Machine Learning in Business:

An Introduction to the World of Data Science

Machine Learning in Business:

An Introduction to the World of Data Science

Second Edition

John C. Hull

University Professor
Joseph L. Rotman School of Management
University of Toronto

Second Printing

ISBN: 9798644074372

To my students

Contents

Preface

This book is based on my experience teaching introductory courses on machine learning to business school students and executive groups. The purpose of the material is not to convert the reader into a data scientist. Instead, it is to give the reader an understanding of the tools used by data scientists and how they can further the objectives of an organization. The second edition improves the presentation of material and contains three new chapters.

Most students recognize that they need some knowledge of machine learning to survive in a world where jobs will be increasingly impacted by it. Today, all executives need to know how to use computers. Tomorrow, all executives will need to be comfortable managing large data sets and working with data science professionals to improve their productivity.

I have used no matrix or vector algebra and no calculus in this book. Although these areas of study can help specialists, it has been my experience that most business school students and most executives are not comfortable with them.

The book explains the most popular algorithms used by data scientists. This will enable the reader to assess their strengths and weaknesses for a particular situation and work productively with data science professionals. The algorithms are illustrated with a number of different data sets, which can be downloaded from my website:

www-2.rotman.utoronto.ca/~hull

Both Excel worksheets and Python code accompany the data sets. Virtually all my students are comfortable with Excel before taking my courses. I insist that all become comfortable with Python as well. This is not a hard sell. Students recognize that coding skills have become a necessary prerequisite for many jobs in business.

Several hundred PowerPoint slides can be downloaded from my website. Instructors who choose to adopt the book are welcome to adapt the slides to meet their own needs.

A number of people have helped me move this book to a second edition. I would particularly like to thank Emilio Barone, Jacky Chen, Peter Hull, Raymond Kan, Eddie Mizzi, and Jun Yuan, who made many suggestions for improving the material. I am grateful to Jay Cao, Jeff Li, and Niti Mishra who worked on some of the Python code that accompanies the book. I would also like to thank Rotman's FinHub center, the TD bank, and the Global Risk Institute in Financial Services for providing funding for the development of research and teaching materials in machine learning and financial innovation. Peter Christoffersen (prior to his untimely death in 2018) and Andreas Park have been great colleagues at FinHub and provided much of the inspiration for the book.

I welcome comments on the book from readers. My email address is hull@rotman.utoronto.ca.

John Hull

About the Author

John Hull is a University Professor at the Joseph L. Rotman School of Management, University of Toronto. Prior to writing this book, he wrote three best-selling books in the derivatives and risk management area. His books have an applied focus and he is proud that they sell equally well in the practitioner and college markets. He is academic director of FinHub, Rotman's Financial Innovation Lab, which carries out research and develops educational material in all aspects of financial innovation. He has consulted for many companies throughout the world and has won many teaching awards, including University of Toronto's prestigious Northrop Frye award.

Chapter 1

Introduction

Machine learning is becoming an increasingly important tool in business—so much so that almost all employees are likely to be impacted by it in one way or another over the next few years. Machine learning is concerned with using large data sets to learn the relationships between variables, make predictions, and take decisions in a changing environment.

The data available for machine learning is growing exponentially. It is estimated that in any two-year period we generate nine times as much data as existed at the beginning of the two years.[1] Companies now have more information than ever before about their customers and their purchasing habits. Hedge funds and pension plans can collect large amounts of data and opinions about companies they invest in. Advances in computer processing speeds and reductions in data storage costs allow us process this data and reach conclusions in ways that were simply not possible in the past.

Machine learning is a branch of artificial intelligence (AI). AI is concerned with developing ways in which machines can imitate human intelligence, possibly improving on it. Machine learning involves the creation of intelligence by learning from large volumes of data. It is arguably

[1] For discussion of this see: www.mediapost.com/publications/article/291358/90-of-todays-data-created-in-two-years.html

the most exciting development within AI and one that has the potential to transform virtually all aspects of a business.[2]

What are the advantages for society of replacing human decision making by machines? One advantage is speed. Machines can process data and come to a conclusion much faster than humans. The results produced by a machine are consistent and easily replicated on other machines. By contrast, humans occasionally behave erratically and training a human for a task can be quite time consuming and expensive.

To explain how machine learning differs from other AI approaches consider the simple task of programming a computer to play tic tac toe (also known as noughts and crosses). One approach would be to provide the computer with a look-up table listing the positions that can arise and the move that would be made by an expert human player in each of those positions. Another would be to create for the computer a large number of games (e.g., by arranging for the computer to play against itself thousands of times) and let it learn the best move. The second approach is an application of machine learning. Either approach can be successfully used for a simple game such as tic tac toe. Machine learning approaches have been shown to work well for more complicated games such as chess and Go where the first approach is clearly not possible.

A good illustration of the power of machine learning is provided by language translation. How can a computer be programmed to translate between two languages, say from English to French? One idea is to give the computer an English to French dictionary and program it to translate word-by-word. Unfortunately, this produces very poor results. A natural extension of this idea is to develop a look up table for translating phrases rather than individual words. The results from this are an improvement, but still far from perfect. Google has pioneered a better approach using machine learning. This was announced in November 2016 and is known as "Google Neural Machine Translation" (GNMT).[3] A computer is given a large volume of material in English together with the French translation. It learns from that material and develops its own (quite complex) translation rules. The results from this have been a big improvement over previous approaches.

Data science is the field that includes machine learning but is sometimes considered to be somewhat broader including such tasks as the setting of objectives, implementing systems, and communicating with

[2] Some organizations now use the terms "machine learning" and "artificial intelligence" interchangeably.

[3] See https://arxiv.org/pdf/1609.08144.pdf for an explanation of GNMT by the Google research team.

stakeholders.[4] We will consider the terms "machine learning" and "data science" to be interchangeable in this book. This is because it is difficult to see how machine learning specialists can be effective in business if they do not get involved in working toward the objectives of their employers.

Machine learning or data science can be portrayed as the new world of statistics. Traditionally, statistics has been concerned with such topics as probability distributions, confidence intervals, significance tests, and linear regression. A knowledge of these topics is important, but we are now able to learn from large data sets in ways that were not possible before. For example:

- We can develop non-linear models for forecasting and improved decision making.
- We can search for patterns in data to improve a company's understanding of its customers and the environment in which it operates.
- We can develop decision rules where we are interacting with a changing environment.

As mentioned earlier, these applications of machine learning are now possible because of increases in computer processing speeds, reductions in data storage costs, and the increasing amounts of data that are becoming available.

When a statistician or econometrician dabbles in machine learning the terminology is liable to seem strange at first. For example, statisticians and econometricians talk about independent variables and dependent variables while decision scientists talk about features and targets. The terminology of data science will be explained as the book progresses and a glossary of terms is provided at the end.

1.1 This Book and the Ancillary Material

This book is designed to provide readers with the knowledge to enable them to work effectively with data science professionals. It will not convert the reader into a data scientist, but it is hoped that the book will

[4] See, for example, H. Bowne-Anderson, "What data scientists really do, according to 35 data scientists," *Harvard Business Review*, August 2018: https://hbr.org/2018/08/what-data-scientists-really-do-according-to-35-data-scientists

inspire some readers to learn more and develop their abilities in this area. Data science may well prove to be the most rewarding and exciting profession in the 21st century.

To use machine learning effectively you have to understand how the underlying algorithms work. It is tempting to learn a programming language such as Python and apply various packages to your data without really understanding what the packages are doing or even how the results should be interpreted. This would be a bit like a finance specialist using the Black–Scholes–Merton model to value options without understanding where it comes from or its limitations.

The objective of this book is to explain the algorithms underlying machine learning so that the results from using the algorithms can be assessed knowledgeably. Anyone who is serious about using machine learning will want to learn a language such as Python for which many packages have been developed. This book takes the unusual approach of using both Excel and Python to provide backup material. This is because it is anticipated that some readers will, at least initially, be much more comfortable with Excel than with Python.

The backup material can be found on the author's website:

www-2.rotman.utoronto.ca/~hull

Readers can start by focusing on the Excel worksheets and then move to Python as they become more comfortable with it. Python will enable them use machine learning packages, handle data sets that are too large for Excel, and benefit from Python's faster processing speeds.

1.2 Types of Machine Learning Models

There are four main categories of machine learning models

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning

Supervised learning is concerned with using data to make predictions. In the next section, we will show how a simple regression model can be used to predict salaries. This is an example of supervised learning. In Chapter 3, we will consider how a similar model can be used to predict house prices. We can distinguish between supervised learning models that are used to predict a variable that can take a continuum of values

(such as an individual's salary or the price of a house) and models that are used for classification. Classification models are very common in machine learning. As an example, we will later look at an application of machine learning where potential borrowers are classified as acceptable or unacceptable credit risks.

Unsupervised learning is concerned with recognizing patterns in data. The main objective is not to forecast a particular variable. Rather it is to understand the environment represented by the data better. Consider a company that markets a range of products to consumers. Data on consumer purchases could be used to determine the characteristics of the customers who buy different products. This in turn could influence the way the products are advertised. As we will see in Chapter 2, clustering is the main tool used in unsupervised learning.

The data for supervised learning contains what are referred to as *features* and *labels*. The labels are the values of the *target* that is to be predicted. The features are the variables from which the predictions are to be made. For example, when predicting the price of a house the features could be the square feet of living space, the number of bedrooms, the number of bathrooms, the size of the garage, whether the basement is finished, and so on. The label would be the house price. The data for unsupervised learning consists of features but no labels because the model is being used to identify patterns, not to forecast something. We could use an unsupervised learning model to understand the houses that exist in a certain neighborhood without trying to predict prices. We might find that there is a cluster of houses with 1,500 to 2,000 square feet of living space, three bedrooms, and a one-car garage and another cluster of houses with 5,000 to 6,000 square feet of living area, six bedrooms, and a two-car garage.

Semi-supervised learning is a cross between supervised and unsupervised learning. It arises when we are trying to predict something and we have some data with labels (i.e., values for the target) and some (usually much more) unlabeled data. It might be thought that the unlabeled data is useless, but this is not necessarily the case. The unlabeled data can be used in conjunction with the labeled data to produce clusters which help prediction. For example, suppose we are interested in predicting whether a customer will purchase a particular product from features such as age, income level, and so on. Suppose further that we have a small amount of labeled data (i.e., data which indicates the features of customers as well as whether they bought or did not buy the product) and a much larger amount of unlabeled data (i.e., data which indicates the features of potential customers, but does not indicate whether they bought the product). We can apply unsupervised learning

to use the features to cluster potential customers. Imagine a simple situation where:

- There are two clusters, A and B, in the full data set.
- The purchasers from the labeled data all correspond to points in Cluster A while the non-purchasers from the labeled data all correspond to points in the other Cluster B.

We might reasonably classify all individuals in Cluster A as buyers and all individuals in Cluster B as non-buyers.

Human beings use semi-supervised learning. Imagine that you do not know the names "cat" and "dog," but are observant. You notice two distinct clusters of domestic pets in your neighborhood. Finally someone points at two particular animals and tells you one is a cat while the other is a dog. You will then have no difficulty in using semi-supervised learning to apply the labels to all the other animals you have seen. If humans use semi-supervised learning in this way, it should come as no surprise that machines can do so as well. Many machine learning algorithms are based on studying the ways our brains process data.

The final type of machine learning, reinforcement learning, is concerned with situations where a series of decisions is to be taken. The environment is typically changing in an uncertain way as the decisions are being taken. Driverless cars use reinforcement learning algorithms. The algorithms underlie the programs mentioned earlier for playing games such as Go and chess. They are also used for some trading and hedging decisions. We will discuss reinforcement learning in Chapter 7.

1.3 Validation and Testing

When a data set is used for forecasting or determining a decision strategy, there is a danger that the machine learning model will work well for the data set, but will not generalize well to other data. An obvious point is that it is important that the data used in a machine learning model be representative of the situations to which the model is to be applied. For example, using data for a region where customers have a high income to predict the national sales for a product is likely to give biased results.

As statisticians have realized for a long time, it is also important to test a model out-of-sample. By this we mean that the model should be tested on data that is different from the sample data used to determine the parameters of the model.

Data scientists are typically not just interested in testing one model. They typically try several different models, choose between them, and then test the accuracy of the chosen model. For this, they need three data sets:

- a training set
- a validation set
- a test set

The training set is used to determine the parameters of the models that are under consideration. The validation set is used to determine how well each of the models generalizes to a different data set. The test set is held back to provide a measure of the accuracy of the chosen model.

We will illustrate this with a simple example. Suppose that we are interested in predicting the salaries of people working in a particular profession in a certain part of the United States from their age. We collect data on a random sample of 30 individuals. (This is a very small data set created to provide a simple example. The data sets used in machine learning are many times larger than this.) The first ten observations (referred to in machine learning as *instances*) will be used to form the training set. The next ten observations will be used for form the validation set and the final ten observations will be used to form the test set.

The training set is shown in Table 1.1 and plotted in Figure 1.1. It is tempting to choose a model that fits the training set really well. Some experimentation shows that a polynomial of degree five does this. This is the model:

$$Y = a + b_1 X + b_2 X^2 + b_3 X^3 + b_4 X^4 + b_5 X^5$$

where Y is salary and X is age. The result of fitting the polynomial to the data is shown in Figure 1.2. Details of all analyses carried out, are in

www-2.rotman.utoronto.ca/~hull

The model provides a good fit to the data. The standard deviation of the difference between the salary given by the model and the actual salary for the ten individuals in the training data set, which is referred to as the *root-mean-squared error* (*rmse*), is $12,902. However, common sense would suggest that we may have over-fitted the data. (This is because the curve in Figure 1.2 seems unrealistic. It declines, increases, declines, and then increases again as age increases.) We need to check the model out-of-sample. To use the language of data science, we need

to determine whether the model generalizes well to a validation data set that is different from the training set in Table 1.1.

Table 1.1 The training data set: salaries for a random sample of ten people working in a particular profession in a certain area.

