Machine Learning IN ACTION

Peter Harrington

SAMPLE CHAPTER

MANNING

Machine Learning in Action

by Peter Harrington

Chapter 1

brief contents

Part 1

Classification

The first two parts of this book are on supervised learning. Supervised learning asks the machine to learn from our data when we specify a target variable. This reduces the machine's task to only divining some pattern from the input data to get the target variable.

We address two cases of the target variable. The first case occurs when the target variable can take only nominal values: true or false; reptile, fish, mammal, amphibian, plant, fungi. The second case of classification occurs when the target variable can take an infinite number of numeric values, such as 0.100, 42.001, 1000.743, ... This case is called regression. We'll study regression in part 2 of this book. The first part of this book focuses on classification.

Our study of classification algorithms covers the first seven chapters of this book. Chapter 2 introduces one of the simplest classification algorithms called k-Nearest Neighbors, which uses a distance metric to classify items. Chapter 3 introduces an intuitive yet slightly harder to implement algorithm: decision trees. In chapter 4 we address how we can use probability theory to build a classifier. Next, chapter 5 looks at logistic regression, where we find the best parameters to properly classify our data. In the process of finding these best parameters, we encounter some powerful optimization algorithms. Chapter 6 introduces the powerful support vector machines. Finally, in chapter 7 we see a meta-algorithm, AdaBoost, which is a classifier made up of a collection of classifiers. Chapter 7 concludes part 1 on classification with a section on classification imbalance, which is a real-world problem where you have more data from one class than other classes.

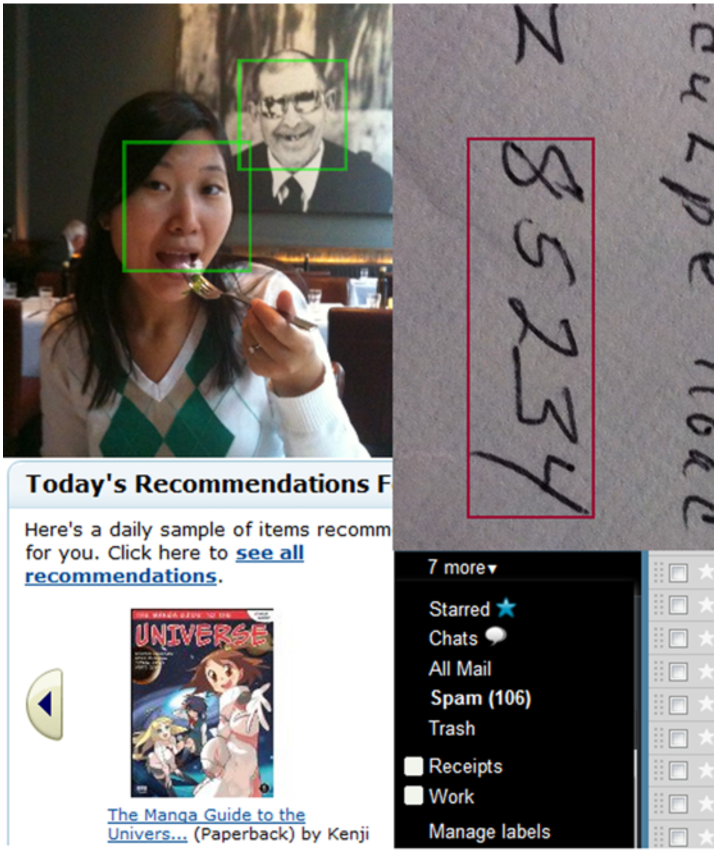
1 Machine learning basics

This chapter covers

- A brief overview of machine learning
- Key tasks in machine learning
- Why you need to learn about machine learning
- Why Python is so great for machine learning

I was eating dinner with a couple when they asked what I was working on recently. I replied, “Machine learning.” The wife turned to the husband and said, “Honey, what’s machine learning?” The husband replied, “Cyberdyne Systems T-800.” If you aren’t familiar with the Terminator movies, the T-800 is artificial intelligence gone very wrong. My friend was a little bit off. We’re not going to attempt to have conversations with computer programs in this book, nor are we going to ask a computer the meaning of life. With machine learning we can gain insight from a dataset; we’re going to ask the computer to make some sense from data. This is what we mean by learning, not cyborg rote memorization, and not the creation of sentient beings.

