the Top of a File: head

When you've downloaded a 10,000-line file and you want to quickly inspect it, it's so easy to use the `cat` command and dump the whole file out. Obviously, you can't read it as the lines are appearing faster than your eyes. The `head` command shows the output of a defined number of lines.

```
$ head -n 5 sample.txt
987    1391548780   hhh bbb
988    1391548781   sda jjj
989    1391548782   asd asd
990    1391548783   gjh jkl
991    1391548784   abc abc
```

If you want to show the last number of lines of the file, then use the `tail` command.

Counting Words: wc

Using `wc` counts the number of lines, words, and characters used within the text document.

```
$ wc sample.txt
     13      48    277 sample.txt
```

If you only want to see the number of lines, then use `-l` as an option, `-w` for words, and `-m` for characters.

Locating Anything: find

The `find` command is one of my favorites. It takes a little getting used to, but once you do, it will be a big help to you.

The syntax starts with a source directory (using a period [.] if you want to start in the current directory) and various text options. For example, here's what it looks like to run the `find` command on the home directory on my machine.

```
$ find . -type f -name '*.txt' -print
./.gradle/caches/1.10/scripts/build_361heej71i7errcd096649rjns/
ProjectScript/buildscript/classes/emptyScript.txt
./.gradle/caches/1.8/scripts/build_4p3rjceekul5gbo8of3d9h4hso/
ProjectScript/buildscript/classes/emptyScript.txt
./.rvm/config/displayed-notes.txt
./.rvm/gems/ruby-1.9.2-p320/gems/arel-3.0.2/History.txt
./.rvm/gems/ruby-1.9.2-p320/gems/arel-3.0.2/Manifest.txt
```

With the `-type` flag, I am looking for files (`f`). `-name` gives the types of file I'm looking for—in this instance any file with the extension `.txt`. I print the results with the `-print` flag.

So far, there's nothing here that a few Unix commands couldn't do. The great thing about `find` is that you can use Unix commands within the command itself. Imagine you're looking for any files with the phrase "SOFTWARE IS PROVIDED" in the body of the text file:

```
$ find . -type f -name "*.txt" -exec grep "SOFTWARE IS PROVIDED" {} \; -print
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
./.rvm/gems/ruby-1.9.2-p320/gems/arel-3.0.2/MIT-LICENSE.txt
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
./.rvm/gems/ruby-1.9.2-p320/gems/polyglot-0.3.3/License.txt
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
./.rvm/gems/ruby-1.9.2-p320/gems/polyglot-0.3.3/README.txt
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
./.rvm/gems/ruby-1.9.2-p320/gems/rack-test-0.6.1/MIT-LICENSE.txt
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
./.rvm/gems/ruby-1.9.2-p320/gems/rack-test-0.6.2/MIT-LICENSE.txt
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
./.rvm/gems/ruby-1.9.2-p320/gems/uglifier-1.2.6/LICENSE.txt
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
./.rvm/gems/ruby-1.9.2-p320/gems/uglifier-1.3.0/LICENSE.txt
```

This example uses the standard `grep` command that is covered earlier in this chapter, but it replaces the filename with `{}`, which is a placeholder for the filename on which the `find` command is working.

`find` has saved my programming sanity many times over, and if I'm looking for a file that has a method name but I can't remember where it is (this happens a lot with large codebases), I don't have to fire up my IDE. I just use `find` from the command line.

Combining Commands and Redirecting Output

The explanations for the Unix commands showed you how to use one command at a time. Using the pipe symbol (|), you can chain these commands together. With the greater than symbol (>), you can redirect output to a new file.

```
$ grep 'hhh' sample.txt | sort | less
987    1391548780    hhh bbb
993    1391548785    hhh bbb
```

The preceding command uses `grep` to search for `'hhh'` in the `sample.txt` file and then runs the result through the `sort` command; this is then written to a new file called `newsample.txt`. Finally, the output of the new file is sent to the console with `cat`.