Age (years)	*Salary ($)*
25	135,000
55	260,000
27	105,000
35	220,000
60	240,000
65	265,000
45	270,000
40	300,000
50	265,000
30	105,000

Figure 1.1 Scatter plot of the training data set in Table 1.1

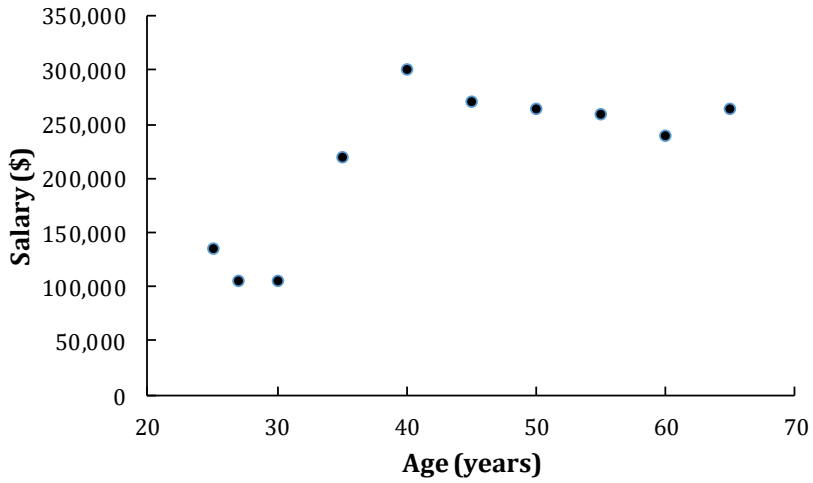
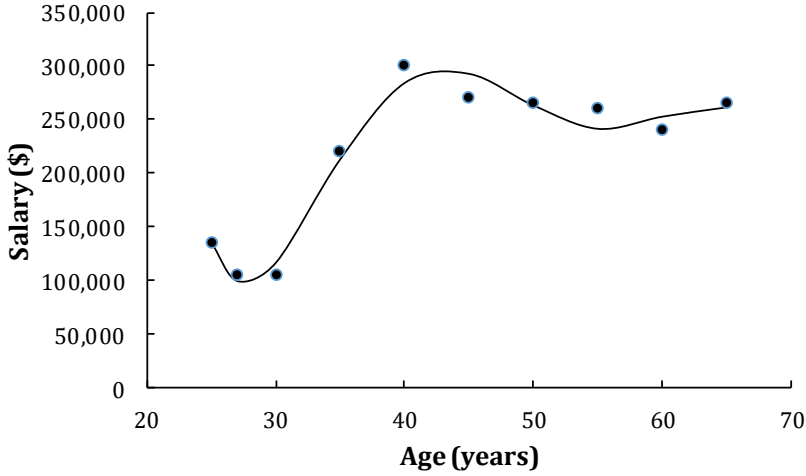


Figure 1.2 Result of fitting a polynomial of degree 5 to the data in Table 1.1 and Figure 1.1 (see Salary vs. Age Excel file)

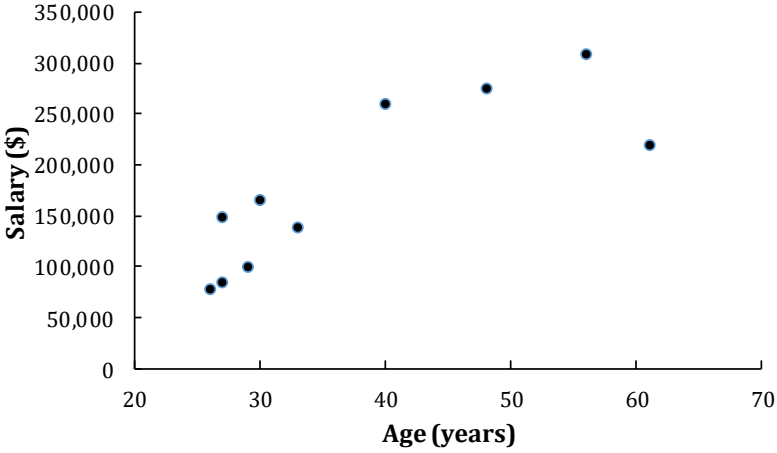


The validation set is shown in Table 1.2. The scatter plot for this data is in Figure 1.3. When we use the model in Figure 1.2 for this data, we find that the root mean square error (rmse) is about $38,794, much higher than the $12,902 we obtained using the training data set in Table 1.1. This is a clear indication that the model in Figure 1.2 is over-fitting: it does not generalize well to new data.

Table 1.2 The validation data set

Age (years)	*Salary ($)*
30	166,000
26	78,000
58	310,000
29	100,000
40	260,000
27	150,000
33	140,000
61	220,000
27	86,000
48	276,000

Figure 1.3 Scatter plot for data in Table 1.2



The natural next step is to look for a simpler model. The scatter plot in Figure 1.1 suggests that a quadratic model might be appropriate. This model is:

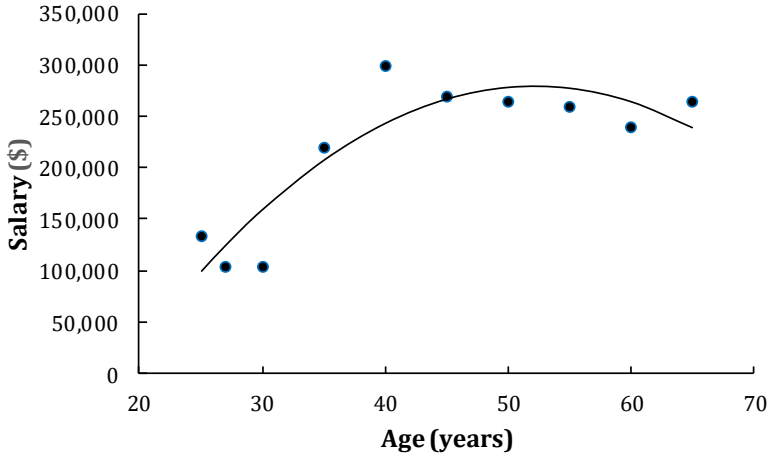
$$Y = a + b_1 X + b_2 X^2$$

i.e., a polynomial of degree two.

The best-fit quadratic model together with the training data set from Figure 1.1 is shown in Figure 1.4. The fit to the training set is of course not as good as the model in Figure 1.2. The standard deviation of the error is $32,932. However, the model generalizes to new data reasonably well. The standard deviation of the errors given by the quadratic model for the validation data set in Table 1.2 and Figure 1.3 is $33,554, only a little worse than the $32,932 for the training data. The quadratic model therefore generalizes better than the more elaborate model in Figure 1.2.

The model in Figure 1.4 is simpler than the model in Figure 1.2 and generalizes well to the validation set. However, this does not mean that simpler models are always better than more complex models. In the case of the data we are considering, we could use a linear model. This would lead to the predictions in Figure 1.5.

Figure 1.4 Result of fitting a quadratic model to the data in Table 1.1 and Figure 1.1 (see Salary vs. Age Excel file)



Visually it can be seen that this model does not capture the decline in salaries as individuals age beyond 50. This observation is confirmed by the standard deviation of the error for the training data set, which is $49,731, much worse than that for the quadratic model.

Figure 1.5 Result of fitting a linear model to training data (see Salary vs. Age Excel file)

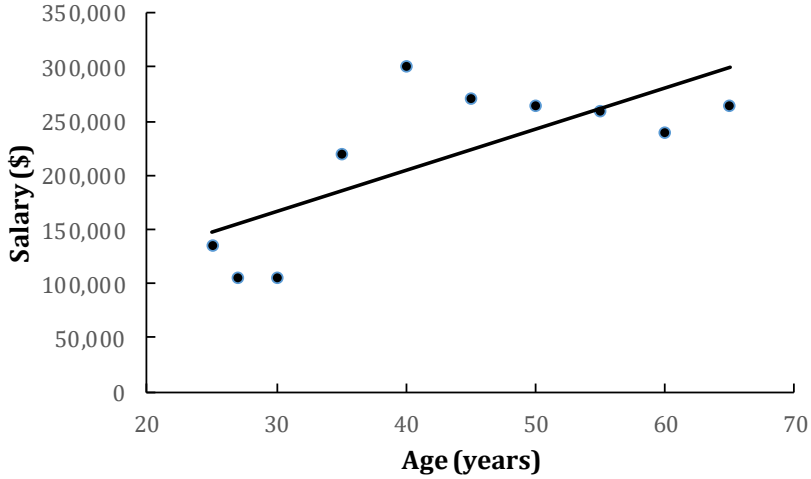


Table 1.3 summarizes the root mean square errors given by the three models we have considered. Note that both the linear model and the quadratic model generalize well to the validation data set, but the quadratic model is preferred because it is more accurate. By contrast, the five-degree polynomial model does not generalize well. It over-fits the training set while the linear model under-fits the training set.

Table 1.3 Root mean square errors (see Excel file)

	Polynomial of degree 5	*Quadratic model*	*Linear model*
Training set (Table 1.1)	12, 902	32,932	49,731
Validation set (Table 1.2)	38,794	33,554	49,990

How accurate is the quadratic model? We could rely on the results from the validation set. But we used the validation set to help choose the best model and so it may overstate the accuracy of the model. We therefore use the test data set to produce an accuracy measure. This data set has played no role in analyses so far.

Suppose the test data set results are as shown in Table 1.4. The root mean squared error for the test set is $34,273. When information about the performance of the chosen model is presented, it should be based on results for the test data set, not on those for the validation set or the training set.

How should the balance between over-fitting and under-fitting be achieved? This is an important issue in machine learning. Some machine learning algorithms, such as neural networks (see Chapter 6), can involve a very large number of parameters. It is then easy to over-fit, even when the training data set is large.

Based on the simple example we have looked at, a rule of thumb would seem to be as follows:

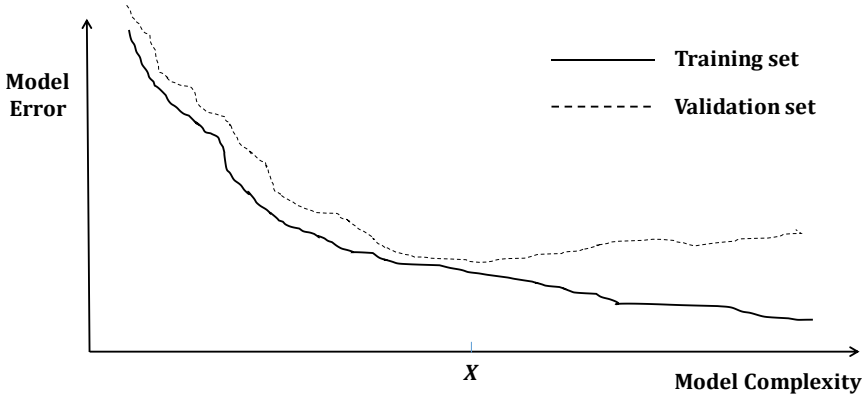
The complexity of the model should be increased until out-of-sample tests indicate that it does not generalize well.

Table 1.4 Errors when quadratic model is applied to the test set

Age (years)	*Salary ($)*	*Predicted salary ($)*	*Error ($)*
26	110,000	113,172	−3,172
52	278,000	279,589	−1,589
38	314,000	230,852	+83,148
60	302,000	264,620	+37,380
64	261,000	245,457	+15,543
41	227,000	249325	−22,325
34	200,000	199,411	+589
46	233,000	270,380	−37,380
57	311,000	273,883	−37,117
55	298,000	277,625	+20,375

This rule is illustrated in Figure 1.6. The figure assumes that there is a continuum of models that get progressively more complex. For each model, we calculate a measure of the model's error, such as root mean square error, for both the training set and the validation set. When the complexity of the model is less than *X*, the model generalizes well: the error of the model for the validation set is only a little more than that for the training set. As model complexity is increased beyond *X*, the errors for the validation set start to increase.

Figure 1.6 Errors of a model for the training set and the validation set.



The best model is the one with model complexity *X*. This is because that model has the lowest error for the validation set. A further increase in complexity lowers errors for the training set but increases them for the validation set, which is a clear indication of over-fitting.

Finding the right balance between under-fitting and over-fitting is referred to as the *bias-variance trade-off* in machine learning. The bias is the error due the assumptions in the model that cause it to miss relevant relations. The variance is the error due to the model over-fitting by reflecting random noise in the training set.

To summarize the points we have made:

- The training set is used to develop alternative models.
- The validation set is used to investigate how well the models generalize to new data and to choose between the models.
- The test set is kept back and is used as an out-of-sample test of the accuracy of the chosen model at the end of the analysis.

In the simple example we have looked at, the training set, validation set, and test set had equal numbers of observations. In a typical machine learning application much more data is available and at least 60% of it is allocated to the training set while 10% to 20% is allocated to each of the validation set and the test set.

It is important to emphasize that the data sets in machine learning involve many more observations that the baby data set we have used in this section. (Ten observations are obviously insufficient to reliably learn a relationship.) However, the baby data set does provide a simple illustration of the bias-variance trade-off.

1.4 Data Cleaning

Data cleaning is a very important, if not terribly exciting, aspect of machine learning. It has been estimated that data scientists spend 80% of their time on collecting and cleaning data.[5] Large data sets typically have issues that need to be fixed. Good data cleaning can make all the difference between successful and unsuccessful machine learning. The

[5] See https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/#2f8970aa6f63 for a discussion of this.

expression "garbage-in, garbage-out" applies just as much to machine learning as to other analyses.

At this stage, it is appropriate to point out that there are two types of data: numerical and categorical. Numerical data consists of numbers. Categorical data is data which can fall into a number of different categories. For example, data to predict a house price might categorize driveways as asphalt, concrete, grass, etc. As we will see in Chapter 3, categorical data must be converted to numbers for the purposes of analysis.

We now list some data cleaning issues and how they can be handled.

Inconsistent Recording

Either numerical or categorical data can be subject to inconsistent recording. For example, numerical data for the square footage of a house might be input manually as 3300, 3,300, 3,300 ft, or 3300+, and so on. It is necessary to inspect the data to determine variations and decide the best approach to cleaning. Categorical data might list the driveway as "asphalt", "Asphalt", or even "aphalt." The simplest approach here is to list the alternatives that have been input for a particular feature and merge them as appropriate.

Unwanted Observations

If you are developing a model to predict house prices in a certain area, some of your data might refer to the prices of apartments or to the prices of houses that are not in the area of interest. It is important to find a way of identifying this data and removing it before any analysis is attempted.

Duplicate Observations

When data is merged from several different sources or several different people have been involved in creating a data set there are liable to be duplicate observations. These can bias results. It is therefore important to use a search algorithm to identify and remove duplicates as far as possible.

Outliers

In the case of numerical data, outliers can be identified by either plotting data or searching for data that is, say, six standard deviations away from the mean. Sometimes it is clear that the outlier is a typo. For example, if the square footage of a house with three bedrooms is input

as 33,000, it is almost certainly a typo and should probably be 3,300. However, outliers should be removed only if there is a good reason for doing so. Unusually large or small values for features or targets, if correct, are likely to contain useful information. The impact of outliers on the results of machine learning depends on the model being used. Outliers tend to have a big effect on regression models such as those considered in Chapter 3. Other models, such as those involving decision trees (which will be explained in Chapter 4) are less influenced by outliers.

Missing Data

In any large data set there are likely to be missing data values. A simple approach is to remove data with missing values for one or more features. But this is probably undesirable because it reduces the sample size and may create biases. In the case of categorical data, a simple solution is to create a new category titled "Missing." In the case of numerical data, one approach is to replace the missing data by the mean or median of the non-missing data values. For example, if the square footage of a house is missing and we calculate the median square footage for the houses for which this data is available to be 3,500, we could populate all the missing values with 3,500. More sophisticated approaches can involve regressing the target against non-missing values and then using the results to populate missing values. Sometimes it is reasonable to assume that data is missing at random and sometimes the very fact that data is missing is itself informative. In the latter case it can be desirable to create a new indicator variable which is zero if the data is present and one if it is missing.

1.5 Bayes' Theorem

Sometimes in machine learning we are interested in estimating the probability of an outcome from data. The outcome might be a customer defaulting on a loan or a transaction proving to be fraudulent. Typically there is an initial probability of the outcome. When data is received, the probability is updated to be a probability conditional on the data. A result known as Bayes' theorem is sometimes useful for calculating conditional probabilities.