Machine learning is actively being used today, perhaps in many more places than you’d expect. Here’s a hypothetical day and the many times you’ll encounter machine learning: You realize it’s your friend’s birthday and want to send her a card via snail mail. You search for funny cards, and the search engine shows you the 10 most relevant links. You click the second link; the search engine learns from this. Next, you check some email, and without your noticing it, the spam filter catches unsolicited ads for pharmaceuticals and places them in the Spam folder. Next, you head to the store to buy the birthday card. When you're shopping for the card, you pick up some diapers for your friend's child. When you get to the checkout and purchase the items, the human operating the cash register hands you a coupon for $1 off a six-pack of beer. The cash register's software generated this coupon for you because people who buy diapers also tend to buy beer. You send the birthday card to your friend, and a machine at the post office recognizes your handwriting to direct the mail to the proper delivery truck. Next, you go to the loan agent and ask them if you are eligible for loan; they don't answer but plug some financial information about you into the computer and a decision is made. Finally, you head to the casino for some late-night entertainment, and as you walk in the door, the person walking in behind you gets approached by security seemingly out of nowhere. They tell him, "Sorry, Mr. Thorp, we're going to have to ask you to leave the casino. Card counters aren't welcome here." Figure 1.1 illustrates where some of these applications are being used.


Today's Recommendations F

Here's a daily sample of items recomm for you. Click here to see all recommendations.

The Manga Guide to the Univers... (Paperback) by Kenji

Figure 1.1 Examples of machine learning in action today, clockwise from top left: face recognition, handwriting digit recognition, spam filtering in email, and product recommendations from Amazon.com

In all of the previously mentioned scenarios, machine learning was present. Companies are using it to improve business decisions, increase productivity, detect disease, forecast weather, and do many more things. With the exponential growth of technology, we not only need better tools to understand the data we currently have, but we also need to prepare ourselves for the data we will have.

Are you ready for machine learning? In this chapter you'll find out what machine learning is, where it's already being used around you, and how it might help you in the future. Next, we'll talk about some common approaches to solving problems with machine learning. Last, you'll find out why Python is so great and why it's a great language for machine learning. Then we'll go through a really quick example using a module for Python called NumPy, which allows you to abstract and matrix calculations.

1.1 What is machine learning?

In all but the most trivial cases, insight or knowledge you're trying to get out of the raw data won't be obvious from looking at the data. For example, in detecting spam email, looking for the occurrence of a single word may not be very helpful. But looking at the occurrence of certain words used together, combined with the length of the email and other factors, you could get a much clearer picture of whether the email is spam or not. Machine learning is turning data into information.

Machine learning lies at the intersection of computer science, engineering, and statistics and often appears in other disciplines. As you'll see later, it can be applied to many fields from politics to geosciences. It's a tool that can be applied to many problems. Any field that needs to interpret and act on data can benefit from machine learning techniques.

Machine learning uses statistics. To most people, statistics is an esoteric subject used for companies to lie about how great their products are. (There's a great manual on how to do this called *How to Lie with Statistics* by Darrell Huff. Ironically, this is the best-selling statistics book of all time.) So why do the rest of us need statistics? The practice of engineering is applying science to solve a problem. In engineering we're used to solving a deterministic problem where our solution solves the problem all the time. If we're asked to write software to control a vending machine, it had better work all the time, regardless of the money entered or the buttons pressed. There are many problems where the solution isn't deterministic. That is, we don't know enough about the problem or don't have enough computing power to properly model the problem. For these problems we need statistics. For example, the motivation of humans is a problem that is currently too difficult to model.

In the social sciences, being right 60% of the time is considered successful. If we can predict the way people will behave 60% of the time, we're doing well. How can this be? Shouldn't we be right all the time? If we're not right all the time, doesn't that mean we're doing something wrong?

Let me give you an example to illustrate the problem of not being able to model the problem fully. Do humans not act to maximize their own happiness? Can't we just predict the outcome of events involving humans based on this assumption? Perhaps, but it's difficult to define what makes everyone happy, because this may differ greatly from one person to the next. So even if our assumptions are correct about people maximizing their own happiness, the definition of happiness is too complex to model. There are many other examples outside human behavior that we can't currently model deterministically. For these problems we need to use some tools from statistics.

1.1.1 Sensors and the data deluge

We have a tremendous amount of human-created data from the World Wide Web, but recently more nonhuman sources of data have been coming online. The technology behind the sensors isn't new, but connecting them to the web is new. It's estimated that shortly after this book's publication physical sensors will create 20 percent of non-video internet traffic.[1]

The following is an example of an abundance of free data, a worthy cause, and the need to sort through the data. In 1989, the Loma Prieta earthquake struck northern California, killing 63 people, injuring 3,757, and leaving thousands homeless. A similarly sized earthquake struck Haiti in 2010, killing more than 230,000 people. Shortly after the Loma Prieta earthquake, a study was published using low-frequency magnetic field measurements claiming to foretell the earthquake.[2] A number of subsequent studies showed that the original study was flawed for various reasons.[3,4] Suppose we want to redo this study and keep searching for ways to predict earthquakes so we can avoid the horrific consequences and have a better understanding of our planet. What would be the best way to go about this study? We could buy magnetometers with our own money and buy pieces of land to place them on. We could ask the government to help us out and give us money and land on which to place these magnetometers. Who's going to make sure there's no tampering with the magnetometers, and how can we get readings from them? There exists another low-cost solution.