Picking a Text Editor

The choice of text editor is a very personal one, akin to liking a specific music group, sports team, or a favorite actor or actress. I'm not saying fights have broken out about these editors, but conversations over coffee and beer have sometimes been emotional. If you want to work out the personality type of a software developer, just ask him what text editor he uses.

Ultimately it's up to you what sort of text editor you'll use. You might not be using it all the hours of the day, but for the times you need to quickly look at or edit something, you'll want an editor that you can use fluently and without fuss.

Colon Frenzy: Vi and Vim

Vi (see Figure C-1) was written in 1976 by Bill Joy and was introduced in 1978 when it was released as part of BSD Unix. Pronounced "vee eye," it's been the mainstay of Unix system administrators for a long time. It's still one of the most widely used text editors today.

The thing that makes vi challenging is the concept of "modes" for the arcane commands: There are line-oriented "ex" commands that operate on lines of text; they are needed to write the file (`:w`) or quit the application (`:q`). If you want to quit without saving, then `:q!` is needed. Likewise, overwriting an existing file uses `:w!` There is an "insert mode" while you are entering new text, and there is an "edit mode" for moving and changing text. Vi is so simple there is a coffee mug available with all (100%) of the vi commands on it.

It takes time to learn the commands, how to delete lines, yank text into "buffers" then paste it, how to search for text, and other basic functions. After a bit of time, the "modes" actually becomes second nature. Vi includes a powerful "macro language" for adding your own commands or command sequences.
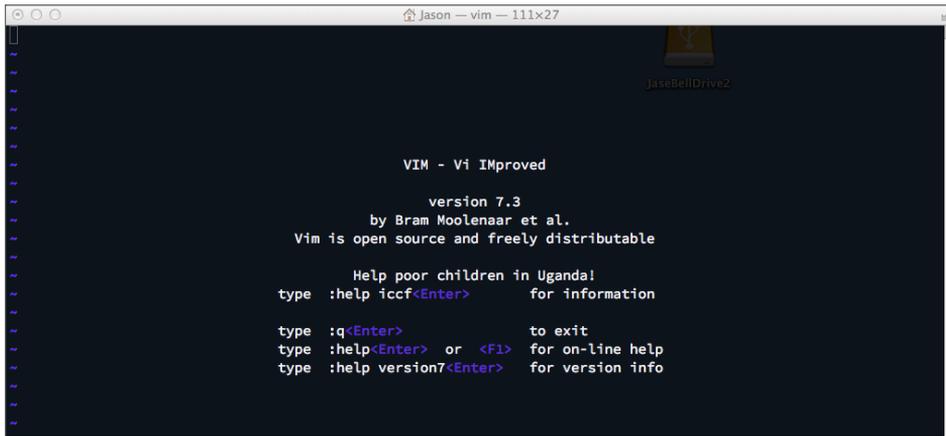
Figure C-1: The vi editor

I'm still a user of vi; it covers most of my needs. Though, I have to admit it drove me mad when I first used it in 1995. I've still not moved over to vim (Vi iMproved), though it has windows, better editing, mouse support, and colorful syntax highlighting.

Nano

For those that can't cope with the colon command, the nano editor (see Figure C-2) provides command-line heaven without all the complication. The nano editor lets you navigate around using the arrow keys (vi uses the h, j, k, and l keys for navigating). The editor includes a decent number of hints at the bottom of the screen, so you can see how you can save and quit.

For sheer speed of loading, editing, saving, and exiting a document, the nano is a very good choice.

Emacs

Compared to vi and nano, Emacs (see Figure C-3) is the big gun of the text editors. It's highly customizable and configurable. With the resurgence in Lisp-like languages, Clojure being an example, usage of Emacs is on the rise.