Thomas Bayes discovered Bayes' theorem in about 1760. We will write $P(X)$ as the probability of event X happening and $P(Y|X)$ as the probability of event Y happening conditional that event X has happened. Bayes' theorem states that

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{1.1}$$

The proof of Bayes' theorem is straightforward. From the meaning of conditional probabilities:

$$P(Y|X) = \frac{P(X \text{ and } Y)}{P(X)}$$

and

$$P(X|Y) = \frac{P(X \text{ and } Y)}{P(Y)}$$

Substituting for $P(X \text{ and } Y)$ from the second of these equations into the first leads to the Bayes' theorem result in equation (1.1).

For an application of Bayes' theorem, suppose that a bank is trying to identify customers who are attempting to do fraudulent transactions at branches. It observes that 90% of fraudulent transactions involve over $100,000 and occur between 4pm and 5pm. In total, only 1% of transactions are fraudulent and 3% of all transactions involve over $100,000 and occur between 4pm and 5pm.

In this case we define:

X: transaction occurring between 4pm and 5pm involving over $100,000
Y: fraudulent transaction

We know that $P(Y) = 0.01$, $P(X|Y) = 0.9$, and $P(X) = 0.03$. From Bayes' theorem:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{0.9 \times 0.01}{0.03} = 0.3$$

The probability of a random transaction being fraudulent transaction is only 1%. But when it is known that the transaction is undertaken between 4pm and 5pm and involves over $100,000, Bayes' theorem shows that this probability should be updated to 30%. The implications of this are clear. If the bank has an on-line approval system for transactions, it should not approve transactions between 4pm and 5pm where over $100,000 is involved without further investigation.

Effectively what Bayes' theorem allows one to do is to invert the conditionality when probabilities are measured. Sometimes this prod-

uces counterintuitive results. Suppose that a test for a certain disease is "99% accurate." By this it is usually meant that, when a person has the disease, it gives a positive result (i.e., it predicts that the person has the disease) 99% of the time. We also assume that, when a person does not have the disease, it gives a negative result (i.e., it predicts that the person does not have the disease) 99% of the time.[6] Suppose that the disease is rare so that the (unconditional) probability of an individual having the disease is one in 10,000. If you are tested positive, what is the probability that you have the disease?

A natural response to this question is 99%. (After all, the test is 99% accurate.) However, this is confusing the conditionality. Suppose that X indicates that the test is positive and Y indicates that a person has the disease. We are interested in $P(Y|X)$. We know that $P(X|Y) = 0.99$. We also know that $P(Y) = 0.0001$. Let us extend our notation so that $\bar{X}$ indicates that the test result is negative and $\bar{Y}$ indicates that the person does not have the disease. We also know that

$$P(\bar{Y}) = 0.9999$$

and

$$P(\bar{X}|\bar{Y}) = 0.99$$

Because either X or $\bar{X}$ is true $P(\bar{X}|\bar{Y}) + P(X|\bar{Y}) = 1$ so that

$$P(X|\bar{Y}) = 0.01$$

and we can calculate the probability of a positive test result as

$$\begin{aligned} P(X) &= P(X|Y)P(Y) + P(X|\bar{Y})P(\bar{Y}) \\ &= 0.99 \times 0.0001 + 0.01 \times 0.9999 = 0.0101 \end{aligned}$$

Using the Bayes' theorem result in equation (1.1),

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{0.99 \times 0.0001}{0.0101} = 0.0098$$

[6] It does not have to be the case that that the accuracy measure is the same for positive and negative test results.

This shows that there is a less than 1% chance that you have the disease if you get a positive test result. The test result increases the probability that you have the disease from the unconditional 0.0001 by a factor of about 98 but the probability is still low. The key point here is that "accuracy" is defined as the probability of getting the right result conditional that a person has the disease, not the other way round.

We will use Bayes' theorem to explain a popular tool known as the naïve Bayes classifier in Chapter 4 and use it in natural language processing in Chapter 8.

Summary

Machine learning is a branch of artificial intelligence concerned with learning from big data sets. It involves developing algorithms to make predictions, cluster data, or develop a strategy for making a series of decisions.

Statistical analysis has traditionally involved forming a hypothesis (without looking at data) and then testing the hypothesis with data. Machine learning is different. There is no hypothesis. The model is derived entirely from data.

Before using any machine learning algorithms, it is important to clean the data. The features constituting the data are either numerical or categorical. In either case there may be inconsistencies in the way the data has been input. These inconsistencies need to be identified and corrected. Some observations may be irrelevant to the task at hand and should be removed. The data should be checked for duplicate observations that can create biases. Outliers that are clearly a result of input errors should be removed. Finally, missing data must be dealt with in a way that will not bias the results.

Bayes' theorem is a result that is sometimes used when it is necessary to quantify uncertainty. It is a way of inverting conditionality. Suppose we are interested in knowing the probability of an event Y occurring and can observe whether another related event X happens. Suppose also that from experience we know the probability of X happening when Y happens. Bayes' theorem allows us to calculate the probability of Y conditional on X.

As mentioned in this chapter, machine learning has its own terminology which is different from that traditionally used in statistics. We close this chapter by summarizing the new terminology that has been introduced so far. A *feature* is a variable on which we have observations. Each observation is referred to as an *instance.* A *target* is a variable about which we want to make predictions. *Labels* are observations on the target. *Supervised learning* is a type of machine learning where we use data on the features and targets to predict the target for new data. *Unsupervised learning* is where we attempt to find patterns in data to help us understand its structure. (There is no target and therefore there are no labels in unsupervised learning). *Semi-supervised* learning involves making predictions about a target from data that is partly labeled (so that values of the target are provided) and partly unlabeled (so that values of the target are not provided). Finally, *reinforcement learning* is concerned with producing algorithms for sequential decision making where the decision maker is interacting with a changing environment. Other terminology will be introduced as the book progresses.

SHORT CONCEPT QUESTIONS

1.1 What is the difference between machine learning and artificial intelligence?
1.2 Explain two types of predictions that are made in supervised learning.
1.3 When is unsupervised learning appropriate?
1.4 When is reinforcement learning appropriate?
1.5 When is semi-supervised learning appropriate?
1.6 How can you tell whether a machine learning model is overfitting data?
1.7 Explain the role of the validation data set and the test data set.
1.8 What is meant by a categorical feature?
1.9 What is meant by the bias-variance trade-off? Does the linear model in Figure 1.5 give a bias error or a variance error? Does the fifth-order-polynomial model in Figure 1.2 give a bias error or a variance error?
1.10 List five different types of data cleaning.
1.11 "Bayes' theorem allows one to invert the conditionality." What is meant by this statement?

EXERCISES

1.12 How well do polynomials of degree 3 and 4 work for the data on salary vs. age in Section 1.3.? Consider whether the best fit model generalizes well from the training set to the validation set.

1.13 Suppose that 25% of emails are spam and it is found that spam contains a particular word 40% of the time. Overall only 12.5% of the emails contain the word. What is the probability of an email being spam when it contains the word?

Chapter 2

Unsupervised Learning

As explained in Chapter 1, unsupervised learning is concerned with identifying patterns in data. The immediate objective is not to predict the value of a target variable. Rather it is to understand the structure of data and find clusters. This is a useful exercise for many businesses. Banks, for example, often use unsupervised learning to cluster their customers so that they can communicate with them better and provide an improved level of service. One cluster might be young couples who are likely to want a mortgage soon. Another might be what are termed HENRYs (High Earners, Not Rich Yet). These are families earning between $250,000 and $500,000 who may be in the market for wealth management services.

This chapter explains a popular clustering procedure known as the *k-means algorithm*. It illustrates the algorithm by clustering countries according to their risk from the perspective of a foreign investor. Data on 122 countries and four features are used. The features are the real GDP growth rate, a corruption index, a peace index, and a legal risk index. The chapter then mentions some alternative algorithms and explains principal components analysis, which is a useful tool for both super-vised and unsupervised learning.

2.1 Feature Scaling

Before covering clustering algorithms, it is appropriate to discuss what is known as *feature scaling*. This is also referred to as the *normalization* or *standardization* of data. It is a necessary first step to for many machine learning algorithms, including the *k*-means algorithm. The purpose of feature scaling is to ensure that the features are given equal importance in an algorithm. Suppose for example that we are clustering men according to two features: height in inches and weight in pounds. Heights might range from 60 to 80 inches while weights range from 100 to 350 pounds. Without feature scaling, the two features will not be treated with equal importance because the range of heights is much less than the range of weights (20 inches vs 250 pounds).

One approach to feature scaling is to calculate the mean and standard deviation of each feature and scale observations on the feature by subtracting the mean and dividing by the standard deviation. If V is a feature value for a particular observation,

$$\text{Scaled Feature Value} = \frac{V - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation calculated from observations on the feature. This method of feature scaling is sometimes referred to as *Z-score scaling* or *Z-score normalization.* The scaled features have means equal to zero and standard deviations equal to one. If we want a particular feature to have more effect than other features in determining cluster separation, we could scale it so that its standard deviation is greater than one.

An alternative approach to feature scaling is to subtract the minimum feature value and divide by the difference between the maximum and minimum values so that:

$$\text{Scaled Feature Value} = \frac{V - \min}{\max - \min}$$

where max and min denote the maximum and minimum feature values. This is referred to as *min-max scaling*. The scaled feature values lie between zero and one.

Scaling using the *Z*-score method is usually preferred because it is less sensitive to extreme values, but it can make sense to use min-max scaling when features have been measured on bounded scales. In our description of the *k*-means algorithm in the rest of this chapter, we assume that feature values have been scaled using one of the two methods we have described.

The usual approach is to use the training data set to define the scaling parameters (i.e., the means and standard deviations of features or their minimums and maximums). The scaling defined by the training set is then applied to the validation set and the test set as well to new data.

2.2 The *k*-Means Algorithm

To cluster observations we need a distance measure. Suppose first that there are only two features, *x* and *y*, so that we can plot the observations on a two-dimensional chart. Consider the two observations, A and B, in Figure 2.1. A natural distance measure is the Euclidean distance. This is the length of the straight line AB. Suppose that for observation A, $x = x_A$ and $y = y_A$, while for observation B, $x = x_B$ and $y = y_B$. The Euclidean distance between A and B (using Pythagoras' theorem) is

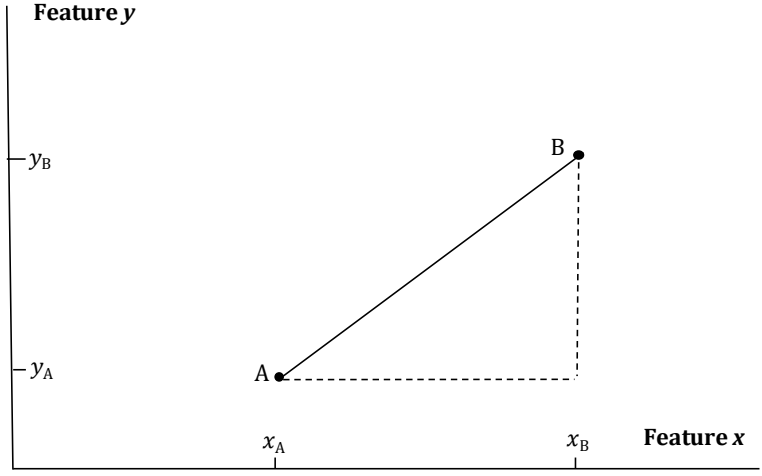
$$\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

This distance measure can be extended to many dimensions. Suppose we have observations on *m* features and that the value of the *j*th feature for the *i*th observation is v_{ij}. The distance between the *p*th observation and the *q*th observation is

$$\sqrt{\sum_{j=1}^{m}\left(v_{pj} - v_{qj}\right)^2}$$

The extension from two features to three features is fairly easy to understand. It involves measuring the distance in three dimensions rather than two. Imagining distances when $m > 3$ is not so easy, but the formula is a natural extension of that for one, two, and three dimensions.

Figure 2.1 The Euclidean distance between observations A and B, with co-ordinates (x_A, y_A) and (x_B, y_B), is the length of the line AB.



Another concept we need in order to understand the *k*-means algorithm is the center of a cluster (sometimes referred to as the cluster's *centroid*). Suppose that a certain set of observations is regarded as a cluster. The center is calculated by averaging the values of each of the features for the observations in the cluster. Suppose there are four features and the five observations in Table 2.1 are considered to be a cluster. The center of the cluster is a point that has values of 0.914, 0.990, 0.316, and 0.330 for features 1, 2, 3, and 4, respectively. (For example, 0.914 is the average of 1.00, 0.80, 0.82, 1.10, and 0.85.) The distance between each observation and the center of the cluster (shown in the final column of Table 2.1) is calculated in the same way as the distance between A and B in Figure 2.1. For example, the distance of the first observation from the center of the cluster is

$$\sqrt{(1.00 - 0.914)^2 + (1.00 - 0.990)^2 + (0.40 - 0.316)^2 + (0.25 - 0.330)^2}$$

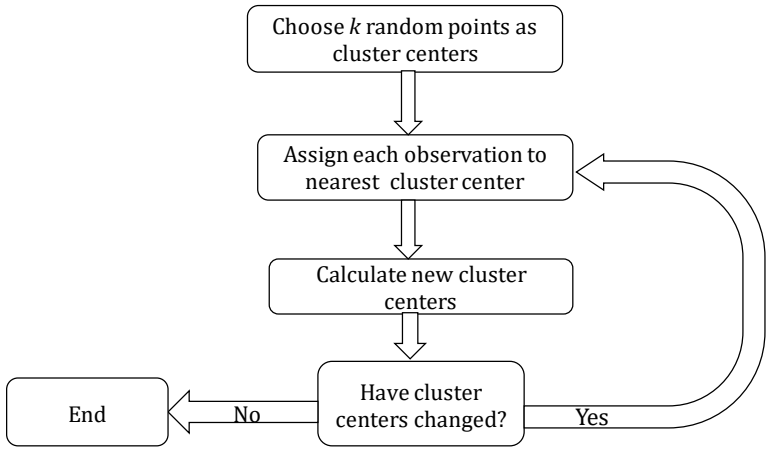
which equals 0.145.

Table 2.1 Calculation of the center of a cluster of five observations on four features.

Observ-ation	*Feature 1*	*Feature 2*	*Feature 3*	*Feature 4*	*Distance to center*
1	1.00	1.00	0.40	0.25	0.145
2	0.80	1.20	0.25	0.40	0.258
3	0.82	1.05	0.35	0.50	0.206
4	1.10	0.80	0.21	0.23	0.303
5	0.85	0.90	0.37	0.27	0.137
Center	0.914	0.990	0.316	0.330	

Figure 2.2 illustrates how the *k*-means algorithm works. The first step is to choose *k*, the number of clusters (more on this later). We then randomly choose *k* points for the centers of the clusters. The distance of each observation from each cluster center is calculated as indicated above and observations are assigned to the nearest cluster center. This produces a first division of the observations into *k* clusters. We then compute new centers for each of the clusters, as indicated in Figure 2.2. The distances of each observation from the new cluster centers is then computed and the observations are re-assigned to the nearest cluster center. We then compute new centers for each of the clusters and continue in this fashion until the clusters do not change.