Mobile phones or smartphones today ship with three-axis magnetometers. The smartphones also come with operating systems where you can execute your own programs; with a few lines of code you can get readings from the magnetometers hundreds of times a second. Also, the phone already has its own communication system set up; if you can convince people to install and run your program, you could record a large amount of magnetometer data with very little investment. In addition to the magnetometers, smartphones carry a large number of other sensors including yaw-rate gyros, three-axis accelerometers, temperature sensors, and GPS receivers, all of which you could use to support your primary measurements.

[1] http://www.gartner.com/it/page.jsp?id=876512, retrieved 7/29/2010 4:36 a.m.

[2] Fraser-Smith et al., "Low-frequency magnetic field measurements near the epicenter of the M_S 7.1 Loma Prieta earthquake," *Geophysical Research Letters* 17, no. 9 (August 1990), 1465–68.

[3] W. H. Campbell, "Natural magnetic disturbance fields, not precursors, preceding the Loma Prieta earthquake," *Journal of Geophysical Research* 114, A05307, doi:10.1029/2008JA013932 (2009).

[4] J. N. Thomas, J. J. Love, and M. J. S. Johnston, "On the reported magnetic precursor of the 1989 Loma Prieta earthquake," *Physics of the Earth and Planetary Interiors* 173, no. 3–4 (2009), 207–15.

The two trends of mobile computing and sensor-generated data mean that we'll be getting more and more data in the future.

1.1.2 Machine learning will be more important in the future

In the last half of the twentieth century the majority of the workforce in the developed world has moved from manual labor to what is known as *knowledge work*. The clear definitions of "move this from here to there" and "put a hole in this" are gone. Things are much more ambiguous now; job assignments such as "maximize profits," "minimize risk," and "find the best marketing strategy" are all too common. The fire hose of information available to us from the World Wide Web makes the jobs of knowledge workers even harder. Making sense of all the data with our job in mind is becoming a more essential skill, as Hal Varian, chief economist at Google, said:

> *I keep saying the sexy job in the next ten years will be statisticians. People think I'm joking, but who would've guessed that computer engineers would've been the sexy job of the 1990s? The ability to take data—to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it—that's going to be a hugely important skill in the next decades, not only at the professional level but even at the educational level for elementary school kids, for high school kids, for college kids. Because now we really do have essentially free and ubiquitous data. So the complementary scarce factor is the ability to understand that data and extract value from it. I think statisticians are part of it, but it's just a part. You also want to be able to visualize the data, communicate the data, and utilize it effectively. But I do think those skills—of being able to access, understand, and communicate the insights you get from data analysis—are going to be extremely important. Managers need to be able to access and understand the data themselves.*
>
> —*McKinsey Quarterly*, January 2009

With so much of the economic activity dependent on information, you can't afford to be lost in the data. Machine learning will help you get through all the data and extract some information. We need to go over some vocabulary that commonly appears in machine learning so it's clear what's being discussed in this book.

1.2 Key terminology

Before we jump into the machine learning algorithms, it would be best to explain some terminology. The best way to do so is through an example of a system someone may want to make. We'll go through an example of building a bird classification system. This sort of system is an interesting topic often associated with machine learning called *expert systems*. By creating a computer program to recognize birds, we've replaced an ornithologist with a computer. The ornithologist is a bird expert, so we've created an expert system.

In table 1.1 are some values for four parts of various birds that we decided to measure. We chose to measure weight, wingspan, whether it has webbed feet, and the color of its back. In reality, you'd want to measure more than this. It's common practice to measure just about anything you can measure and sort out the important parts later. The four things we've measured are called *features*; these are also called *attributes*, but we'll stick with the term *features* in this book. Each of the rows in table 1.1 is an *instance* made up of features.

