CHAPTER 1

Introduction

Suppose that you have a website selling software that you've written. You want to make the website more personalised to the user, so you start to collect data about visitors, such as their computer type/operating system, web browser, the country that they live in, and the time of day they visited the website. You can get this data for any visitor, and for people who actually buy something, you know what they bought, and how they paid for it (say PayPal or a credit card). So, for each person who buys something from your website, you have a list of data that looks like (computer type, web browser, country, time, software bought, how paid). For instance, the first three pieces of data you collect could be:

- Macintosh OS X, Safari, UK, morning, SuperGame1, credit card
- Windows XP, Internet Explorer, USA, afternoon, SuperGame1, PayPal
- Windows Vista, Firefox, NZ, evening, SuperGame2, PayPal

Based on this data, you would like to be able to populate a 'Things You Might Be Interested In' box within the webpage, so that it shows software that might be relevant to each visitor, based on the data that you can access while the webpage loads, i.e., computer and OS, country, and the time of day. Your hope is that as more people visit your website and you store more data, you will be able to identify trends, such as that Macintosh users from New Zealand (NZ) love your first game, while Firefox users, who are often more knowledgeable about computers, want your automatic download application and virus/internet worm detector, etc.

Once you have collected a large set of such data, you start to examine it and work out what you can do with it. The problem you have is one of prediction: given the data you have, predict what the next person will buy, and the reason that you think that it might work is that people who seem to be similar often act similarly. So how can you actually go about solving the problem? This is one of the fundamental problems that this book tries to solve. It is an example of what is called supervised learning, because we know what the right answers are for some examples (the software that was actually bought) so we can give the learner some examples where we know the right answer. We will talk about supervised learning more in Section 1.3.

1.1 IF DATA HAD MASS, THE EARTH WOULD BE A BLACK HOLE

Around the world, computers capture and store terabytes of data every day. Even leaving aside your collection of MP3s and holiday photographs, there are computers belonging to shops, banks, hospitals, scientific laboratories, and many more that are storing data incessantly. For example, banks are building up pictures of how people spend their money, hospitals are recording what treatments patients are on for which ailments (and how they respond to them), and engine monitoring systems in cars are recording information about the engine in order to detect when it might fail. The challenge is to do something useful with this data: if the bank's computers can learn about spending patterns, can they detect credit card fraud quickly? If hospitals share data, then can treatments that don't work as well as expected be identified quickly? Can an intelligent car give you early warning of problems so that you don't end up stranded in the worst part of town? These are some of the questions that machine learning methods can be used to answer.

Science has also taken advantage of the ability of computers to store massive amounts of data. Biology has led the way, with the ability to measure gene expression in DNA microarrays producing immense datasets, along with protein transcription data and phylogenetic trees relating species to each other. However, other sciences have not been slow to follow. Astronomy now uses digital telescopes, so that each night the world's observatories are storing incredibly high-resolution images of the night sky; around a terabyte per night. Equally, medical science stores the outcomes of medical tests from measurements as diverse as magnetic resonance imaging (MRI) scans and simple blood tests. The explosion in stored data is well known; the challenge is to do something useful with that data. The Large Hadron Collider at CERN apparently produces about 25 petabytes of data per year.

The size and complexity of these datasets mean that humans are unable to extract useful information from them. Even the way that the data is stored works against us. Given a file full of numbers, our minds generally turn away from looking at them for long. Take some of the same data and plot it in a graph and we can do something. Compare the table and graph shown in Figure 1.1: the graph is rather easier to look at and deal with. Unfortunately, our three-dimensional world doesn't let us do much with data in higher dimensions, and even the simple webpage data that we collected above has four different features, so if we plotted it with one dimension for each feature we'd need four dimensions! There are two things that we can do with this: reduce the number of dimensions (until our simple brains can deal with the problem) or use computers, which don't know that high-dimensional problems are difficult, and don't get bored with looking at massive data files of numbers. The two pictures in Figure 1.2 demonstrate one problem with reducing the number of dimensions (more technically, projecting it into fewer dimensions), which is that it can hide useful information and make things look rather strange. This is one reason why machine learning is becoming so popular — the problems of our human limitations go away if we can make computers do the dirty work for us. There is one other thing that can help if the number of dimensions is not too much larger than three, which is to use glyphs that use other representations, such as size or colour of the datapoints to represent information about some other dimension, but this does not help if the dataset has 100 dimensions in it.