Figure 2.2 The *k*-means algorithm



A measure of the performance of the algorithm is the within-cluster sum of squares, also known as the *inertia*. Define d_i as the distance of the ith observation from the center of the cluster to which it belongs. Then:

$$\text{Inertia} = \text{Within-cluster sum of squares} = \sum_{i=1}^{n} d_i^2$$

where n is the number of observations. For any given value of k, the objective of the k-means algorithm should be to minimize the inertia. The results from one run of the algorithm may depend on the initial cluster centers that are chosen. It is therefore necessary to re-run the algorithm many times with different initial cluster centers. The best result across all runs is the one for which the inertia is least.

Generally, the inertia decreases as k increases. In the limit when k equals the number of observations there is one cluster for each observation and the inertia is zero.

2.3 Choosing *k*

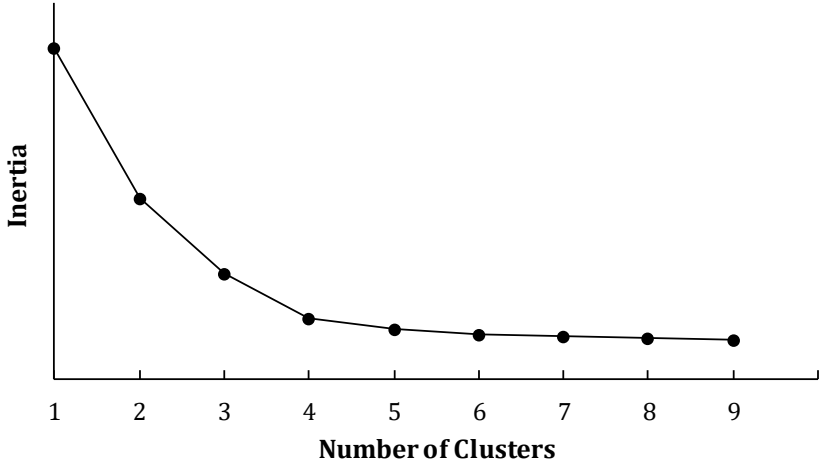
In some cases, the choice of k may depend on the objective of the clustering. For example, a company that is planning to produce small, medium, large, and extra-large sweaters for men might collect data on various relevant features (arm length, shoulder width, chest measurement, etc.) for a random sample of men and then create four clusters to help with product design. In other situations, the user of the algorithm may not have any preconceived ideas about k and just want to optimally group each observation with other similar observations.

The *elbow method* is a popular approach for determining the number of clusters. The k-means algorithm is carried out for a range of values of k (e.g., all values between 1 and 10). The inertia is then plotted against the number of clusters as indicated in Figure 2.3. The slope of the line in this chart indicates how the within-cluster sum of squares declines as the number of clusters increases. In this example, the decline is quite large when we move from one to two, two to three, and three to four clusters. After four clusters, the decline is much smaller. We conclude that the optimal number of clusters is four.

In addition to the within-cluster sum of squares, we are likely to be interested in how distinct the clusters are. If two clusters are very close together we might reasonably conclude that not much is gained by keeping them separate. Analysts therefore often monitor the distance

between cluster centers. If changing the number of clusters from k to $k + 1$ leads to two clusters with centers that are very close to each other, it might be considered best not to make the change.

Figure 2.3 Application of the elbow method. The inertia (within-cluster sum of squares) is plotted against the number of clusters



A less subjective way of choosing the number of clusters is the *silhouette method*. Again, we carry out the k-means algorithm for a range of values of k. For each value of k, we calculate for each observation, i, the average distance between the observation and the other observations in the cluster to which it belongs. Define this as $a(i)$. We also calculate, for each of the other clusters, the average distance between the observation and the observations in that cluster. We define $b(i)$ as the minimum value of these average distances across all the other clusters. We expect $b(i)$ to be greater than $a(i)$ as otherwise it probably would have made sense to allocate observation i to a different cluster. The silhouette of an observation measures the extent to which $b(i)$ is greater than $a(i)$. It is[1]

[1] See L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis,* Wiley 1990.

$$s(i) = \frac{b(i) - a(i)}{\max[a(i), b(i)]}$$

The silhouette, $s(i)$, lies between −1 and +1. (As already indicated, for observations that have been allocated correctly it is likely to be positive.) As it becomes closer to +1, the observation more clearly belongs to the group to which it has been assigned. The average of $s(i)$ over all observations in a cluster is a measure of the tightness of the grouping of those observations. The average of $s(i)$ over all observations in all clusters is an overall measure of the appropriateness of the clustering and is referred to as the *average silhouette score*. If for a particular data set the average silhouette scores are 0.70, 0.53, 0.65, 0.52, and 0.45 for k = 2, 3, 4, 5, and 6, respectively, we would conclude that k = 2 and 4 are better choices for the number of clusters than k = 3, 5, and 6.

Yet another approach for choosing k, known as the *gap statistic*, was suggested by Tibshirani et al (2001).[2] In this, the within-cluster sum of squares is compared with the value we would expect under the null hypothesis that the observations are created randomly. We create N sets of random points and, for each value of k that is considered, we cluster each set, calculating the within-cluster sum of squares. (N=500 usually works well.) Define

m_k: the mean of the within-cluster sum of squares for randomly created data when there are k clusters

s_k: the standard deviation of the within-cluster sum of squares for randomly created data when there are k clusters

w_k: the within-cluster sum of squares for the data we are considering when there are k clusters

We set

$$\text{Gap}(k) = m_k - w_k$$

This is the difference between the within-cluster sum of squares statistic for the random data and the data of interest. It is argued that the best choice for k is the smallest value such that Gap(k) is within s_{k+1} of Gap(k+1).

[2] See R. Tibshirani, G. Walther, and T. Hastie (2001), "Estimating the number of clusters in a data set via the gap statistic," *Journal of the Royal Statistical Society*, B, 63, Part 2: 411-423.

2.4 The Curse of Dimensionality

As the number of features increases, the *k*-means algorithm becomes affected by what is known as the "curse of dimensionality." Distances between observations increase. Consider the Euclidean distance between a point where all features equal 1.0 and a point where all features equal 0.0. When there is one feature the distance is 1.0; when there are two features the distance is $\sqrt{2}$ or 1.4; when there are three features, it is $\sqrt{3}$ or 1.7; when then are 100 features it is 10; and when there are 1,000 features it is 31.6. One consequence of this is that we cannot compare a within-cluster sum of squares given by data with a small number of features to one given by data with a large number of features.

Another problem is that, as the number of features increases, the distance measure that we have defined does not always differentiate well between observations that are close and those that are far apart. As a result the *k*-means algorithm works less well. This has led some users of the algorithm to search for alternatives to the Euclidean distance measure.

The Euclidean distance between an observation where feature j is x_j and another observation where feature j is y_j can be written

$$\sqrt{\sum_{j=1}^{m}(x_j - y_j)^2}$$

One alternative is

$$1 - \frac{\sum_{j=1}^{m} x_j y_j}{\sqrt{\sum_{j=1}^{m} x_j^2 \sum_{j=1}^{m} y_j^2}}$$

This always lies between 0 and 2.

2.5 Country Risk

Consider the problem of understanding the risk of countries for foreign investment. Among the features that can be used for this are:

1. The real GDP growth rate (using data from the International Monetary Fund)
2. A corruption index (produced by Transparency International)
3. A peace index (produced by Institute for Economics and Peace)
4. A legal risk index (produced by Property Rights Association)

Values for each of these features for 122 countries and all analyses carried out are at www-2.rotman.utoronto.ca/~hull. Table 2.2 provides an extract from the data. The table shows the importance of feature scaling (see Section 2.1). The real GDP growth rate (%) is typically a positive or negative number with a magnitude less than 10. The corruption index is on a scale from 0 (highly corrupt) to 100 (no corruption). The peace index is on a scale from 1 (very peaceful) to 5 (not at all peaceful). The legal risk index runs from 0 to 10 (with high values being favorable). Table 2.3 shows the data in Table 2.2 after it has been scaled using *Z*-score normalization. It shows that Australia's real GDP growth rate is slightly above average and its corruption index is 1.71 standard deviations above the average. Its peace index is 1.20 standard deviations below average (but low peace indices are good) and the legal risk index is 1.78 standard deviations above the average.

Table 2.2 First few observations for clustering countries according to their risk for international investment (see csv file)

Country	*Real GDP growth rate (% per yr)*	*Corruption index*	*Peace index*	*Legal risk index*
Albania	3.403	39	1.867	3.822
Algeria	4.202	34	2.213	4.160
Argentina	−2.298	36	1.957	4.568
Armenia	0.208	33	2.218	4.126
Australia	2.471	79	1.465	8.244
Austria	1.482	75	1.278	8.012
Azerbaijan	−3.772	30	2.450	3.946

Once the data has been scaled, a natural next step, given that there are only four features, is to examine the features in pairs with a series of scatter plots. This reveals that the corruption index and legal risk index are highly correlated as shown in Figure 2.4. (This is perhaps not surprising. Corruption is likely to be more prevalent in countries where the legal systems are poor.) We therefore eliminate the corruption

index as the information it provides is largely captured by the legal risk index. This means that we can consider our data as being points in three-dimensional space, the dimensions being: real GDP growth rate, peace index, and legal risk index

Table 2.3 Data in Table 2.2 after using *Z*-score scaling (see Excel file)

Country	*Real GDP growth rate (% per yr)*	*Corruption index*	*Peace index*	*Legal risk index*
Albania	0.32	−0.38	−0.31	−1.20
Algeria	0.56	−0.64	0.47	−0.97
Argentina	−1.44	−0.54	−0.10	−0.69
Armenia	−0.67	−0.69	0.48	−0.99
Australia	0.03	1.71	−1.20	1.78
Austria	−0.27	1.50	−1.62	1.62
Azerbaijan	−1.90	−0.85	1.00	−1.11

Figure 2.4 Scatter plot of scaled legal risk index and corruption index (see Excel file)

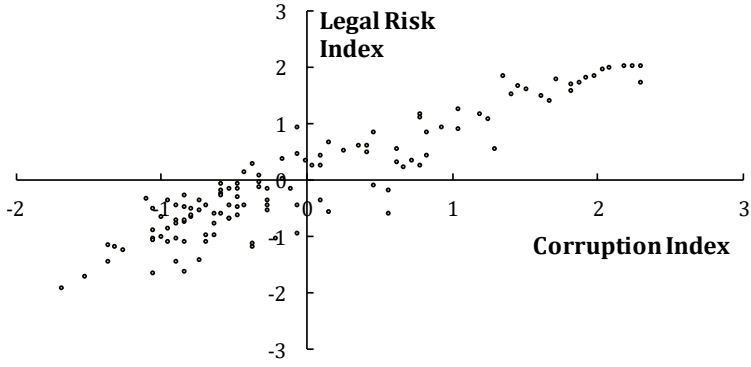
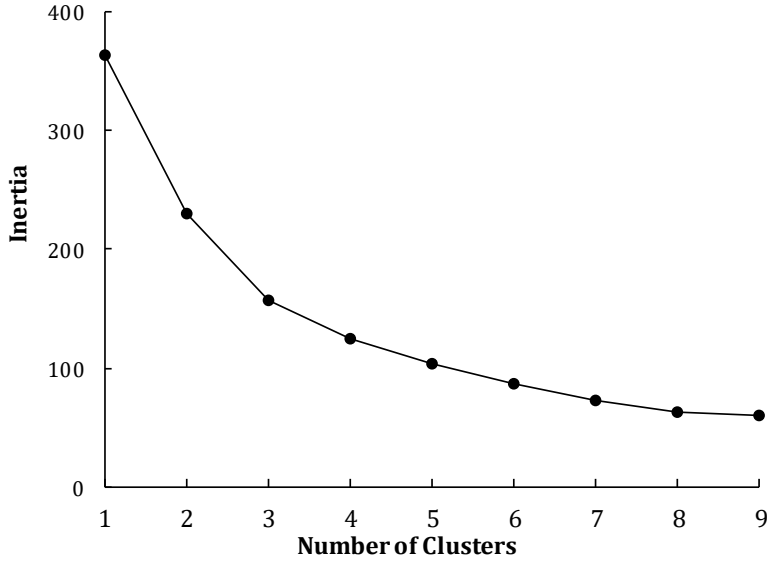


Figure 2.5 shows the results of applying the *k*-means algorithm to the country risk data when three features (real GDP growth rate, peace index, and legal risk index) are used. As expected, the total within-cluster sum of squares declines as the number of clusters, *k*, is increased. As explained earlier we can use the figure to look for an elbow,

a point where the benefit from increasing the number of clusters starts to be relatively small. The elbow is not as pronounced in Figure 2.5 as it is in Figure 2.3. However, a case can be made for three clusters as the decrease in the inertia as we move from one to two and two to three clusters is quite a bit greater than when we move from three to four clusters.

Figure 2.5 Variation of inertia (within-cluster sum of squares) with number of clusters for country risk example (from Python output)



The results from the silhouette method are given in Table 2.4. It can be seen that the average silhouette score is greatest when the number of clusters is three. For this particular data set, both the elbow method and the silhouette method point to the use of three clusters.[3]

Table 2.5 shows the cluster centers after scaling. It shows that high-risk countries are on average over one standard deviation worse than the mean for all three features. (Remember, high values are bad for the peace index.) Tables 2.6, 2.7, and 2.8 give the allocation of countries to three clusters.

[3] The elbow method and the silhouette method do not always agree.