Table 1.1 Bird species classification based on four features

	Weight (g)	Wingspan (cm)	Webbed feet?	Back color	Species
1	1000.1	125.0	No	Brown	Buteo jamaicensis
2	3000.7	200.0	No	Gray	Sagittarius serpentarius
3	3300.0	220.3	No	Gray	Sagittarius serpentarius
4	4100.0	136.0	Yes	Black	Gavia immer
5	3.0	11.0	No	Green	Calothorax lucifer
6	570.0	75.0	No	Black	Campephilus principalis

The first two features in table 1.1 are numeric and can take on decimal values. The third feature (webbed feet) is binary: it can only be 1 or 0. The fourth feature (back color) is an enumeration over the color palette we're using, and I just chose some very common colors. Say we ask the people doing the measurements to choose one of seven colors; then back color would be just an integer. (I know choosing one color for the back of a bird is a gross oversimplification; please excuse this for the purpose of illustration).

If you happen to see a Campephilus principalis (Ivory-billed Woodpecker), give me a call ASAP! Don't tell anyone else you saw it; just call me and keep an eye on the bird until I get there. (There's a $50,000 reward for anyone who can lead a biologist to a living Ivory-billed Woodpecker.)

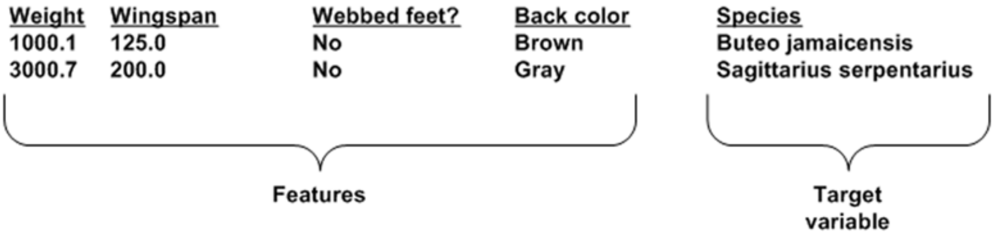
One task in machine learning is *classification*; I'll illustrate this using table 1.1 and the fact that information about an Ivory-billed Woodpecker could get us $50,000. We want to identify this bird out of a bunch of other birds, and we want to profit from this. We could set up a bird feeder and then hire an ornithologist (bird expert) to watch it and when they see an Ivory-billed Woodpecker give us a call. This would be expensive, and the person could only be in one place at a time. We could also automate this process: set up many bird feeders with cameras and computers attached to them to identify the birds that come in. We could put a scale on the bird feeder to get the bird's weight and write some computer vision code to extract the bird's wingspan, feet type, and back color. For the moment, assume we have all that information. How do we then decide if a bird at our feeder is an Ivory-billed Woodpecker or something else? This task is called *classification*, and there are many machine learning algorithms that are good at classification. The class in this example is the bird species; more specifically, we can reduce our classes to Ivory-billed Woodpecker or everything else.

Say we've decided on a machine learning algorithm to use for classification. What we need to do next is train the algorithm, or allow it to learn. To train the algorithm we feed it quality data known as a *training set*. A training set is the set of training examples we'll use to train our machine learning algorithms. In table 1.1 our training set has six *training examples*. Each training example has four features and one *target variable*; this is depicted in figure 1.2. The target variable is what we'll be trying to predict with our machine learning algorithms. In classification the target variable takes on a nominal value, and in the task of regression its value could be continuous. In a training set the target variable is known. The machine learns by finding some relationship between the features and the target variable. The target variable is the species, and as I mentioned earlier, we can reduce this to take nominal values. In the classification problem the target variables are called *classes*, and there is assumed to be a finite number of classes.

> **NOTE** Features or attributes are the individual measurements that, when combined with other features, make up a training example. This is usually columns in a training or test set.

To test machine learning algorithms what's usually done is to have a training set of data and a separate dataset, called a *test set*. Initially the program is fed the training examples; this is when the machine learning takes place. Next, the test set is fed to the program. The target variable for each example from the test set isn't given to the program, and the program decides which class each example should belong to. The target variable or class that the training example belongs to is then compared to the predicted value, and we can get a sense for how accurate the algorithm is. There are better ways to use all the information in the test set and training set. We'll discuss them later.

In our bird classification example, assume we've tested the program and it meets our desired level of accuracy. Can we see what the machine has learned? This is called *knowledge representation*. The answer is it depends. Some algorithms have knowledge representation that's more readable by humans than others. The knowledge representation may be in the form of a set of rules; it may be a probability distribution or an example from the training set. In some cases we may not be interested in building an expert system but interested only in the knowledge representation that's acquired from training a machine learning algorithm.



Weight	Wingspan	Webbed feet?	Back color	Species
1000.1	125.0	No	Brown	Buteo jamaicensis
3000.7	200.0	No	Gray	Sagittarius serpentarius

Figure 1.2 Features and target variable identified

We've covered a lot of key terms of machine learning, but we didn't cover them all. We'll introduce more key terms in later chapters as they're needed. We'll now address the big picture: what we can do with machine learning.