Richard Stallman and Guy Steel, Jr. wrote the original program in 1976. If you want to try Emacs I'd recommend the xemacs version of the program.
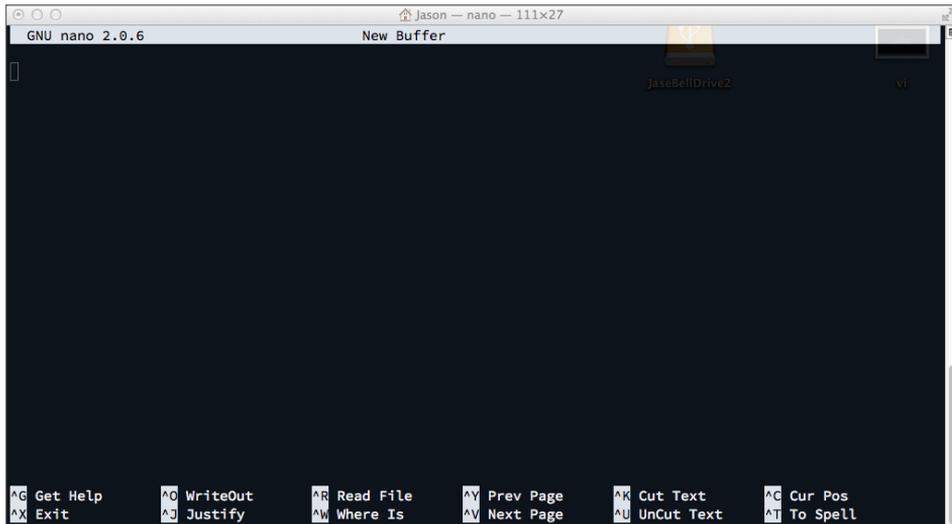
Figure C-2: The nano editor
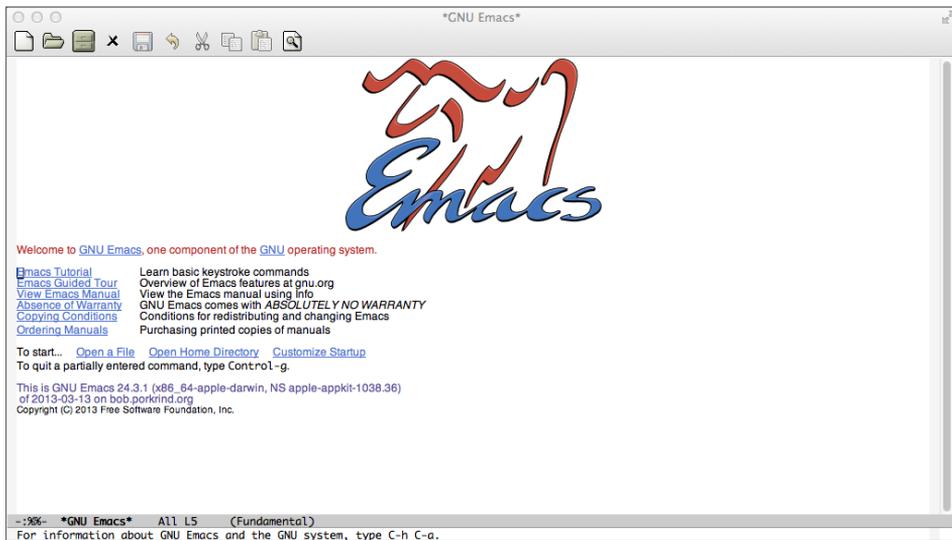
Figure C-3: The Emacs editor

Emacs bypasses the whole insert mode issue that vi users are willing to adopt. Like nano, in Emacs you type onscreen and you see the changes. It also has more than 2,000 built-in commands for those who are happy to use Emacs as their main editor for everything. It is extensible by writing Lisp programs. To exit Emacs if you start it accidentally, press Ctrl-X Ctrl-C.

Further Reading

Machine learning is only part of the story; it's the application of knowing what to use to get the insight you need. The domain of data science combines several disciplines that cover programming, math, domain knowledge, and visualization.

It's very rare for one book to cover it all. To that end, I've included some further reading that will be of help to you on your machine learning and data journey. (I know what you're thinking, and yes, I have bought and read all of these books.)