Table 2.4 Variation of the average silhouette score with the number of clusters (from Python output)

Number of clusters	*Average silhouette Score*
2	0.363
3	0.388
4	0.370
5	0.309
6	0.303
7	0.315
8	0.321
9	0.292
10	0.305

Table 2.5 Cluster centers after features have been scaled so that mean is zero and standard deviation is one (from Python output)

	Peace index	*Legal index*	*Real GDP growth rate*
High risk	1.39	−1.04	−1.79
Moderate risk	0.27	−0.45	0.36
Low risk	−0.97	1.17	0.00

Table 2.6 High-risk countries (from Python output)

Argentina	Lebanon
Azerbaijan	Nigeria
Brazil	Russia
Burundi	Trinidad and Tobago
Chad	Ukraine
Democratic Republic of Congo	Venezuela
Ecuador	Yemen

Table 2.7 Moderate-risk countries (from Python output)

Albania	Madagascar
Algeria	Malawi
Armenia	Mali
Bahrain	Mauritania
Bangladesh	Mexico
Benin	Moldova
Bolivia	Montenegro
Bosnia and Herzegovina	Morocco
Bulgaria	Mozambique
Cameroon	Nepal
China	Nicaragua
Colombia	Oman
Croatia	Pakistan
Cyprus	Panama
Dominican Republic	Paraguay
Egypt	Peru
El Salvador	Philippines
Ethiopia	Romania
Gabon	Rwanda
Georgia	Saudi Arabia
Ghana	Senegal
Greece	Serbia
Guatemala	Sierra Leone
Honduras	South Africa
India	Sri Lanka
Indonesia	Tanzania
Iran	Thailand
Israel	The FYR of Macedonia
Jamaica	Tunisia
Jordan	Turkey
Kazakhstan	Uganda
Kenya	Vietnam
Kuwait	Zambia
Latvia	Zimbabwe
Liberia	

Table 2.8 Low-risk countries (from Python output)

Australia	Malaysia
Austria	Mauritius
Belgium	Netherlands
Botswana	New Zealand
Canada	Norway
Chile	Poland
Costa Rica	Portugal
Czech Republic	Qatar
Denmark	Singapore
Estonia	Slovakia
Finland	Slovenia
France	Spain
Germany	Sweden
Hungary	Switzerland
Iceland	Taiwan
Ireland	United Arab Emirates
Italy	United Kingdom
Japan	United States
Korea (South)	Uruguay
Lithuania	

2.6 Alternative Clustering Approaches

The *k*-means algorithm is the most popular approach to clustering, but there are alternatives. One is *agglomerative hierarchical clustering.* This involves the following steps:

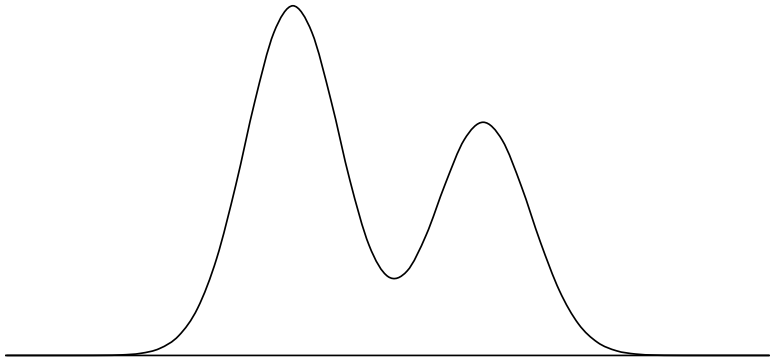
1. Start with each observation in its own cluster
2. Combine the two closest clusters
3. Repeat step 2 until all observations are in single cluster

The advantage of this approach is that the clusters form a hierarchy so that we can see clusters within clusters. The hierarchy can be used to obtain exactly *k* clusters for any given value of *k* between one and the number of observations. Its disadvantage is that it tends to be computationally very time consuming when there are a large number of observations.

A number of different measures of closeness between two clusters, A and B, have been proposed for use in step 2. One is the average Euclidean distance between an observation in cluster A and an observation in cluster B. Alternatively, we can use the minimum of these distances or the maximum of them. Another measure (a version of what is known as Ward's method) equals the increase in inertia when two clusters are combined. Whatever the measure chosen, step 2 involves searching for the two clusters with the smallest measure and then combining them.

Sometimes clusters are estimated from assumed statistical distributions. This is known as *distribution-based clustering*. Suppose for simplicity that there is only one feature and that the observations exhibit the probability distribution shown in Figure 2.6. We might reasonably assume that the observations come from a mixture of two normal distributions. This is a distribution when there is a probability p that an observation comes from a normal distribution with a particular mean and standard deviation and a probability $1-p$ that it comes from another normal distribution with a different mean and standard deviation. Statistical tools can be used to distinguish between the two distributions and therefore identify two clusters. A similar exercise can be carried out when there are several features and more than two distributions.

Figure 2.6 Probability distribution for data on a feature from which two normally distributed clusters could be separated



Density-based clustering involves forming clusters according to the closeness of individual observations. We might initially form a cluster of eight observations that are close to each other, then add another observation to the cluster that is close to at least five of these observations, then add another observation that is close to at least five of the observations in the new cluster, and so on. This can lead to clusters that have quite different shapes from the ones considered by the *k*-means algorithm. Figure 2.7 gives two examples. The *k*-means algorithm would not find these clusters because of the way it uses cluster centers. The two clusters in Figure 2.7a have the same centers and would therefore not be distinguished by *k*-means. In Figure 2.7b, *k*-means might identify several clusters but not the ones that are visually obvious.

Figure 2.7 Clusters which might be identified by a density-based clustering algorithm

(a) (b)

2.7 Principal Components Analysis

As an alternative to clustering, we can use principal components analysis (PCA) to understand the structure of data.[4] This takes data on *m* features and replaces it with a new set of *m* variables, referred to as factors or principal components, so that:

[4] PCA was proposed by Karl Pearson as early as 1901: K. Pearson (1901), "On lines and planes of closest fit to systems on points in space," *Philosophical Magazine*, 2(11): 559–572.

- Any observation on features is a linear combination of the factors
- The *m* factors are uncorrelated

PCA works best on normalized data. The first factor accounts for as much of the variability in the data as possible. Each succeeding factor then accounts for as much of the remaining variability in the data subject to the condition that it is uncorrelated to preceding factors. The quantity of a particular factor in a particular observation is the *factor score*.

A principal components analysis is often carried out for interest rate movements. (This could be relevant to a data scientist who is interested in studying the effect of interest rates on consumer behavior.) Table 2.9 shows the first three factors that are obtained when principal components analysis is applied to daily movements in interest rates with maturities of 1, 2, 3, 4, 5, 7, 10, and 30 years over a 12-year time period.[5] The numbers in each column are referred to as *factor loadings* and have the property that their sum of squares is 1. All rates move in the same direction for the first factor (principal component one, PC1). If we have +10 basis point units of that factor, the one-year rate increases by 2.16 basis points (or 0.0216%), the two-year rate increases by 3.31 basis points, and so on. If we have −20 basis point units of the first factor the one-year rate decreases by 4.32 basis points, the two-year rate decreases by 6.62 basis points, and so on.

PC2 is different from PC1 in that the first four rates move in one direction while the next four rates move in the opposite direction. This provides a "twist" to the term structure of interest rates where the slope changes. In PC3 short and long rates move in one direction while intermediate rates move in the other direction. This is referred to as a "bowing" of the term structure.

The importance of factors is measured by the standard deviations of their factors score across the observations. These are shown in Table 2.10 for the first three factors in our interest rate example. The variance accounted for by all eight factor scores is 388.8 in this example. The fraction of the variance accounted for by the first (most important) factor is therefore

[5] See J. Hull, *Options, Futures, and Other Derivatives*, 10th edition, Pearson, page 513 for this example. The worksheet for this is under the principal components analysis tab at

www-2.rotman.utoronto.ca/~hull/ofod

$$\frac{17.55^2}{338.8}$$

or about 90%. The fraction accounted for by the first two factors is

$$\frac{17.55^2 + 4.77^2}{338.8}$$

or about 97%. This shows that replacing the eight features defining term structure movements by two new variables (PC1 and PC2) captures most of the variation in the data. This illustrates what we are trying to achieve with PCA. We are trying to find a small number of variables that capture the structure of the data.

Table 2.9 Factor loadings defining the principal components for interest rate movements

Maturity	*PC1*	*PC2*	*PC3*
1yr	0.216	−0.501	0.627
2yr	0.331	−0.429	0.129
3yr	0.372	−0.267	−0.157
4yr	0.392	−0.110	−0.256
5yr	0.404	0.019	−0.355
7yr	0.394	0.194	−0.195
10yr	0.376	0.371	0.068
30yr	0.305	0.554	0.575

Table 2.10 Standard deviation of factor scores for interest rates (in basis points)

PC1	*PC2*	*PC3*
17.55	4.77	2.08

For a further example of PCA we return to the country risk data that we considered in Section 2.5. When all four features are used, the factors and the factor scores are shown in Tables 2.11 and 2.12. These reveal some interesting properties of the data. The first factor which accounts for 64% of the variance places roughly equal weight on corruption, peace, and legal risk. (Remember that a low score for peace is

good.)

The second factor accounts for a further 24% of the variance and places most of the weight on the real GDP growth rate. This recognizes that the real GDP growth rate is giving quite different information from the other three features.

In interpreting Table 2.11, note that we can change the signs of all the factor loadings in a column without changing the model. This is because the number of units of a factor that are present in an observation can be positive or negative. We should not for example read anything into the negative factor loading for the real GDP growth rate in PC2. We could change the signs of all the factor loadings for PC2 without changing the model.

Table 2.11 Factor loadings showing principal components for the country risk data (see Excel PCA file)

	PC1	*PC2*	*PC3*	*PC4*
Corruption index	0.594	0.154	−0.292	−0.733
Peace index	-0.530	0.041	−0.842	−0.086
Legal risk index	0.585	0.136	−0.431	0.674
GDP growth rate	0.152	−0.978	−0.141	−0.026

Table 2.12 Standard deviation of factor scores for country risk data (see Excel PCA file)

PC1	*PC2*	*PC3*	*PC4*
1.600	0.988	0.625	0.270

The third factor which accounts for about 10% of the variance places most weight on the peace index and suggests that this has extra information over that in corruption and legal risk. The fourth factor is relatively unimportant, accounting for less than 2% of the variance. The PCA confirms the result in Figure 2.4 that the corruption index and legal risk index provide similar information.

PCA is sometimes used in supervised learning as well as unsupervised learning. We can use it to replace a long list of features by a much smaller list of manufactured features derived from a PCA. The manufactured features are chosen so that they account for most of the variability in the data we are using for prediction and have the nice property that they are uncorrelated.

Finally, we emphasize that, when clustering or using PCA, we are not trying to predict anything. We are merely investigating the structure of the data. In our country risk example, an analyst might assume that the features are related to investment risk, but there is no guarantee that this is so. (For example, we have not tried to relate the features to losses incurred on investments in different countries, as we might do in supervised learning.)

Summary

Unsupervised learning is concerned with understanding patterns within data. Typically, it involves looking for clusters, i.e., groups of similar observations. Companies often use unsupervised learning to understand the different types of customers they have so that they can communicate with them more effectively.

Feature scaling is usually necessary for clustering. Without feature scaling, the impact of a feature on clustering is likely to depend on the scale used to measure it. There are two main approaches to feature scaling. One is *Z*-score normalization where features are scaled so that they have a mean of zero and a standard deviation of one. The other is the min-max method where all features are scaled so that they have values between zero and one.

A clustering algorithm requires a distance measure. The most popular such measure is the Euclidean distance, which is the square root of the sum of squared differences between feature values. A cluster's center is the point obtained by averaging the feature values for all observations in the cluster. The most popular clustering algorithm is *k*-means. For a particular value of *k* (the number of clusters), it minimizes inertia, which is defined as the total sum of squared distances of the observations from their cluster centers.

Choosing the best value for the number of clusters, *k*, is not always easy. One approach is the "elbow method" which involves continuing to increase *k* until the improvement in the inertia is relatively small. Another approach is the silhouette method which compares for each observation (a) the average distance of the observation from other points in its own cluster and (b) the average distance of the observation from points in the closest other cluster. A third approach involves calculating the gap statistic which compares the clustered observations to observations that are created randomly.

As the number of features increases, the Euclidean distance measure increases. This is an aspect of the curse of dimensionality and makes it more difficult to use the *k*-means algorithm when there are a large number of features. As a result, it may be desirable to change the distance measure so that it stays within certain bounds as the number of features increases.

There are a number of alternatives to the *k*-means algorithm. One is hierarchical clustering. In this we start with the situation where each observation is in its own cluster. We then slowly reduce the number of clusters by combining clusters that are close to each other. Distribution-based clustering involves assuming a distribution for the data that is a mixture of normal (or other) distributions and estimating the parameters of those distributions. Density-based clustering involves looking for regions where data is dense without reference to cluster centers.

Principal components analysis (PCA) is an important tool in machine learning. It involves replacing a large number of features by a smaller number of manufactured features that capture most of the variability. The manufactured features are uncorrelated with each other.

SHORT CONCEPT QUESTIONS

2.1 Why is feature scaling important in unsupervised learning? Explain two methods for feature scaling. What are the advantages and disadvantages of each method?

2.2 Suppose there are three features, A, B, and C. One observation has values 2, 3, and 4 for A, B, and C, respectively. Another has values 6, 8, and 7 for A, B, and C respectively. What is the Euclidean distance between the two observations?

2.3 What would be the center of a cluster consisting of the two observations in question 2.2?

2.4 Explain the steps in the *k*-means algorithm.

2.5 Explain (a) the elbow method and (b) the silhouette method for choosing the number of clusters, *k*.

2.6 Why do the Euclidean distances between observations increase as the number of features increases? Suppose that you start with ten features and then by mistake create ten more features that are identical to the first ten. What effect does this have on the distance between two of the observations?

2.7 How does hierarchical clustering work? What are its advantages and disadvantages relative to *k*-means?

2.8 Explain what is meant by (a) distribution-based clustering and (b) density-based clustering.
2.9 Under what circumstances is principal components analysis most useful for understanding data?
2.10 What is meant by (a) a factor loading and (b) a factor score?

EXERCISES

2.11 Use data at

www-2.rotman.utoronto.ca/~hull

to calculate cluster centers without scaling for the 14 high risk countries in Table 2.6. Scale the cluster centers. Verify that your answer agrees with the result in Table 2.5 for high risk countries.

2.12 Use the principal components analysis results to determine how you would describe country risk with two factors. Give results for both the scaled data and the original non-scaled data.

2.13 Python code used to produce the results in this chapter can be found at

www-2.rotman.utoronto.ca/~hull

(a) Carry out k-means clustering for k=3 with all four features (corruption index, peace index, legal risk index, and real GDP growth rate). Compare the countries that are in the high-risk cluster with those that are in the high-risk cluster when only three features are used (see Table 2.6).

(b) Use hierarchical clustering to determine three clusters from the peace index, legal risk index, and real GDP growth rate. Compare the countries that are in the high-risk cluster with those that are in the high-risk cluster when the k-means algorithm is used (see Table 2.6). A Python package, AgglomerativeClustering, for hierarchical clustering can be imported from sklearn.cluster. Try different measures of closeness (referred to as "linkage" in the package).

2.14 The country risk data used in Section 2.5 is from the years 2016 and 2017. Data for the year 2019 is at

www-2.rotman.utoronto.ca/~hull

Use the 2019 data to determine clusters. How do the clusters differ from those in Tables 2.6 to 2.8? What other data would you like to collect to improve your clustering?

Chapter 3

Supervised Learning: Linear and Logistic Regression

Linear regression has been used by statisticians for many years. The famous mathematician Carl Friedrich Gauss is credited with first suggesting the least squares approach that underlies linear regression in about the year 1800. In machine learning, we do not have to assume linear relationships. (Indeed, many of the tools we will present later in this book lead to non-linear models.) In spite of this, linear regression remains an important tool in machine learning. It is often one of the first tools used by analysts in supervised learning.

Plain vanilla linear regression involves minimizing the mean squared error (mse) when the value of a target is being predicted from one or more features. This will be familiar to many readers. This chapter discusses how categorical features (i.e., features that are not numerical) can be incorporated into linear regression and therefore used for prediction. It also discusses Ridge, Lasso, and Elastic Net regression which are particularly useful when predictions are being made from a large number of features. It then moves on to explain logistic regression, which is a way of handling situations where the objective is to learn how to classify data. Finally, it covers the *k*-nearest-neighbor algorithm which is a simple alternative to linear and logistic regression.