In fact, you have probably interacted with machine learning algorithms at some time. They are used in many of the software programs that we use, such as Microsoft's infamous paperclip in Office (maybe not the most positive example), spam filters, voice recognition software, and lots of computer games. They are also part of automatic number-plate recognition systems for petrol station security cameras and toll roads, are used in some anti-skid braking and vehicle stability systems, and they are even part of the set of algorithms that decide whether a bank will give you a loan.

The attention-grabbing title to this section would only be true if data was very heavy. It is very hard to work out how much data there actually is in all of the world's computers, but it was estimated in 2012 that was about 2.8 zettabytes (2.8×10^{21} bytes), up from about 160 exabytes (160×10^{18} bytes) of data that were created and stored in 2006, and projected to reach 40 zettabytes by 2020. However, to make a black hole the size of the earth would

x_1	x_2	Class
0.1	1	1
0.15	0.2	2
0.48	0.6	3
0.1	0.6	1
0.2	0.15	2
0.5	0.55	3
0.2	1	1
0.3	0.25	2
0.52	0.6	3
0.3	0.6	1
0.4	0.2	2
0.52	0.5	3

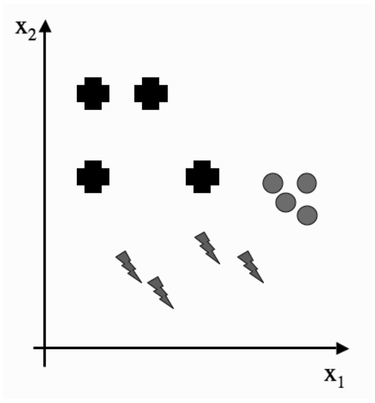

FIGURE 1.1 A set of datapoints as numerical values and as points plotted on a graph. It is easier for us to visualise data than to see it in a table, but if the data has more than three dimensions, we can't view it all at once.

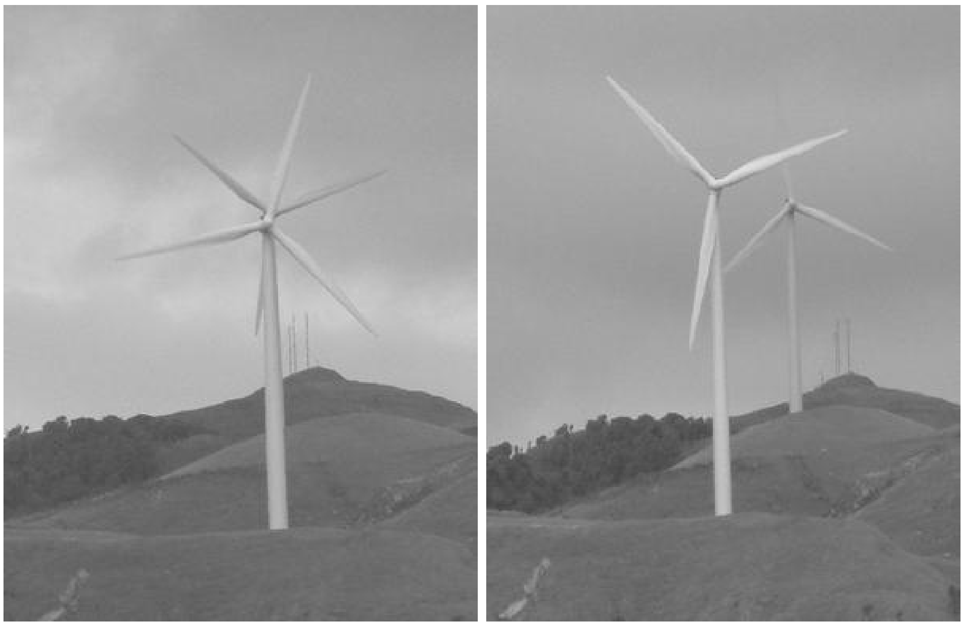

FIGURE 1.2 Two views of the same two wind turbines (Te Apiti wind farm, Ashhurst, New Zealand) taken at an angle of about 30° to each other. The two-dimensional projections of three-dimensional objects hides information.