1.3 Key tasks of machine learning

In this section we'll outline the key jobs of machine learning and set a framework that allows us to easily turn a machine learning algorithm into a solid working application.

The example covered previously was for the task of classification. In classification, our job is to predict what class an instance of data should fall into. Another task in machine learning is *regression*. Regression is the prediction of a numeric value. Most people have probably seen an example of regression with a best-fit line drawn through some data points to generalize the data points. Classification and regression are examples of *supervised learning*. This set of problems is known as supervised because we're telling the algorithm what to predict.

The opposite of supervised learning is a set of tasks known as *unsupervised learning*. In unsupervised learning, there's no label or target value given for the data. A task where we group similar items together is known as *clustering*. In unsupervised learning, we may also want to find statistical values that describe the data. This is known as *density estimation*. Another task of unsupervised learning may be reducing the data from many features to a small number so that we can properly visualize it in two or three dimensions. Table 1.2 lists some common tasks in machine learning with algorithms used to solve these tasks.

Supervised learning tasks	
k-Nearest Neighbors	Linear
Naive Bayes	Locally weighted linear
Support vector machines	Ridge
Decision trees	Lasso
Unsupervised learning tasks	
k-Means	Expectation maximization
DBSCAN	Parzen window

Table 1.2 Common algorithms used to perform classification, regression, clustering, and density estimation tasks

If you noticed in table 1.2 that multiple techniques are used for completing the same task, you may be asking yourself, "If these do the same thing, why are there four different methods? Why can't I just choose one method and master it?" I'll answer that question in the next section.

1.4 How to choose the right algorithm

With all the different algorithms in table 1.2, how can you choose which one to use? First, you need to consider your goal. What are you trying to get out of this? (Do you want a probability that it might rain tomorrow, or do you want to find groups of voters with similar interests?) What data do you have or can you collect? Those are the big questions. Let's talk about your goal.

If you're trying to predict or forecast a target value, then you need to look into supervised learning. If not, then unsupervised learning is the place you want to be. If you've chosen supervised learning, what's your target value? Is it a discrete value like Yes/No, 1/2/3, A/B/C, or Red/Yellow/Black? If so, then you want to look into classification. If the target value can take on a number of values, say any value from 0.00 to 100.00, or -999 to 999, or +∞ to -∞, then you need to look into regression.

If you're not trying to predict a target value, then you need to look into unsupervised learning. Are you trying to fit your data into some discrete groups? If so and that's all you need, you should look into clustering. Do you need to have some numerical estimate of how strong the fit is into each group? If you answer yes, then you probably should look into a density estimation algorithm.

The rules I've given here should point you in the right direction but are not unbreakable laws. In chapter 9 I'll show you how you can use classification techniques for regression, blurring the distinction I made within supervised learning. The second thing you need to consider is your data.

You should spend some time getting to know your data, and the more you know about it, the better you'll be able to build a successful application. Things to know about your data are these: Are the features nominal or continuous? Are there missing values in the features? If there are missing values, why are there missing values? Are there outliers in the data? Are you looking for a needle in a haystack, something that happens very infrequently? All of these features about your data can help you narrow the algorithm selection process.

With the algorithm narrowed, there's no single answer to what the best algorithm is or what will give you the best results. You're going to have to try different algorithms and see how they perform. There are other machine learning techniques that you can use to improve the performance of a machine learning algorithm. The relative performance of two algorithms may change after you process the input data. We'll discuss these in more detail later, but the point is that finding the best algorithm is an iterative process of trial and error.

Many of the algorithms are different, but there are some common steps you need to take with all of these algorithms when building a machine learning application. I'll explain these steps in the next section.

1.5 Steps in developing a machine learning application

Our approach to understanding and developing an application using machine learning in this book will follow a procedure similar to this:

1. *Collect data.* You could collect the samples by scraping a website and extracting data, or you could get information from an RSS feed or an API. You could have a device collect wind speed measurements and send them to you, or blood glucose levels, or anything you can measure. The number of options is endless. To save some time and effort, you could use publicly available data.
2. *Prepare the input data.* Once you have this data, you need to make sure it's in a useable format. The format we'll be using in this book is the Python list. We'll talk about Python more in a little bit, and lists are reviewed in appendix A. The benefit of having this standard format is that you can mix and match algorithms and data sources.