3.1 Linear Regression: One Feature

We start with the simple situation where the target Y is being predicted from a single feature X. In linear regression we assume a linear relationship so that the model is:

$$Y = a + bX + \varepsilon$$

where a and b are constants and ε is the error term. Denote X_i and Y_i ($1 \le i \le n$) as the values of X and Y for the ith observation in the training set. The "best fit" values of a and b are defined as those that minimize the mean squared error (mse) for the observations in the training set. This means that we choose a and b so that:

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - a - bX_i)^2$$

is minimized.[1] We can use calculus to find the minimum. Denoting the averages of the observations on X and Y by $\bar{X}$ and $\bar{Y}$

$$b = \frac{\sum_{i=1}^{n} X_i Y_i - n\bar{X}\bar{Y}}{\sum_{i=1}^{n} X_i^2 - n\bar{X}^2}$$

$$a = \bar{Y} - b\bar{X}$$

An example of linear regression when there is only one feature is provided by the model in Figure 1.5 of Chapter 1. This is based on the training data set in Table 1.1, which is reproduced in Table 3.1. In this case $n = 10$, $\bar{X} = 43$, and $\bar{Y} = 216{,}500$. Also,

$$\sum_{i=1}^{10} X_i Y_i = 100{,}385{,}000$$

$$\sum_{i=1}^{10} X_i^2 = 20{,}454$$

so that

[1] This is the same as minimizing the sum of squared errors as n is a constant for any given data set.

$$b = \frac{100{,}385{,}000 - 10 \times 43 \times 216{,}500}{20{,}454 - 10 \times 43^2} = 3{,}827.3$$

$$a = 216{,}500 - 3827.3 \times 43 = 51{,}160.4$$

and the model is

$$Y = 51{,}160.4 + 3{,}827.3X$$

Sometimes the parameter a is set equal to zero. In this case,

$$b = \frac{\sum_{i=1}^{n} X_i Y_i}{\sum_{i=1}^{n} X_i^2}$$

(We use the linear model as an example here but it will be recalled that in Chapter 1 we did not find a linear model to be the best model for the data in Table 3.1.)

Table 3.1 The training set: salaries for a random sample of ten people working in a particular profession in a certain area

Age (years)	*Salary ($)*
25	135,000
55	260,000
27	105,000
35	220,000
60	240,000
65	265,000
45	270,000
40	300,000
50	265,000
30	105,000

3.2 Linear Regression: Multiple Features

When more than one feature is used to predict a target we can write the model as

$$Y = a + b_1 X_1 + b_2 X_2 + \cdots + b_m X_m + \varepsilon \tag{3.1}$$

where Y is the value of the target and the X_j ($1 \le j \le m$) are the values of the features that are used to predict Y. As before, the prediction error is denoted by ε. The parameters a and b_j ($1 \le j \le m$) are chosen to minimize the mean squared error over the training data set. This means that the task in linear regression is to find values for a and the b_j that minimize

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - a - b_1X_{i1} - b_2X_{i2} - \cdots - b_mX_{im})^2 \qquad (3.2)$$

where Y_i and X_{ij} are the values of the target and the jth feature for observation i. In machine learning, the parameter a is referred to as the *bias* and the coefficients b_j are referred to as the *weights*. As in the case of a single feature, calculus can be used to determine the conditions for a minimum. This leads to a set of simultaneous equations for determining the a and b_j. These equations can be solved using matrix algebra.

Statisticians list a number of conditions that must be satisfied for linear regression to be valid. The relationship between the target and the features should be linear; there should be no correlation between the feature values; the errors in the prediction should be normally distributed with a constant standard deviation; and the observations should be independent. In practice, these conditions are at best satisfied only approximately. Identifying serious violations can lead an analyst to find a better model (e.g., by switching from a linear to a quadratic model or by transforming feature values in some way). But it is worth noting that machine learning is different from traditional statistics in that we are usually working with very large data sets and can handle model suitability issues by dividing the data into a training set, validation set, and test set, as outlined in Chapter 1.

Gradient Descent Algorithm

An alternative to matrix algebra, which has to be used for some of the analyses presented later in this chapter, is the *gradient descent algorithm*. This is an iterative search routine for finding the minimum. Imagine plotting the expression in equation (3.2) as a function of a and the b_j ($1 \le j \le m$) in $m + 1$ dimensions. We can think of this function as a valley and our task as that of finding the bottom of the valley. Wherever we are in the valley, we can use calculus to determine the path of steepest descent down the valley (i.e., the direction in which each of the a and b_j should be changed to descend as quickly as possible). The gradient descent method proceeds as follows:

- choose initial values for a and the b_j
- calculate path of steepest descent.
- take a step down the path of steepest descent
- re-compute the path of steepest descent
- take another step
- and so on

We will discuss this methodology in greater detail in Chapter 6.

Polynomial Regressions

Chapter 1 uses the data in Table 3.1 to carry out polynomial regressions. These are regressions where there is a single feature X and X_j is set equal to X^j so that the model is:

$$Y = a + b_1X + b_2X^2 + \cdots + b_mX^m + \varepsilon$$

In Chapter 1 we found that the model where m = 5 fits the training set well but does not generalize well to the validation set. The quadratic model where m = 2 was chosen because it provided a better fit than the linear model while still generalizing well.

Sometimes products of features as well as powers of features are used in a regression. An example where the target is being predicted from two features is

$$Y = a + b_1X_1 + b_2X_1^2 + b_3X_2 + b_4X_2^2 + b_5X_1X_2 + \varepsilon$$

Regression Statistics

A number of statistics can be produced from a linear regression and can be useful if the assumptions mentioned earlier are approximately satisfied. The *R*-squared statistic is between zero and one and measures the proportion of the variance in the target that is explained by the features. It is

$$1 - \frac{\text{Variance of the errors } \varepsilon}{\text{Variance of the observations on the target } Y}$$

When there is only one feature, *R*-squared is the square of the coefficient of correlation. In the case of the data in Table 3.1, the *R*-squared for the linear model is 0.54 while that for the quadratic model is 0.80.

The *t*-statistic of the a or b_j parameters estimated by a linear regression is the value of the parameter divided by its standard error. The *P*-

value is the probability of obtaining a *t*-statistic as large as the one observed if we were in the situation where the parameter had no explanatory power at all. A *P*-value of 5% or less is generally considered as indicating that the parameter is significant. If the *P*-value for the parameter b_j in equation (3.1) is less than 5%, we are over 95% confident that the feature X_j has some effect on Y. The critical *t*-value for 95% confidence when the data set is large is 1.96 (i.e., *t*-statistics greater than 1.96 are significant in the sense that they give rise to *P*-values less than 5%).[2]

3.3 Categorical Features

The features used for prediction can be categorical as well as numerical. As explained in Chapter 1, a categorical variable is a variable that can fall into one of a number of categories. For example, the purchasers of a product might be categorized as male or female. The hair color of women buying a particular beauty product might be categorized as blonde, red, brown, or black.

The standard way of dealing with categorical features is to create a *dummy variable* for each category. The value of this variable is one if the feature is in the category and zero otherwise. This is referred to as *one-hot encoding*. In the situation where individuals are categorized as male or female, we could create two dummy variables. For men the first dummy variable would be one and the second would be zero. For women the first dummy variable would be zero and the second dummy variable would be one. In the hair color example, there would be four dummy variables and, in the case of each observation, we would assign one to the relevant variable and zero to the other three.

The procedure we have described is appropriate when there is no natural ordering between the feature values. When there is a natural ordering, we can reflect this in the numbers assigned. For example, if the size of an order is classified as small, medium, or large, we can replace the feature by a numerical variable where small = 1, medium = 2, and large = 3. Similarly, if job title is a feature and the categorization is analyst, associate, vice president, executive director, and managing di-

[2] This is for what is referred to as a "two-tailed test" where the analyst is testing for the significance of either a positive or negative relationship between the feature and the target. *P*-values are usually quoted for a two-tailed test. In a one-tailed test, we expect the relationship to have a particular sign (positive or negative) and disregard the possibility of it having the other sign. The critical *P*-value for a one-tailed test when the data set is large is 1.65.

rector, we might replace the feature by a numerical value where analyst = 1, associate = 2, vice president = 3, executive director = 4, and managing director = 5. But, after considering salaries and responsibilities, we might choose a different set of numerical values such as analyst = 1, associate = 2, vice president = 4, executive director = 7, and managing director = 10.

Once categorical features have been converted to numerical values, a linear regression can be carried out in the usual way. Some of the dummy variables created from categorical variables may have a significant effect on the target while others do not.

The Dummy Variable Trap

When one-hot encoding is used for one or more categorical variables and there is a constant (bias) term in the regression, there is no unique best-fit linear regression equation. This is referred to as the *dummy variable trap*.

Suppose that the following equation has been derived for predicting a target Y

$$Y = a + b_1X_1 + b_2X_2 + \cdots + b_mX_m + \varepsilon$$

and that the first few features, X_1, X_2,....,X_k ($k \le m$), are dummy variables created from the one-hot encoding of a particular categorical variable. Imagine what happens if we add a constant C to the bias, a, and subtract C from each of the weights $b_1, b_2, \ldots, b_k$. From the nature of one-hot encoding, it must be the case that $X_j = 0$ for $1 \le j \le k$ except for one particular value of j where $X_j = 1$. As a result, subtracting C from each of b_1, b_2,....,b_k reduces the estimated value of Y by exactly C. Adding C to a increases the estimated value of Y by C. The estimated value of Y is therefore the same when we make these two changes. This is true for any value of C.

Fortunately, regularization, which is designed to simplify models and avoid over-fitting (see next section), has the side effect of dealing with the dummy variable trap problem by finding a single "best" regression equation. This is an equation where the magnitudes of the weights are small.

3.4 Regularization

In machine learning there are often a large number of features, some of which may be correlated with each other. This can lead to over-fitting

and models that are unnecessarily complex. A common way of handling this is known as *regularization*. In the next three sections we introduce three regularization techniques and illustrate them with the data in Table 3.1. All calculations are in www-2.rotman.utoronto.ca/~hull.

Before using regularization, it is important to carry out feature scaling to ensure that the numerical values of features are comparable. This is described in Section 2.1.

3.5 Ridge Regression

Ridge regression (also referred to as Tikhonov regression) is a regularization technique where we change the function that is to be minimized from that in equation (3.2) to:[3]

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - a - b_1X_{i1} - b_2X_{i2} - \cdots - b_mX_{im})^2 + \lambda\sum_{j=1}^{m} b_j^2 \qquad (3.3)$$

For each feature j, Ridge regression involves adding the term λb_j^2 to the mean squared error. (Note that we do not add a term corresponding to the bias, a.) This change has the effect of encouraging the model to keep the weights b_j as small as possible. Ridge regression is referred to as *L2 regularization*.

Consider the situation where there are two highly correlated features, X_1 and X_2, that have been scaled so that they have mean zero and standard deviation one. Suppose that the best fit linear model, obtained by minimizing the objective function in equation (3.2), is

$$Y = a + 1000X_1 - 980X_2$$

Because the features are close substitutes, simpler models such as

$$Y = a + bX_1 \qquad \text{or} \qquad Y = a + bX_2$$

[3] Alternative equivalent objective functions for Ridge regression are sometimes used. The value of λ depends on the way the objective function is specified. Sklearn's LinearRegression package for Python adds the sum of squared errors, rather than the mean squared error, to $\lambda\sum_{j=1}^{m} b_j^2$. This means that Sklearn's λ should be n times the λ in equation (3.3). In Géron's book *Hands on machine learning with Scikit-Learn and TensorFlow*, $1/n$ in equation (3.3) is replaced by $1/(2n)$. The value of λ used in this formulation should be half that in equation (3.3).

where b is about 20 are likely to generalize better. Ridge regression, because it penalizes large positive or negative values of the coefficients, would find one of these models.

The Ridge regression model in equation (3.3) should only be used for determining model parameters using the training set. Once the model parameters have been determined, equation (3.2) (i.e., the equation without the $\lambda \sum_{j=1}^{m} b_j^2$ term) should be used for prediction. A validation set should be used to test whether equation (3.2) generalizes well. The accuracy of the model that is finally chosen should be quantified using equation (3.2) on the test set.

The parameter λ is referred to as a *hyperparameter* because it is used to train the model, but is not part of the model that is used to predict Y. The choice of a value for λ is obviously important. A large value for λ would lead to all the b_j being set equal to zero. (The resulting model would then be uninteresting as it would always predict a value equal to a for Y.) In practice, it is desirable to try several different values for λ and see how well the resulting models generalize to a validation set.

We mentioned that the standard linear regression objective function in equation (3.2) can be minimized analytically by using matrix algebra. We can also minimize the Ridge regression objective function in equation (3.3) analytically. The gradient descent method introduced in Section 3.2 is an alternative.

We will use the model where we fitted a polynomial of degree five to the training data set in Table 3.1 as an illustration of regularization. We know from Chapter 1 that the model over-fits the training set. It will be instructive to see how regularization handles this.

As a first step, feature scaling is necessary. In this example we have five features. These are X, X^2, X^3, X^4, and X^5, where X is age in years. Each of these must be scaled. (Note that it is not sufficient to scale only X.) We will use Z-score scaling. Table 3.2 shows values of the features together with means and standard deviations. (The size of the numbers in the table emphasizes the importance of scaling.) Table 3.3 shows the scaled features.

The best fit linear regression when the features have the scaled values in Table 3.3 and the salary Y is measured in \$'000s is

$$Y = 216.5 - 32{,}622.6X + 135{,}402.7X^2 - 215{,}493.1X^3 + 155{,}314.6X^4 - 42{,}558.8X^5 \quad (3.4)$$

Table 3.2 Feature values. X equals the age of individuals in the training set

Instance	X	X^2	X^3	X^4	X^5
1	25	625	15,625	390,625	9,765,625
2	55	3,025	166,375	9,150,625	503,284,375
3	27	729	19,683	531,441	14,348,907
4	35	1,225	42,875	1,500,625	52,521,875
5	60	3,600	216,000	12,960,000	777,600,000
6	65	4,225	274,625	17,850,625	1,160,290,625
7	45	2,025	91,125	4,100,625	184,528,125
8	40	1,600	64,000	2,560,000	102,400,000
9	50	2,500	125,000	6,250,000	312,500,000
10	30	900	27,000	810,000	24,300,000
Mean	43.2	2,045	104,231	5,610,457	314,153,953
S.D.	14.1	1,259	89,653	5,975,341	389,179,640

Table 3.3 Values of features in Table 3.2 after scaling

Instance	X	X^2	X^3	X^4	X^5
1	−1.290	−1.128	−0.988	−0.874	−0.782
2	0.836	0.778	0.693	0.592	0.486
3	−1.148	−1.046	−0.943	−0.850	−0.770
4	−0.581	−0.652	−0.684	−0.688	−0.672
5	1.191	1.235	1.247	1.230	1.191
6	1.545	1.731	1.901	2.048	2.174
7	0.128	−0.016	−0.146	−0.253	−0.333
8	−0.227	−0.354	−0.449	−0.511	−0.544
9	0.482	0.361	0.232	0.107	−0.004
10	−0.936	−0.910	−0.861	−0.803	−0.745

We can now apply Ridge regression. Table 3.4 shows the bias, a, and weights, b_j, for two different values of λ. Setting $\lambda = 0$ would give the "no-frills" regression result in equation (3.4). It can be seen that moving from $\lambda = 0$ to $\lambda = 0.02$ has a dramatic effect on the weights, reducing them by several orders of magnitude. Increasing λ from 0.02 to 1.0 reduces the weights further. Figures 3.1 to 3.3 plot the forecasted salary as a function of age for the predictions given by $\lambda = 0$, 0.02, and 0.1. It can be seen that, as λ increases, the model becomes less complex. The $\lambda = 0.02$ model is very similar to the quadratic model (see Figure 1.4), which we found in Chapter 1 generalizes well to new data.