Machine Learning

The machine learning arena is a huge domain and the majority of the books written are big, in-depth, heavy affairs that can take time to read, digest, and appreciate. Two stand out:

Data Mining – Practical Machine Learning Tools and Techniques by Ian H. Witten, Eibe Frank, and Mark A. Hall (Morgan Kaufmann, 2011, ISBN 9780123748560).

Collective Intelligence in Action by Satnam Alag (Manning, 2008, ISBN 9781933988313).

Statistics

More and more emphasis is being put on statistical knowledge and its application. Sometimes it feels hard to get into, especially for software developers, so these two titles will help you along:

Naked Statistics: Stripping the Dread from the Data by Charles Wheelan (Norton, 2013, ISBN 9780393071955).

Keeping Up with the Quants: Your Guide to Understanding and Using Analytics by Thomas H. Davenport and Jinho Kim (Harvard Business Review Press, 2013, ISBN 9781422187258).

Big Data and Data Science

Regardless of whether you are a supporter of the term "Big Data," there's no denying the impact that data has on industry. In Big Data, planning is key, and it's important to have a proper understanding of the implications of planning and insight.

Data Just Right: Introduction to Large-Scale Data & Analytics by Michael Manoochehri (Addison-Wesley, 2014, ISBN 9780321898654).

Big Data: Understanding How Data Powers Big Business by Bill Schmarzo (Wiley, 2013, ISBN 9781118739570).

Big Data @ Work: Dispelling the Myths, Uncovering the Opportunities by Thomas H. Davenport (Harvard Business Review Press, 2014, 9781422168165).

Big Data: A Revolution That Will Transform How We Live, Work, and Think by Viktor Mayer-Schonberger and Kenneth Cukier (Eamon Dolan/Houghton Mifflin Harcourt, 2013, ISBN 9780544002692).

Data Smart: Using Data Science to Transform Information into Insight by John W. Foreman (Wiley, 2013, ISBN 9781118661468).

Data Science for Business: What You Need To Know About Data Mining and Data-Analytic Thinking by Foster Provost and Tom Fawcett (O'Reilly Media, 2013, ISBN 9781449361327).

Hadoop

The Hadoop platform has earned its place as the tool of use for distributed computing. It has transformed how companies can process volumes of data over commodity hardware. Although Hadoop 1.x was about the processing of blocks of data, Hadoop 2.x is about the data platform as an enterprise operating system. These books will get you up to speed:

Apache Hadoop YARN: Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2 by Arun C. Murthy, Vinod Kumar Vavilapalli, Doug Eadline, Joseph Niemiec, and Jeff Markham (Addison-Wesley, 2014, ISBN 9780321934505).

Professional Hadoop Solutions by Boris Lublinsky, Kevin T. Smith, and Alexey Yakubovich (Wiley, 2013, ISBN 9781118611937)

Hadoop: The Definitive Guide by Tom White (O'Reilly Media, 2012, ISBN 9781449311520).

Programming Pig by Alan Gates (O'Reilly Media, 2011, ISBN 9781449302641).

Visualization

My book concentrates on the pure back-end processing of data with machine learning techniques, but do not discount the power of visualization to communicate your results. These books will help:

Visualize This: The Flowing Data Guide to Design, Visualization, and Statistics by Nathan Yau (Wiley, 2011, ISBN 9780470944882).

Information Is Beautiful by David McCandless (Harper Collins, 2012, ISBN 9780007492893).

Facts Are Sacred by Simon Rogers (Faber & Faber, 2013, 9780571301614).

Making Decisions

The key to machine learning projects is making good decisions. With insight in hand, you can form next steps. The books listed here aren't software oriented at all, but they will give you vast pools of thinking about how to process and make decisions with the information you have:

Eyes Wide Open by Noreena Hertz (HarperCollins, 2013, ISBN 9780062268617).

The Signal and the Noise: Why So Many Predictions Fail—but Some Don't by Nate Silver (Penguin Books, 2012, ISBN 9781594204111).