take a mass of about 40×10^{35} grams. So data would have to be so heavy that you couldn't possibly lift a data pen, let alone a computer before the section title were true! However, and more interestingly for machine learning, the same report that estimated the figure of 2.8 zettabytes ('*Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East*' by John Gantz and David Reinsel and sponsored by EMC Corporation) also reported that while a quarter of this data could produce useful information, only around 3% of it was tagged, and less that 0.5% of it was actually used for analysis!

1.2 LEARNING

Before we delve too much further into the topic, let's step back and think about what learning actually is. The key concept that we will need to think about for our machines is learning from data, since data is what we have; terabytes of it, in some cases. However, it isn't too large a step to put that into human behavioural terms, and talk about learning from experience. Hopefully, we all agree that humans and other animals can display behaviours that we label as intelligent by learning from experience. Learning is what gives us flexibility in our life; the fact that we can adjust and adapt to new circumstances, and learn new tricks, no matter how old a dog we are! The important parts of animal learning for this book are remembering, adapting, and generalising: recognising that last time we were in this situation (saw this data) we tried out some particular action (gave this output) and it worked (was correct), so we'll try it again, or it didn't work, so we'll try something different. The last word, generalising, is about recognising similarity between different situations, so that things that applied in one place can be used in another. This is what makes learning useful, because we can use our knowledge in lots of different places.

Of course, there are plenty of other bits to intelligence, such as reasoning, and logical deduction, but we won't worry too much about those. We are interested in the most fundamental parts of intelligence—learning and adapting—and how we can model them in a computer. There has also been a lot of interest in making computers reason and deduce facts. This was the basis of most early Artificial Intelligence, and is sometimes known as symbolic processing because the computer manipulates symbols that reflect the environment. In contrast, machine learning methods are sometimes called subsymbolic because no symbols or symbolic manipulation are involved.

1.2.1 Machine Learning

Machine learning, then, is about making computers modify or adapt their actions (whether these actions are making predictions, or controlling a robot) so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones. Imagine that you are playing Scrabble (or some other game) against a computer. You might beat it every time in the beginning, but after lots of games it starts beating you, until finally you never win. Either you are getting worse, or the computer is learning how to win at Scrabble. Having learnt to beat you, it can go on and use the same strategies against other players, so that it doesn't start from scratch with each new player; this is a form of generalisation.

It is only over the past decade or so that the inherent multi-disciplinarity of machine learning has been recognised. It merges ideas from neuroscience and biology, statistics, mathematics, and physics, to make computers learn. There is a fantastic existence proof that learning is possible, which is the bag of water and electricity (together with a few trace chemicals) sitting between your ears. In Section 3.1 we will have a brief peek inside and see if there is anything we can borrow/steal in order to make machine learning algorithms. It turns out that there is, and neural networks have grown from exactly this, although even their own father wouldn't recognise them now, after the developments that have seen them reinterpreted as statistical learners. Another thing that has driven the change in direction of machine learning research is data mining, which looks at the extraction of useful information from massive datasets (by men with computers and pocket protectors rather than pickaxes and hard hats), and which requires efficient algorithms, putting more of the emphasis back onto computer science.

The computational complexity of the machine learning methods will also be of interest to us since what we are producing is algorithms. It is particularly important because we might want to use some of the methods on very large datasets, so algorithms that have high-degree polynomial complexity in the size of the dataset (or worse) will be a problem. The complexity is often broken into two parts: the complexity of training, and the complexity of applying the trained algorithm. Training does not happen very often, and is not usually time critical, so it can take longer. However, we often want a decision about a test point quickly, and there are potentially lots of test points when an algorithm is in use, so this needs to have low computational cost.