Table 3.4 Variation of bias and weights for different values of λ for Ridge regression. Salary is measured in $'000s (see Excel file for salary vs. age example)

λ	a	b_1	b_2	b_3	b_4	b_5
0.02	216.5	97.8	36.6	−8.5	−35.0	−44.6
0.10	216.5	56.5	28.1	3.7	−15.1	−28.4

Figure 3.1 Prediction of salary ($'000s), no regularization (λ = 0)

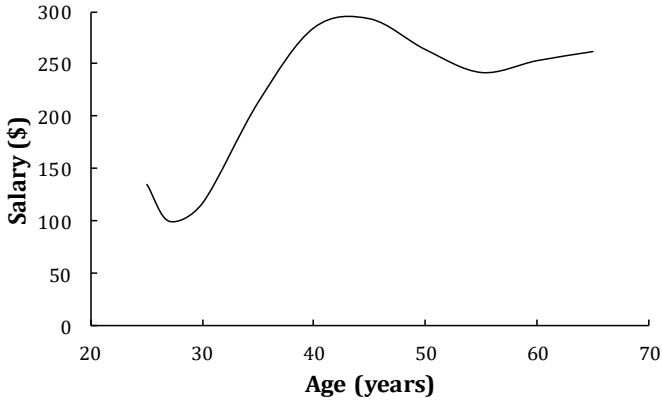


Figure 3.2 Prediction of salary ($'000s) when Ridge regression is used with λ = 0.02

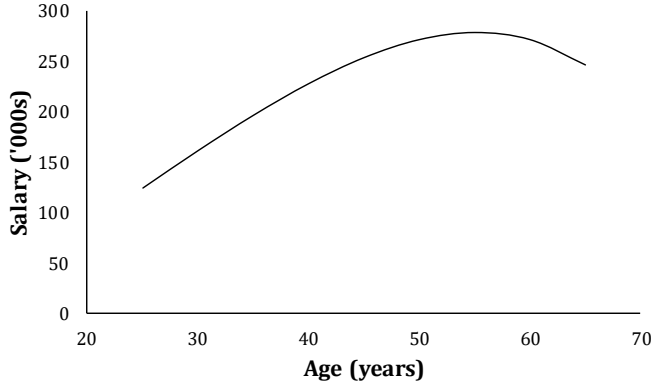
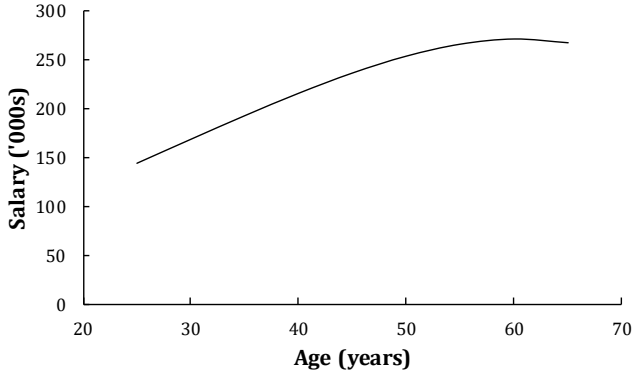


Figure 3.3 Prediction of salary when Ridge regression is used with λ = 0.1



3.6 Lasso Regression

Lasso is short for "Least absolute shrinkage and selection operator." In Ridge, we added a constant times the sum of the squared weights to the objective function. In Lasso, we add a constant times the sum of the absolute weights. This gives the following objective function: [4]

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - a - b_1X_{i1} - b_2X_{i2} - \cdots - b_mX_{im})^2 + \lambda\sum_{j=1}^{m}|b_j| \quad (3.5)$$

This function cannot be minimized analytically and so an approach similar to the gradient descent algorithm explained earlier must be used. Lasso regression is referred to as *L1 regularization*.

We saw in the previous section that Ridge regression reduces the weights assigned to features in order to simplify the model. The simplified model often generalizes better than the unregularized model. Lasso regression also has the effect of simplifying the model. It does this by setting the weights of unimportant features to zero. When there are a large number of features, Lasso can identify a relatively small subset of them to form a good prediction model.

[4] There are variations in the way the objective function is specified. Sklearn's LinearRegression package for Python replaces $1/n$ by $1/(2n)$ in equation (3.5). This means that Sklearn's λ should be half as much as the λ in equation (3.5).

The results of using Lasso for our example where a fifth order polynomial is used to predict the salary of an individual from the individual's age is shown in Table 3.5. The table shows that Lasso does indeed set the weights of some features equal to zero. We might expect that b_5, b_4, and possibly b_3 will be zero so that the model reduces to a quadratic or cubic model. In fact, this is not what happens. When $\lambda = 0.02$, Lasso only reduces b_3 to zero; when $\lambda = 0.1$, Lasso reduces b_2 and b_4 to zero; when $\lambda = 1$, Lasso reduces b_2, b_3, and b_5 to zero.

Figures 3.4, 3.5 and 3.6 show the predictive models that are created for $\lambda = 0.02$, 0.1, and 1. They are simpler than the fifth-degree polynomial model in equation (3.4) with much lower weights. As in the case of Ridge regression, the models become simpler as λ is increased. The model in Figure 3.6 is very similar to the quadratic model in Figure 1.4.

Table 3.5 Variation of bias and weights for different values of λ for Lasso regression. Salary is measured in $'000s (see Excel file for salary vs. age example)

λ	a	b_1	b_2	b_3	b_4	b_5
0.02	216.5	−646.4	2,046.6	0	−3,351.0	2,007.9
0.10	216.5	355.4	0	−494.8	0	196.5
1.00	216.5	147.4	0	0	−99.3	0

Figure 3.4 Prediction of salary when Lasso regression is used with $\lambda = 0.02$. Salary is measured in $'000s.

Figure 3.5 Prediction of salary when Lasso regression is used with λ = 0.1. Salary is measured in $'000s.

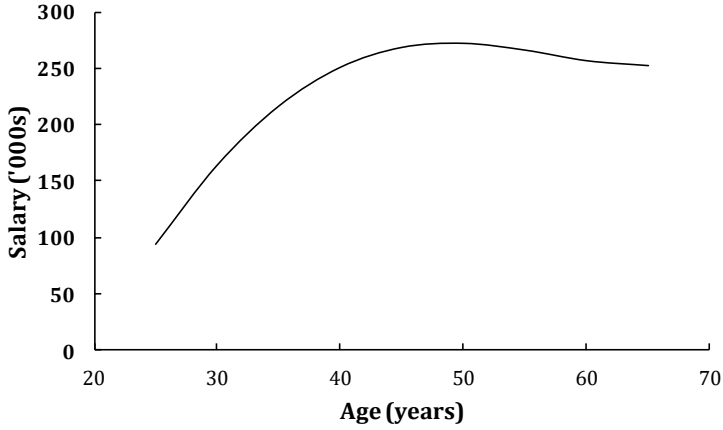
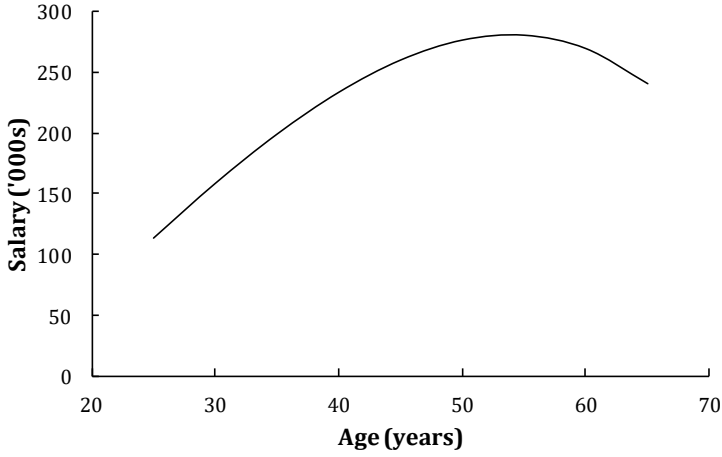


Figure 3.6 Prediction of salary when Lasso regression is used with λ = 1. Salary is measured in $'000s.



3.7 Elastic Net Regression

Elastic Net regression is a mixture of Ridge and Lasso. The function to be minimized includes both a constant times the sum of the squared weights and a different constant time the sum of the absolute values of

the weights. It is

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - a - b_1X_{1i} - b_2X_{2i} - \cdots - b_mX_{mi})^2 + \lambda_1\sum_{j=1}^{m}b_j^2 + \lambda_2\sum_{j=1}^{m}|b_j|$$

In Lasso some weights are reduced to zero, but others may be quite large. In Ridge, weights are small in magnitude, but they are not reduced to zero. The idea underlying Elastic Net is that we may be able to get the best of both worlds by making some weights zero while reducing the magnitude of the others. We illustrate this for our fifth order polynomial example by setting $\lambda_1 = 0.02$ and $\lambda_2 = 1$. The resulting model for predicting salary ($'000s), Y, from age (years), X, is

$$Y = 216.5 + 96.7X + 21.1X^2 - 26.0X^4 - 45.5X^5$$

This has a similar structure to the Lasso model when $\lambda = 0.02$ (see Table 3.5), except that the non-zero weights are much smaller. The model is shown in Figure 3.7. Again, we find that the result is very similar to the quadratic model which was developed in Chapter 1 for this example.

Figure 3.7 Prediction of salary when Elastic Net regression is used with $\lambda_1 = 0.2$ and $\lambda_2 = 1.0$ (see Excel file for salary vs. age example)

3.8 Results for House Price Data

So far, we have explained regularization with a baby data set of only ten observations. We now show how it can be used in a more realistic situation.

Local authorities in many countries need to predict the market prices of houses to determine property taxes. They can do this by relating the known prices of houses that have been sold to features such as the number of bedrooms, number of bathrooms, and the neighborhood in which the house is located. The data set we will use consists of information on houses that were sold in Iowa during a four-year period.[5]

Before starting we emphasize the importance of the material in Chapter 1 concerning the need to divide all available data into three parts: a training set, a validation set, and a test set. The training set is used to determine parameters for trial models. The validation set is used to determine the extent to which the models created from the training set generalize to new data. The test set is used as a final estimate of the accuracy of the chosen model. After data cleaning, we had 2,908 observations. We split this as follows: 1,800 in the training set, 600 in the validation set and 508 in the test set.

The full data set contains a total of about 80 features, some numerical and some categorical. To illustrate the regression techniques discussed in this chapter, we will use a total of 23 features. These are listed in Table 3.6. Twenty-one are numerical and two are categorical. One of the categorical features is concerned with the basement quality, as indicated by the ceiling height. The categories are:

- Excellent (>100 inches)
- Good (90 to 99 inches)
- Typical (80 to 89 inches)
- Fair (70 to 79 inches)
- Poor (< 70 inches)
- No basement

This is an example of a categorical variable where there is a natural ordering. We created a new variable that had values of 5, 4, 3, 2, 1, and 0 for the six categories, respectively.

The other categorical feature specifies the location of the house as in one of 25 neighborhoods. Given the real estate agents' mantra "location,

[5] This data set was used in a Kaggle competition where contestants tried to predict prices for test data.

location, location" we felt it important to include this feature. We therefore introduced 25 dummy variables. The dummy variable equals one for an observation if the neighborhood is that in which the house is located and zero otherwise. The total number of features in the model was therefore 47 (21 numerical features, 1 for basement quality and 25 for location).

Table 3.6 Features for estimating house prices and weights using a linear regression model on scaled data with no regularization (from Python)

Feature	*Weights for simple linear regression*
Lot area (square feet)	0.08
Overall quality (scale: 1 to 10)	0.21
Overall condition (scale: 1 to 10)	0.10
Year built	0.16
Year remodeled (= year built if no remodeling or additions)	0.03
Finished basement (square feet)	0.09
Unfinished basement (square feet)	−0.03
Total basement (square feet)	0.14
First floor (square feet)	0.15
Second floor (square feet)	0.13
Living area (square feet)	0.16
Number of full bathrooms	−0.02
Number of half bathrooms	0.02
Number of bedrooms	−0.08
Total rooms above grade	0.08
Number of fireplaces	0.03
Parking spaces in garage	0.04
Garage area (square feet)	0.05
Wood deck (square feet)	0.02
Open porch (square feet)	0.03
Enclosed porch (square feet)	0.01
Neighborhood (25 features)	−0.05 to 0.12
Basement quality	0.01

There are relationships between the features in Table 3.6. For example, the total basement area is the sum of the finished and unfinished areas. Features such as living area, number of bedrooms, and number of

bathrooms are related to the size of the house and are therefore likely to be correlated. These are the sort of issues that Ridge and Lasso regression can deal with.

The features and dummy variables were scaled using the *Z*-score method and the training set data. We also scaled the target values (i.e., the house prices) using the *Z*-score method and train set observations. (The latter is not necessary but will prove useful.)

When a plain vanilla linear regression is used, the results are those shown in Table 3.6. The mean squared error for the prediction of the prices of houses in the training set is 0.114. Since observations on the price have been scaled so that they have a variance of 1, this means that 1−0.114 or 88.6% of the variance of house prices in the training set is explained by the regression model.

For the data we are considering, it turns out that this regression model generalizes well. The mean squared error for the validation set was only a little higher than that for the training set at 0.117. However, linear regression with no regularization leads to some strange results because of the correlations between features. For example, it makes no sense that the weights for number of full bathrooms and number of bedrooms are negative.

We tried using Ridge regression with different values of the hyperparameter, λ. The impact of this on the prediction errors for the validation set is shown in Figure 3.8. As expected, the prediction error increases as λ increases. Values of λ in the range 0 to 0.1 might reasonably be considered because the increase in prediction errors is small when λ is in this range. However, it turns out that the improvement in the model is quite small for these values of λ. The average absolute value of the weights decreases from about 0.049 to about 0.046 as λ increases from 0 to 0.1. Even when λ is increased to 0.6 the average absolute value of the weights declines to only 0.039.

Lasso regression leads to more interesting results. Figure 3.9 shows how the error in the validation set changes as the value of the Lasso's λ increases from an initial value of zero. For small values of λ the error is actually less than when $\lambda = 0$, but as λ increases beyond about 0.02 the error starts to increase. A value of λ equal to 0.04 could be attractive. The loss in accuracy is quite small. The mean squared error for the validation set is only 12.0% of the total variance in the observations (compared with 11.7% when λ is set equal to 0 so that there is no regularization). However, when λ=0.04, 25 of the weights are zero and the average value of the weights is reduced to 0.034. (This is much better than the corresponding results using Ridge.)