   You may need to do some algorithm-specific formatting here. Some algorithms need features in a special format, some algorithms can deal with target variables and features as strings, and some need them to be integers. We'll get to this later, but the algorithm-specific formatting is usually trivial compared to collecting data.
3. *Analyze the input data.* This is looking at the data from the previous task. This could be as simple as looking at the data you've parsed in a text editor to make sure steps 1 and 2 are actually working and you don't have a bunch of empty values. You can also look at the data to see if you can recognize any patterns or if there's anything obvious, such as a few data points that are vastly different from the rest of the set. Plotting data in one, two, or three dimensions can also help. But most of the time you'll have more than three features, and you can't easily plot the data across all features at one time. You could, however, use some advanced methods we'll talk about later to distill multiple dimensions down to two or three so you can visualize the data.
4. If you're working with a production system and you know what the data should look like, or you trust its source, you can skip this step. This step takes human involvement, and for an automated system you don't want human involvement. The value of this step is that it makes you understand you don't have garbage coming in.
5. *Train the algorithm.* This is where the machine learning takes place. This step and the next step are where the "core" algorithms lie, depending on the algorithm. You feed the algorithm good clean data from the first two steps and extract knowledge or information. This knowledge you often store in a format that's readily useable by a machine for the next two steps.

   In the case of unsupervised learning, there's no training step because you don't have a target value. Everything is used in the next step.
6. *Test the algorithm.* This is where the information learned in the previous step is put to use. When you're evaluating an algorithm, you'll test it to see how well it does. In the case of supervised learning, you have some known values you can use to evaluate the algorithm. In unsupervised learning, you may have to use some other metrics to evaluate the success. In either case, if you're not satisfied,

you can go back to step 4, change some things, and try testing again. Often the collection or preparation of the data may have been the problem, and you'll have to go back to step 1.

7 *Use it.* Here you make a real program to do some task, and once again you see if all the previous steps worked as you expected. You might encounter some new data and have to revisit steps 1–5.

Now we'll talk about a language to implement machine learning applications. We need a language that's understandable by a wide range of people. We also need a language that has libraries written for a number of tasks, especially matrix math operations. We also would like a language with an active developer community. Python is the best choice for these reasons.

1.6 Why Python?

Python is a great language for machine learning for a large number of reasons. First, Python has clear syntax. Second, it makes text manipulation extremely easy. A large number of people and organizations use Python, so there's ample development and documentation.

1.6.1 Executable pseudo-code

The clear syntax of Python has earned it the name *executable pseudo-code*. The default install of Python already carries high-level data types like lists, tuples, dictionaries, sets, queues, and so on, which you don't have to program in yourself. These high-level data types make abstract concepts easy to implement. (See appendix A for a full description of Python, the data types, and how to install it.) With Python, you can program in any style you're familiar with: object-oriented, procedural, functional, and so on.

With Python it's easy to process and manipulate text, which makes it ideal for processing non-numeric data. You can get by in Python with little to no regular expression usage. There are a number of libraries for using Python to access web pages, and the intuitive text manipulation makes it easy to extract data from HTML.

1.6.2 Python is popular

Python is popular, so lots of examples are available, which makes learning it fast. Second, the popularity means that there are lots of modules available for many applications.

Python is popular in the scientific and financial communities as well. A number of scientific libraries such as SciPy and NumPy allow you to do vector and matrix operations. This makes the code even more readable and allows you to write code that looks like linear algebra. In addition, the scientific libraries SciPy and NumPy are compiled using lower-level languages (C and Fortran); this makes doing computations with these tools much faster. We'll be using NumPy extensively in this book.

The scientific tools in Python work well with a plotting tool called Matplotlib. Matplotlib can plot 2D and 3D and can handle most types of plots commonly used in the scientific world. We'll be using Matplotlib extensively throughout this book.

Python also has an interactive shell, which allows you to view and inspect elements of the program as you're developing it.

A new module for Python, called Pylab, seeks to combine NumPy, SciPy, and Matplotlib into one environment and instillation. At the time of writing, this isn't yet done but shows great promise for the future.

1.6.3 What Python has that other languages don't have

There are high-level languages that allow you to do matrix math such as MATLAB and Mathematica. MATLAB has a number of built-in features that make machine learning easier. MATLAB is also very fast. The problem with MATLAB is that to legally use it will cost you a few thousand dollars. There are third-party add-ons to MATLAB but nothing on the scale of an open source project.

There are matrix math libraries for low-level languages such as Java and C. The problem with these languages is that it takes a lot of code to get simple things done. First, you have to typecast variables, and then with Java it seems that you have to write setters and getters every time you sneeze. Don't forget subclassing. You have to subclass methods even if you aren't going to use them. At the end of the day, you have written a lot of code—sometimes tedious code—to do simple things. This isn't the case with Python. Python is clean, concise, and easy to read. Python is easy for nonprogrammers to pick up. Java and C aren't so easy to pick up and much less concise than Python.