1.3 TYPES OF MACHINE LEARNING

In the example that started the chapter, your webpage, the aim was to predict what software a visitor to the website might buy based on information that you can collect. There are a couple of interesting things in there. The first is the data. It might be useful to know what software visitors have bought before, and how old they are. However, it is not possible to get that information from their web browser (even cookies can't tell you how old somebody is), so you can't use that information. Picking the variables that you want to use (which are called features in the jargon) is a very important part of finding good solutions to problems, and something that we will talk about in several places in the book. Equally, choosing how to process the data can be important. This can be seen in the example in the time of access. Your computer can store this down to the nearest millisecond, but that isn't very useful, since you would like to spot similar patterns between users. For this reason, in the example above I chose to quantise it down to one of the set `morning, afternoon, evening, night`; obviously I need to ensure that these times are correct for their time zones, too.

We are going to loosely define learning as meaning getting better at some task through practice. This leads to a couple of vital questions: how does the computer know whether it is getting better or not, and how does it know how to improve? There are several different possible answers to these questions, and they produce different types of machine learning. For now we will consider the question of knowing whether or not the machine is learning. We can tell the algorithm the correct answer for a problem so that it gets it right next time (which is what would happen in the webpage example, since we know what software the person bought). We hope that we only have to tell it a few right answers and then it can 'work out' how to get the correct answers for other problems (generalise). Alternatively, we can tell it whether or not the answer was correct, but not how to find the correct answer, so that it has to search for the right answer. A variant of this is that we give a score for the answer, according to how correct it is, rather than just a 'right or wrong' response. Finally, we might not have any correct answers; we just want the algorithm to find inputs that have something in common.

These different answers to the question provide a useful way to classify the different algorithms that we will be talking about:

Supervised learning A training set of examples with the correct responses (targets) is provided and, based on this training set, the algorithm generalises to respond correctly to all possible inputs. This is also called learning from exemplars.

Unsupervised learning Correct responses are not provided, but instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorised together. The statistical approach to unsupervised learning is known as density estimation.

Reinforcement learning This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right. Reinforcement learning is sometime called learning with a critic because of this monitor that scores the answer, but does not suggest improvements.

Evolutionary learning Biological evolution can be seen as a learning process: biological organisms adapt to improve their survival rates and chance of having offspring in their environment. We'll look at how we can model this in a computer, using an idea of fitness, which corresponds to a score for how good the current solution is.

The most common type of learning is supervised learning, and it is going to be the focus of the next few chapters. So, before we get started, we'll have a look at what it is, and the kinds of problems that can be solved using it.

1.4 SUPERVISED LEARNING

As has already been suggested, the webpage example is a typical problem for supervised learning. There is a set of data (the training data) that consists of a set of input data that has target data, which is the answer that the algorithm should produce, attached. This is usually written as a set of data $(\mathbf{x}_i, \mathbf{t}_i)$, where the inputs are $\mathbf{x}_i$, the targets are $\mathbf{t}_i$, and the i index suggests that we have lots of pieces of data, indexed by i running from 1 to some upper limit N. Note that the inputs and targets are written in boldface font to signify vectors, since each piece of data has values for several different features; the notation used in the book is described in more detail in Section 2.1. If we had examples of every possible piece of input data, then we could put them together into a big look-up table, and there would be no need for machine learning at all. The thing that makes machine learning better than that is generalisation: the algorithm should produce sensible outputs for inputs that weren't encountered during learning. This also has the result that the algorithm can deal with noise, which is small inaccuracies in the data that are inherent in measuring any real world process. It is hard to specify rigorously what generalisation means, but let's see if an example helps.

1.4.1 Regression

Suppose that I gave you the following datapoints and asked you to tell me the value of the output (which we will call y since it is not a target datapoint) when $x = 0.44$ (here, x, t, and y are not written in boldface font since they are scalars, as opposed to vectors).