Risk Savvy: How to Make Good Decisions by Gerd Gigerenzer (Penguin Books, 2014, ISBN 9780670025657).

Lean Analytics: Use Data to Build a Better Startup Faster by Alistair Croll and Benjamin Yoskovitz (O'Reilly Media, 2013, ISBN 9781449335670).

Datasets

Sometimes it's hard to find data to play with. Luckily, there are a few websites with loads of the stuff to download:

- **UCI Machine Learning Repository:** `http://archive.ics.uci.edu/ml/`

  The UCI maintains 290 datasets covering many different domains. What's the most popular downloaded dataset? It's still the iris.

- **Hilary Mason:** `http://bit.ly/bundles/hmason/1`

  Hilary is Scientist Emeritus at Bitly, and she's also a fan of data and cheeseburgers. The website gives you links to research-quality datasets that you can use.

■ **Quora:** `http://www.quora.com/Where-can-I-find-large-datasets-open-to-the-public`

Here you'll find a long list of URLs covering all sorts of topics that you can investigate. (This site requires you to sign in.)

Blogs

And they said RSS feeds were dead…I don't think so! There are a few blogs that I keep an eye on regularly, and these are the ones that relate to what is covered in this book:

■ **FiveThirtyEight:** `http://www.fivethirtyeight.com`

Nate Silver and a team of contributors build this daily digest of stories with data, covering everything from politics down to which is the best burrito in the United States.

■ **Radar:** `http://radar.oreilly.com`

This site for emerging technologies is worth checking out for the daily "Four Short Links," which pinpoints some very interesting programs, stories, and case studies from around the Internet.

■ **MathBabe:** `http://mathbabe.org`

Cathy O'Neill's blog discusses data, quantitative issues, and other subjects within the analytics arena.

Useful Websites

Although Google does a very good job of showing you where to find the best sites, I still refer to the following sites when I'm looking for specifics.

■ **Wiley:** `http://www.wiley.com`

This is the main website for all Wiley books and also the place to go for the sample code examples for this book.

■ **Stack Overflow:** `http://www.stackoverflow.com`

A community of developers helping a community of developers, what's not to like? This site is definitely worth a quick look for answers on coding, servers, and machine learning.

The Tools of the Trade

Here are the links to the tools that are used in this book. It's worth having them bookmarked for updates and announcements.

Apache Hadoop: `http://hadoop.apache.org`
SpringXD: `http://projects.spring.io/spring-xd`
Weka: `http://www.cs.waikato.ac.nz/ml/weka`
Mahout: `http://mahout.apache.org`

Index

A

activation functions (artificial neural networks), 94, 95–96
advertising software, 6–7
Aggregator, SpringXD and, 192
algorithms
  assignments, 165–166
  association rules learning, 123–124
  decision trees, 47–48
  Forgy method of initialization, 165
  initialization, 165
  k-means, 164–168
  random partition method of initialization, 165
  updating, 166
anonymity, data and, 26–27
Apache Spark. *See* Spark
Apriori algorithm, 123–124
Arff, converting to from CSV, 114
.arff files, LibSVM library, 154–155
artificial neural networks, 91
  activation functions, 94, 95–96
  back propagation, 98–99
  connections, removing, 108
  credit applications, 93
  data center management, 93
  data preparation, 99–100
  HFT (high-frequency trading), 92–93
  learning rate, 99
  medical monitoring, 93–94
  nodes, 108
  perceptrons, 94–98, 103–105
  robotics, 93
  test data, increasing size, 108–109
  Weka and, 100–109
association rules learning, 119–120
  algorithms, 123–124
  beer and diapers, 118–119
  confidence, 121–122
  conviction, 122
  lift, 122
  Mahout, 124–131
  process, 122–123
  support, 121
  uses, 117–118
  web usage mining, 118
attributes, decision trees, 55
axons, 92

B

back propagation (artificial neural networks), 98–99
batch processing
  EMR (Elastic Map Reduce), 226–227
  frequency and, 224–225

373

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley's ebook EULA.