> *All of us learn to write in the second grade. Most of us go on to greater things.*
>
> —Bobby Knight

Perhaps one day I can replace "write" with "write code" in this quote. Some people are actually interested in programming languages. But for many people a programming language is simply a tool to accomplish some other task. Python is a higher-level language; this allows you to spend more time making sense of data and less time concerned with how a machine approximates the data. Python easily allows you to effortlessly express yourself.

1.6.4 Drawbacks

The only real drawback of Python is that it's not as fast as Java or C. You can, however, call C-compiled programs from Python. This gives you the best of both worlds and allows you to incrementally develop a program. If you experiment with an idea in Python and decide it's something you want to pursue in a production system, it will be easy to make that transition. If the program is built in a modular fashion, you could first get it up and running in Python and then to improve speed start building portions of the code in C. The Boost C++ library makes this easy to do. Other tools such as Cython and PyPy allow you write typed versions of Python with performance gains over regular Python.

If an idea for a program or application is flawed, then it will be flawed at low speed as well as high speed. If an idea is a bad idea, writing code to make it fast or scale to a large number of users doesn't change anything. This makes Python so beautiful that you can quickly see an idea in action and then optimize it if needed.

Now that you know the language we're going to be using, I'm sure you're ready to start using it. In the next section, we'll walk through use of the Python shell and NumPy.

1.7 Getting started with the NumPy library

We'll use NumPy heavily in this book because we'll be doing some linear algebra. Don't worry about linear algebra—we just want to do the same math operation on lots of different data points. If we represent our data as a matrix, we can do simple math without a bunch of messy loops. Before we get into any machine learning algorithms, you should make sure you have Python working and NumPy properly installed. NumPy is a separate module for Python that doesn't come with most distributions of Python, so you'll need to install it after you've installed Python. Start a Python shell by opening a command prompt in Windows or a terminal in Linux and Mac OS. At the command line, type `python` for Linux and Mac or `c:\Python27\python.exe` in Windows. From this point on, anytime you see these symbols

```
>>>
```

it will mean the Python shell. In the Python shell type the following command.

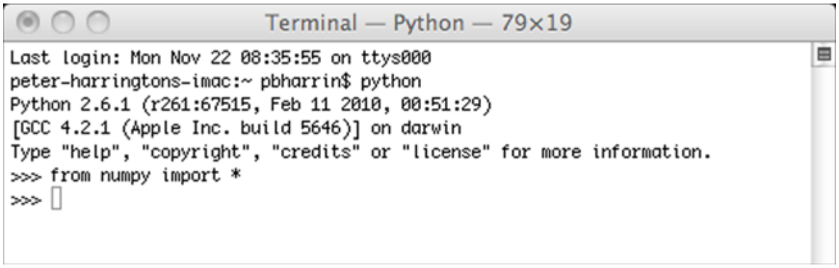
```
>>> from numpy import *
```

This imports all of the NumPy modules into the current namespace. This is shown in figure 1.3 on the Mac OS.

Next, type the following in the Python shell:

```
>>> random.rand(4,4)
array([[ 0.70328595,  0.40951383,  0.7475052 ,  0.07061094],
       [ 0.9571294 ,  0.97588446,  0.2728084 ,  0.5257719 ],
       [ 0.05431627,  0.01396732,  0.60304292,  0.19362288],
       [ 0.10648952,  0.27317698,  0.45582919,  0.04881605]])
```

This creates a random array of size 4x4; don't worry if the numbers you see are different from mine. These are random numbers, so your numbers should look different from mine.



```
Last login: Mon Nov 22 08:35:55 on ttys000
peter-harringtons-imac:~ pbharrin$ python
Python 2.6.1 (r261:67515, Feb 11 2010, 00:51:29)
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from numpy import *
>>> 
```

Figure 1.3 Starting Python from the command line and importing a module in the Python shell

NumPy matrix vs. array

In NumPy there are two different data types for dealing with rows and columns of numbers. Be careful of this because they look similar, but simple mathematical operations such as multiply on the two data types can have different meanings. The matrix data type behaves more like matrices in MATLAB.™

You can always convert an array to a matrix by calling the mat() function; type in the following:

```
>>> randMat = mat(random.rand(4,4))
```

You will probably have different values than I have here because we're getting random numbers:

```
>>> randMat.I
matrix([[ 0.24497106,  1.75854497, -1.77728665, -0.0834912 ],
        [ 1.49792202,  2.12925479,  1.32132491, -9.75890849],
        [ 2.76042144,  1.67271779, -0.29226613, -8.45413693],
        [-2.03011142, -3.07832136,  1.4420448 ,  9.62598044]])
```