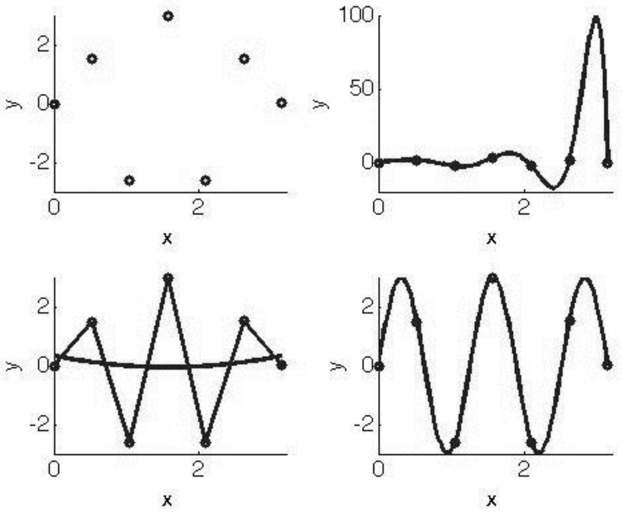

FIGURE 1.3 *Top left:* A few datapoints from a sample problem. *Bottom left:* Two possible ways to predict the values between the known datapoints: connecting the points with straight lines, or using a cubic approximation (which in this case misses all of the points). *Top and bottom right:* Two more complex approximators (see the text for details) that pass through the points, although the lower one is rather better than the top.

x	t
0	0
0.5236	1.5
1.0472	-2.5981
1.5708	3.0
2.0944	-2.5981
2.6180	1.5
3.1416	0

Since the value $x = 0.44$ isn't in the examples given, you need to find some way to predict what value it has. You assume that the values come from some sort of function, and try to find out what the function is. Then you'll be able to give the output value y for any given value of x. This is known as a regression problem in statistics: fit a mathematical function describing a curve, so that the curve passes as close as possible to all of the datapoints. It is generally a problem of function approximation or interpolation, working out the value between values that we know.

The problem is how to work out what function to choose. Have a look at Figure 1.3. The top-left plot shows a plot of the 7 values of x and y in the table, while the other plots show different attempts to fit a curve through the datapoints. The bottom-left plot shows two possible answers found by using straight lines to connect up the points, and also what happens if we try to use a cubic function (something that can be written as $ax^3 + bx^2 + cx + d = 0$). The top-right plot shows what happens when we try to match the function using a different polynomial, this time of the form $ax^{10} + bx^9 + \ldots + jx + k = 0$, and finally the bottom-right plot shows the function $y = 3\sin(5x)$. Which of these functions would you choose?

The straight-line approximation probably isn't what we want, since it doesn't tell us much about the data. However, the cubic plot on the same set of axes is terrible: it doesn't get anywhere near the datapoints. What about the plot on the top-right? It looks like it goes through all of the datapoints exactly, but it is very wiggly (look at the value on the y-axis, which goes up to 100 instead of around three, as in the other figures). In fact, the data were made with the sine function plotted on the bottom-right, so that is the correct answer in this case, but the algorithm doesn't know that, and to it the two solutions on the right both look equally good. The only way we can tell which solution is better is to test how well they generalise. We pick a value that is between our datapoints, use our curves to predict its value, and see which is better. This will tell us that the bottom-right curve is better in the example.

So one thing that our machine learning algorithms can do is interpolate between datapoints. This might not seem to be intelligent behaviour, or even very difficult in two dimensions, but it is rather harder in higher dimensional spaces. The same thing is true of the other thing that our algorithms will do, which is classification—grouping examples into different classes—which is discussed next. However, the algorithms are learning by our definition if they adapt so that their performance improves, and it is surprising how often real problems that we want to solve can be reduced to classification or regression problems.

1.4.2 Classification

The classification problem consists of taking input vectors and deciding which of N classes they belong to, based on training from exemplars of each class. The most important point about the classification problem is that it is discrete — each example belongs to precisely one class, and the set of classes covers the whole possible output space. These two constraints are not necessarily realistic; sometimes examples might belong partially to two different classes. There are fuzzy classifiers that try to solve this problem, but we won't be talking about them in this book. In addition, there are many places where we might not be able to categorise every possible input. For example, consider a vending machine, where we use a neural network to learn to recognise all the different coins. We train the classifier to recognise all New Zealand coins, but what if a British coin is put into the machine? In that case, the classifier will identify it as the New Zealand coin that is closest to it in appearance, but this is not really what is wanted: rather, the classifier should identify that it is not one of the coins it was trained on. This is called novelty detection. For now we'll assume that we will not receive inputs that we cannot classify accurately.