The .I operator solves the inverse of a matrix. Very easy, huh? Try that in Python without NumPy. If you don't remember or never learned how to solve the inverse of a matrix, don't worry; it was just done for you:

```
>>> invRandMat = randMat.I
```

You can also do matrix multiplication. Let's see that in action:

```
>>> randMat*invRandMat
matrix([[  1.00000000e+00,   0.00000000e+00,   2.22044605e-16,
           1.77635684e-15],
        [  0.00000000e+00,   1.00000000e+00,   0.00000000e+00,
           0.00000000e+00],
        [  0.00000000e+00,   4.44089210e-16,   1.00000000e+00,
          -8.88178420e-16],
        [ -2.22044605e-16,   0.00000000e+00,   1.11022302e-16,
           1.00000000e+00]])
```

This gives you just the identity matrix, a 4x4 matrix where all elements are zero except the diagonals, which are one. This isn't exactly true. There are some very small elements left over in the array. Let's see the leftover results:

```
>>> myEye - eye(4)
matrix([[  0.00000000e+00,  -6.59194921e-17,  -4.85722573e-17,
          -4.99600361e-16],
        [  2.22044605e-16,   0.00000000e+00,  -6.03683770e-16,
          -7.77156117e-16],
        [ -5.55111512e-17,  -1.04083409e-17,  -3.33066907e-16,
          -2.22044605e-16],
        [  5.55111512e-17,   1.56125113e-17,  -5.55111512e-17,
```

The function eye(4) just creates an identity matrix of size 4.

If you got through this example, you have NumPy installed correctly. You're now ready to start making some powerful programs using machine learning. Don't worry if you haven't seen all these functions before. More NumPy functionality will be introduced as it's needed in further examples in this book.

1.8 Summary

Machine learning is already being used in your daily lives even though you may not be aware of it. The amount of data coming at you isn't going to decrease, and being able to make sense of all this data will be an essential skill for people working in a data-driven industry.

In machine learning, you look at instances of data. Each instance of data is composed of a number of features. Classification, one the popular and essential tasks of machine learning, is used to place an unknown piece of data into a known group. In order to build or train a classifier, you feed it data for which you know the class. This data is called your training set.

I don't claim that our expert system used to recognize birds will be perfect or as a good as a human. But building a machine with accuracy close to that of a human expert could greatly increase the quality of life. When we build software that can match the accuracy of a human doctor, people can more rapidly get treatment. Better prediction of weather could lead to fewer water shortages and a greater supply of food. The examples where machine learning could be useful are endless.

In the next chapter I'll introduce our first machine learning algorithm. This will be an example of classification, which is a type of supervised learning. The next six chapters will be on classification.

PYTHON/MACHINE LEARNING

Machine Learning IN ACTION

Peter Harrington

A machine is said to learn when its performance improves with experience. Learning requires algorithms and programs that capture data and ferret out the interesting or useful patterns. Once the specialized domain of analysts and mathematicians, machine learning is becoming a skill needed by many.

Machine Learning in Action is a clearly written tutorial for developers. It avoids academic language and takes you straight to the techniques you'll use in your day-to-day work. Many (Python) examples present the core algorithms of statistical data processing, data analysis, and data visualization in code you can reuse. You'll understand the concepts and how they fit in with tactical tasks like classification, forecasting, recommendations, and higher-level features like summarization and simplification.

What's Inside

- A no-nonsense introduction
- Examples showing common ML tasks
- Everyday data analysis
- Implementing classic algorithms like Apriori and Adaboost

Readers need no prior experience with machine learning or statistical processing. Familiarity with Python is helpful.

Peter Harrington is a professional developer and data scientist. He holds five US patents and his work has been published in numerous academic journals.

For access to the book's forum and a free eBook for owners of this book, go to manning.com/MachineLearninginAction

Free eBook
SEE INSERT

"An approachable and useful book."
—Alexandre Alves
Oracle Corporation

"Smart, engaging applications of core concepts."
—Patrick Toohey
Mettler-Toledo Hi-Speed

"Great examples! Teach a computer to learn anything!"
—John Griffin, Coauthor of
Hibernate Search in Action

"An approachable taxonomy, skillfully created from the diversity of ML algorithms."
—Stephen McKamey
Isomer Innovations

ISBN 13: 978-1-61729-018-3
ISBN 10: 1-61729-018-1
54499
9 781617 290183

MANNING $44.99 / Can $47.99 [INCLUDING eBOOK]