Let's consider how to set up a coin classifier. When the coin is pushed into the slot, the machine takes a few measurements of it. These could include the diameter, the weight, and possibly the shape, and are the features that will generate our input vector. In this case, our input vector will have three elements, each of which will be a number showing the measurement of that feature (choosing a number to represent the shape would involve an encoding, for example that 1=circle, 2=hexagon, etc.). Of course, there are many other features that we could measure. If our vending machine included an atomic absorption spectroscope, then we could estimate the density of the material and its composition, or if it had a camera, we could take a photograph of the coin and feed that image into the classifier. The question of which features to choose is not always an easy one. We don't want to use too many inputs, because that will make the training of the classifier take longer (and also, as the number of input dimensions grows, the number of datapoints required increases

FIGURE 1.4 The New Zealand coins.

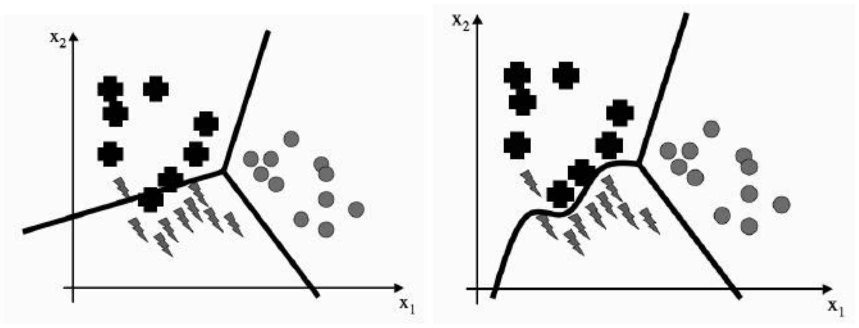

FIGURE 1.5 *Left:* A set of straight line decision boundaries for a classification problem. *Right:* An alternative set of decision boundaries that separate the plusses from the lightening strikes better, but requires a line that isn't straight.

faster; this is known as the curse of dimensionality and will be discussed in Section 2.1.2), but we need to make sure that we can reliably separate the classes based on those features. For example, if we tried to separate coins based only on colour, we wouldn't get very far, because the 20 ¢ and 50 ¢ coins are both silver and the $1 and $2 coins both bronze. However, if we use colour and diameter, we can do a pretty good job of the coin classification problem for NZ coins. There are some features that are entirely useless. For example, knowing that the coin is circular doesn't tell us anything about NZ coins, which are all circular (see Figure 1.4). In other countries, though, it could be very useful.

The methods of performing classification that we will see during this book are very different in the ways that they learn about the solution; in essence they aim to do the same thing: find decision boundaries that can be used to separate out the different classes. Given the features that are used as inputs to the classifier, we need to identify some values of those features that will enable us to decide which class the current input is in. Figure 1.5 shows a set of 2D inputs with three different classes shown, and two different decision boundaries; on the left they are straight lines, and are therefore simple, but don't categorise as well as the non-linear curve on the right.

Now that we have seen these two types of problem, let's take a look at the whole process of machine learning from the practitioner's viewpoint.

1.5 THE MACHINE LEARNING PROCESS

This section assumes that you have some problem that you are interested in using machine learning on, such as the coin classification that was described previously. It briefly examines the process by which machine learning algorithms can be selected, applied, and evaluated for the problem.

Data Collection and Preparation Throughout this book we will be in the fortunate position of having datasets readily available for downloading and using to test the algorithms. This is, of course, less commonly the case when the desire is to learn about some new problem, when either the data has to be collected from scratch, or at the very least, assembled and prepared. In fact, if the problem is completely new, so that appropriate data can be chosen, then this process should be merged with the next step of feature selection, so that only the required data is collected. This can typically be done by assembling a reasonably small dataset with all of the features that you believe might be useful, and experimenting with it before choosing the best features and collecting and analysing the full dataset.

Often the difficulty is that there is a large amount of data that *might* be relevant, but it is hard to collect, either because it requires many measurements to be taken, or because they are in a variety of places and formats, and merging it appropriately is difficult, as is ensuring that it is clean; that is, it does not have significant errors, missing data, etc.

For supervised learning, target data is also needed, which can require the involvement of experts in the relevant field and significant investments of time.

Finally, the quantity of data needs to be considered. Machine learning algorithms need significant amounts of data, preferably without too much noise, but with increased dataset size comes increased computational costs, and the sweet spot at which there is enough data without excessive computational overhead is generally impossible to predict.

Feature Selection An example of this part of the process was given in Section 1.4.2 when we looked at possible features that might be useful for coin recognition. It consists of identifying the features that are most useful for the problem under examination. This invariably requires prior knowledge of the problem and the data; our common sense was used in the coins example above to identify some potentially useful features and to exclude others.

As well as the identification of features that are useful for the learner, it is also necessary that the features can be collected without significant expense or time, and that they are robust to noise and other corruption of the data that may arise in the collection process.

Algorithm Choice Given the dataset, the choice of an appropriate algorithm (or algorithms) is what this book should be able to prepare you for, in that the knowledge of the underlying principles of each algorithm and examples of their use is precisely what is required for this.

Parameter and Model Selection For many of the algorithms there are parameters that have to be set manually, or that require experimentation to identify appropriate values. These requirements are discussed at the appropriate points of the book.

Training Given the dataset, algorithm, and parameters, training should be simply the use of computational resources in order to build a model of the data in order to predict the outputs on new data.

Evaluation Before a system can be deployed it needs to be tested and evaluated for accuracy on data that it was not trained on. This can often include a comparison with human experts in the field, and the selection of appropriate metrics for this comparison.

1.6 A NOTE ON PROGRAMMING

This book is aimed at helping you understand and use machine learning algorithms, and that means writing computer programs. The book contains algorithms in both pseudocode, and as fragments of Python programs based on NumPy (Appendix A provides an introduction to both Python and NumPy for the beginner), and the website provides complete working code for all of the algorithms.

Understanding how to use machine learning algorithms is fine in theory, but without testing the programs on data, and seeing what the parameters do, you won't get the complete picture. In general, writing the code for yourself is always the best way to check that you understand what the algorithm is doing, and finding the unexpected details.

Unfortunately, debugging machine learning code is even harder than general debugging – it is quite easy to make a program that compiles and runs, but just doesn't seem to actually learn. In that case, you need to start testing the program carefully. However, you can quickly get frustrated with the fact that, because so many of the algorithms are stochastic, the results are not repeatable anyway. This can be temporarily avoided by setting the random number seed, which has the effect of making the random number generator follow the same pattern each time, as can be seen in the following example of running code at the Python command line (marked as >>>), where the 10 numbers that appear after the seed is set are the same in both cases, and would carry on the same forever (there is more about the pseudo-random numbers that computers generate in Section 15.1.1):

```
>>> import numpy as np
>>> np.random.seed(4)
>>> np.random.rand(10)
array([ 0.96702984,  0.54723225,  0.97268436,  0.71481599,  0.69772882,
        0.2160895 ,  0.97627445,  0.00623026,  0.25298236,  0.43479153])
>>> np.random.rand(10)
array([ 0.77938292,  0.19768507,  0.86299324,  0.98340068,  0.16384224,
        0.59733394,  0.0089861 ,  0.38657128,  0.04416006,  0.95665297])
>>> np.random.seed(4)
>>> np.random.rand(10)
array([ 0.96702984,  0.54723225,  0.97268436,  0.71481599,  0.69772882,
        0.2160895 ,  0.97627445,  0.00623026,  0.25298236,  0.43479153])
```

This way, on each run the randomness will be avoided, and the parameters will all be the same.

Another thing that is useful is the use of 2D toy datasets, where you can plot things, since you can see whether or not something unexpected is going on. In addition, these datasets can be made very simple, such as separable by a straight line (we'll see more of this in Chapter 3) so that you can see whether it deals with simple cases, at least.

Another way to 'cheat' temporarily is to include the target as one of the inputs, so that the algorithm really has no excuse for getting the wrong answer.

Finally, having a reference program that works and that you can compare is also useful, and I hope that the code on the book website will help people get out of unexpected traps and strange errors.

1.7 A ROADMAP TO THE BOOK

As far as possible, this book works from general to specific and simple to complex, while keeping related concepts in nearby chapters. Given the focus on algorithms and encouraging the use of experimentation rather than starting from the underlying statistical concepts, the book starts with some older, and reasonably simple algorithms, which are examples of supervised learning.

Chapter 2 follows up many of the concepts in this introductory chapter in order to highlight some of the overarching ideas of machine learning and thus the data requirements of it, as well as providing some material on basic probability and statistics that will not be required by all readers, but is included for completeness.

Chapters 3, 4, and 5 follow the main historical sweep of supervised learning using neural networks, as well as introducing concepts such as interpolation. They are followed by chapters on dimensionality reduction (Chapter 6) and the use of probabilistic methods like the EM algorithm and nearest neighbour methods (Chapter 7). The idea of optimal decision boundaries and kernel methods are introduced in Chapter 8, which focuses on the Support Vector Machine and related algorithms.

One of the underlying methods for many of the preceding algorithms, optimisation, is surveyed briefly in Chapter 9, which then returns to some of the material in Chapter 4 to consider the Multi-layer Perceptron purely from the point of view of optimisation. The chapter then continues by considering search as the discrete analogue of optimisation. This leads naturally into evolutionary learning including genetic algorithms (Chapter 10), reinforcement learning (Chapter 11), and tree-based learners (Chapter 12) which are search-based methods. Methods to combine the predictions of many learners, which are often trees, are described in Chapter 13.

The important topic of unsupervised learning is considered in Chapter 14, which focuses on the Self-Organising Feature Map; many unsupervised learning algorithms are also presented in Chapter 6.

The remaining four chapters primarily describe more modern, and statistically based, approaches to machine learning, although not all of the algorithms are completely new: following an introduction to Markov Chain Monte Carlo techniques in Chapter 15 the area of Graphical Models is surveyed, with comparatively old algorithms such as the Hidden Markov Model and Kalman Filter being included along with particle filters and Bayesian networks. The ideas behind Deep Belief Networks are given in Chapter 17, starting from the historical idea of symmetric networks with the Hopfield network. An introduction to Gaussian Processes is given in Chapter 18.

Finally, an introduction to Python and NumPy is given in Appendix A, which should be sufficient to enable readers to follow the code descriptions provided in the book and use the code supplied on the book website, assuming that they have some programming experience in any programming language.

I would suggest that Chapters 2 to 4 contain enough introductory material to be essential for anybody looking for an introduction to machine learning ideas. For an introductory one semester course I would follow them with Chapters 6 to 8, and then use the second half of Chapter 9 to introduce Chapters 10 and 11, and then Chapter 14.

A more advanced course would certainly take in Chapters 13 and 15 to 18 along with the optimisation material in Chapter 9.

I have attempted to make the material reasonably self-contained, with the relevant mathematical ideas either included in the text at the appropriate point, or with a reference to where that material is covered. This means that the reader with some prior knowledge will certainly find some parts can be safely ignored or skimmed without loss.

FURTHER READING

For a different (more statistical and example-based) take on machine learning, look at:

- Chapter 1 of T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, 2nd edition, Springer, Berlin, Germany, 2008.

Other texts that provide alternative views of similar material include:

- Chapter 1 of R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*, 2nd edition, Wiley-Interscience, New York, USA, 2001.
- Chapter 1 of S. Haykin. *Neural Networks: A Comprehensive Foundation*, 2nd edition, Prentice-Hall, New Jersey, USA, 1999.