Figure 3.8 Ridge regression results showing the mean squared error as a percent of the total squared error in the observations for the validation set (from Excel and Python)

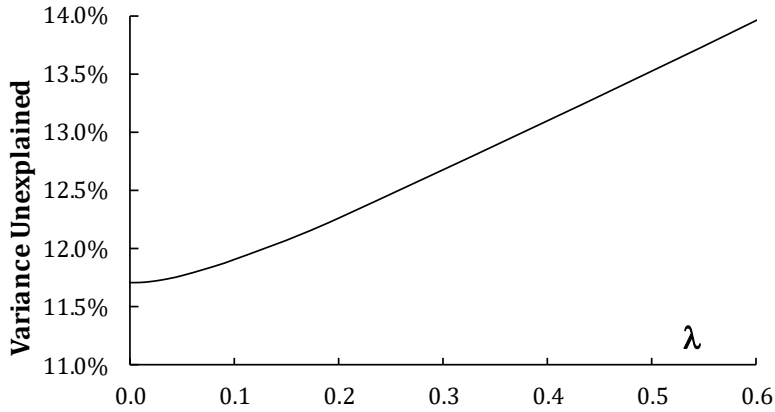
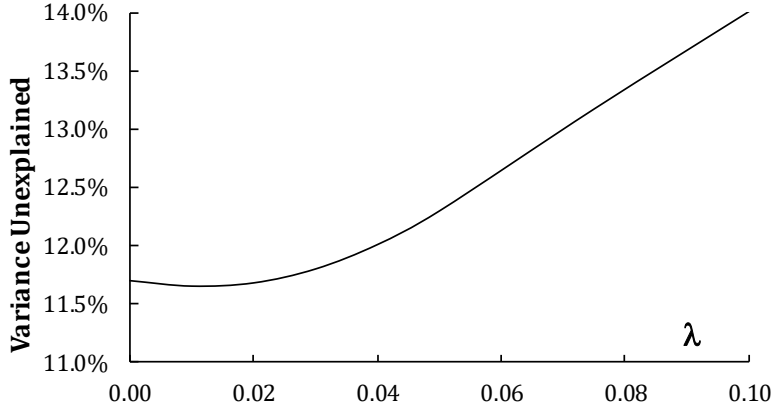


Figure 3.9 Lasso results for validation set for different values of λ (from Python)



If we are prepared to let the percentage mean squared error rise to about 14%, we can set λ equal to 0.1. This results in 30 of the weights becoming zero. The remaining non-zero weights are shown in Table 3.7. (The weights for wood deck and open porch were less than 0.005, but

not quite zero and are not shown in the table.) In Tables 3.6 and 3.7 it can be seen that overall quality and total living area are the most important predictors. A key point is that the negative weights in Table 3.6 that made no sense have been eliminated.

Table 3.7 Non-zero features in the Lasso model and their weights (after scaling) when λ=0.1 (from Python)

Feature	*Weight*
Lot area (square feet)	0.04
Overall quality (scale from 1 to 10)	0.30
Year built	0.05
Year remodeled	0.06
Finished basement (square feet)	0.12
Total basement (square feet)	0.10
First floor (square feet)	0.03
Living area (square feet)	0.30
Number of fireplaces	0.02
Parking spaces in garage	0.03
Garage area (square feet)	0.07
Neighborhoods (3 out of 25 non-zero)	0.01, 0.02, and 0.08
Basement quality	0.02

For the data we are considering, Elastic Net did not produce an improvement over Lasso. It is therefore likely that an analyst would in this case opt for one of the Lasso models. Once the model has been chosen its accuracy should be assessed using the test set. If Lasso with λ=0.04 is chosen, the mean squared error for the test data set is 12.5% (so that 87.5% of the variance in house prices is explained). If Lasso with λ=0.1 is chosen, the mean squared error for the test data set is 14.7% (so that 85.3% of the variance in house prices is explained).

3.9 Logistic Regression

As mentioned in Chapter 1 there are two types of supervised learning models: those that are used to predict a numerical variable and those that are used for classification. Up to now in this chapter, we have considered the problem of predicting a numerical variable. We now move on to the classification problem; that is, the problem of predicting which of two categories new observations will belong to. Logistic re-

gression is one of the tools that can be used for this. Other tools will be presented in Chapters 4, 5, and 6.

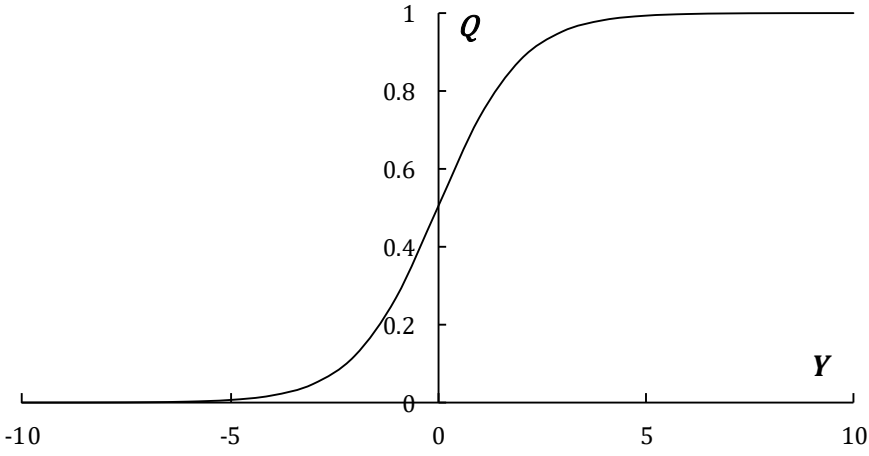
Suppose that there are a number of features X_j ($1 \leq j \leq m$), some of which may be dummy variables created from categorical features. Suppose further that there are two classes to which the observations can belong. One of the classes will be referred to as a positive outcome (typically this will be the thing we are trying to predict). The other class will be referred to as a negative outcome. An example of classification is the detection of junk e-mail from words included in the e-mail. A junk e-mail would be classified as a positive outcome and a non-junk e-mail as a negative outcome.

Logistic regression (also called logit regression) can be used to calculate the probability, Q, of a positive outcome. It does this by using the sigmoid function:

$$Q = \frac{1}{1 + e^{-Y}} \tag{3.6}$$

This is the S-shaped function shown in Figure 3.10. It has values between 0 and 1. When Y is very large and negative, e^{-Y} is very large and the function Q is close to zero. When Y is very large and positive, e^{-Y} is very small and Q is close to one.

Figure 3.10 The sigmoid function



We set Y equal to a constant (the bias) plus a linear combination of the features:

$$Y = a + b_1X_1 + b_2X_2 + \cdots + b_mX_m$$

so that the probability of a positive outcome is

$$Q = \frac{1}{1 + \exp\left(-a - \sum_{j=1}^{m} b_jX_j\right)} \tag{3.7}$$

When we are predicting the value of a target, as in the Iowa house price example, we can use mean square error as the objective function. When classifying observations, a different objective function is necessary. The maximum likelihood method in statistics is a way of choosing parameters from a set of observations in a way that maximizes the chance of the observations occurring. In the present situation, it leads to choosing the a and b_j so that

$$\sum_{\substack{\text{Positive}\\ \text{Outcomes}}} \ln(Q) + \sum_{\substack{\text{Negative}\\ \text{Outcomes}}} \ln(1 - Q) \tag{3.8}$$

is maximized. The first summation here is over all the observations which led to positive outcomes and the second summation is over all observations which led to negative outcomes. This function cannot be maximized analytically and gradient ascent (similar to gradient descent) methods must be used.

Earlier in this chapter we showed how regularization can simplify linear regression models and avoid overfitting. Regularization can be used in a similar way in logistic regression. Ridge regression (L2 regularization) involves adding $\lambda \sum_{j=1}^{m} b_j^2$ to the expression for Y in equation (3.6) so that the probability of a positive outcome becomes

$$Q = \frac{1}{1 + \exp\left(-a - \sum_{j=1}^{m} b_jX_j - \lambda \sum_{j=1}^{m} b_j^2\right)}$$

Similarly, Lasso regression (L1 regularization) leads to

$$Q = \frac{1}{1 + \exp\left(-a - \sum_{j=1}^{m} b_jX_j - \lambda \sum_{j=1}^{m} |b_j|\right)}$$

and Elastic Net leads to

$$Q = \frac{1}{1 + \exp\left(-a - \sum_{j=1}^{m} b_j X_j - \lambda_1 \sum_{j=1}^{m} b_j^2 - \lambda_2 \sum_{j=1}^{m} |b_j|\right)}$$

We emphasize that these formulas are used for estimating the bias and weights from the training set. Once this is done, equation (3.7) is used for predicting Q from the validation set, test set, or new data.

3.10 Decision Criteria

Once the model for estimating the probability of a positive outcome has been developed, it is necessary to choose a criterion for deciding whether a new observation should be classified as positive. It is tempting to maximize a simple accuracy measure: the percentage of observations that are correctly classified. But this does not always work well. Suppose we are trying to detect credit card fraud from features such as number of charges per day, types of purchases, and so on. If only one percent of transactions are fraudulent, we can obtain 99% accuracy simply by forecasting that all transactions are good!

The problem here is that there is a class imbalance. There are two classes:

- transaction good
- transaction fraudulent

and the first class is much bigger than the second. If the classes were equal in size (or approximately equal in size), using the accuracy measure we have just mentioned could be appropriate. Unfortunately, most of the time the classes we deal with are imbalanced.

One way of handling this problem is to create a balanced training set by under-sampling observations from the majority class. For example, in the situation we have just mentioned an analyst could form a training set by collecting data on 100,000 fraudulent transactions and pairing it with a random sample of 100,000 good transactions. Another approach involves over-sampling the minority class by creating synthetic observations. This is known as SMOTE (Synthetic Minority Over-sampling Technique).[6] Balancing the training set in one of these ways is not necessary for logistic regression but does make methods we will talk about later in this book such as SVM and neural networks, work better.

[6] See N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-Sampling Technique," *Journal of Artificial Intelligence Research*, 16 (2002), 321–357.

In practice, a balanced training set is often not used in logistic regression. It is important to keep in mind the purpose of the classification. Often the cost of classifying a new observation as positive when in fact it turns out to be negative is different from the cost of classifying it as negative when in fact it turns out to be positive. It is then a mistake to base decisions on whether Q is greater than or less than 0.5 and a threshold for Q that is different from 0.5 is likely to be appropriate. As we will explain with the example in the next section, it can be useful to present the decision maker with a range of alternative decision criteria.

3.11 Application to Credit Decisions

In this section, we consider a subset of the data provided by the company Lending Club on the performance of its loans. (This data and the analysis are at www-2.rotman.utoronto.ca/~hull.) Lending Club is a peer-to-peer lender that allows investors to lend money to borrowers without an intermediary being involved.[7]

Lending Club uses machine learning. We will attempt the challenging task of trying to improve on Lending Club's criteria by using machine learning ourselves. An extract from the data we use is shown in Table 3.8. In this example, we will be looking at only one model. It will therefore be sufficient to use a training set and a test set. (The validation set, it will be recalled, is necessary when several models are considered, and the analyst must choose between them.)

In the analyses given here and elsewhere in this book, good loans are defined as those listed as "Current" and defaulting loans as those listed as "Charged Off".[8] We define positive outcomes as those that lead to good loans and negative outcomes as those that lead to defaults. This is somewhat arbitrary. Some analysts would argue that we are trying to predict defaults and so defaults should be the positive outcome.[9]

The training set consists of 8,695 observations of which 1,499 were for loans that defaulted and the remaining 7,196 were for loans that proved to be good. The test set consists of 5,916 observations of which 1,058 were for loans that defaulted and the remaining 4,858 were for loans that were good.

---

[7] See https://www.lendingclub.com.

[8] Exercise 3.16 suggests an alternative (possibly better) classification where good loan as defined as "Fully Paid."

[9] See Exercise 3.14 for how the analysis changes when defaulting loans are labeled "positive."

We use four features:

- Home ownership
- Income per annum ($)
- Debt to income ratio (%)
- Credit score (FICO)

(One of these, home ownership, was categorical and was handled with a dummy variable that was 0 or 1.) Table 3.8 shows sample data. The weights estimated for the training set are shown in Table 3.9. The bias was estimated as −6.5645. The probability of a loan not defaulting is therefore given by equation (3.6) with

$$Y = -6.5645 + 0.1395X_1 + 0.0041X_2 - 0.0011X_3 + 0.0113X_4$$

Table 3.8 Training data set used to predict loan defaults

Home ownership, 1=owns, 0=rents X_1	*Income ('000s),* X_2	*Debt to income* X_3	*Credit score,* X_4	*Loan out-come*
1	44.304	18.47	690	Default
1	136.000	20.63	670	Good
0	38.500	33.73	660	Default
1	88.000	5.32	660	Good
……	……	……	……	……
……	……	……	……	……

Table 3.9 Optimal weights (see Excel or Python)

Feature	*Symbol*	*Weight,* b_i
Home ownership (0 or 1)	X_1	0.1395
Income ('000s)	X_2	0.0041
Debt to income ratio (%)	X_3	−0.0011
Credit score (FICO)	X_4	0.0113

The decision for determining whether a loan is acceptable or not can be made by setting a threshold, Z, for the value of Q so that:

- If $Q \geq Z$ the loan is predicted to be good
- If $Q < Z$ the loan is predicted to be bad

The results when a particular value of Z is applied to the test set can be summarized by what is referred to as a *confusion matrix*. This shows the relationship between predictions and outcomes. Tables 3.10, 3.11, and 3.12 show the confusion matrix for three different values of Z for the test set in our model (see Excel or Python results).

Table 3.10 Confusion matrix for test set when $Z = 0.75$

	Predict positive (no default)	*Predict negative (default)*
Outcome positive (no default)	77.59%	4.53%
Outcome negative (default)	16.26%	1.62%

Table 3.11 Confusion matrix for test set when $Z = 0.80$

	Predict positive (no default)	*Predict negative (default)*
Outcome positive (no default)	55.34%	26.77%
Outcome negative (default)	9.75%	8.13%

Table 3.12 Confusion matrix for test set when $Z = 0.85$

	Predict positive (no default)	*Predict negative (default)*
Outcome positive (no default)	28.65%	53.47%
Outcome negative (default)	3.74%	14.15%

The confusion matrix itself is not confusing but the terminology that accompanies it can be. The four elements of the confusion matrix are defined as follows:

- True Positive (TP): Both prediction and outcome are positive

- False Negative (FN): Prediction is negative, but outcome is positive
- False Positive (FP): Prediction is positive and outcome is negative
- True Negative (TN): Prediction is negative and outcome is negative

These definitions are summarized in Table 3.13

Table 3.13 Summary of definitions

	Predict positive outcome	*Predict negative outcome*
Outcome positive	TP	FN
Outcome negative	FP	TN

Ratios that can be defined from the table are:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}}$$

$$\text{True Positive Rate} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{True Negative Rate} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

$$\text{False Positive Rate} = \frac{\text{FP}}{\text{TN} + \text{FP}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Another measure sometimes calculated from these ratios is known as the *F-score* or *F1-score*. This is defined as

$$2 \times \frac{\text{P} \times \text{TPR}}{\text{P} + \text{TPR}}$$

where P is the precision and TPR is the true positive rate. This is an accuracy measure sometimes used for imbalanced data sets that focuses on how well positives have been identified.

The measures are shown in Table 3.14 for the three different values of Z that are considered in Tables 3.10 to 3.12.

Table 3.14 Ratios calculated from the confusion matrices in Tables 3.10 to 3.12 (see Excel or Python results)

	Z = 0.75	*Z = 0.80*	*Z = 0.85*
Accuracy	79.21%	63.47%	42.80%
True Positive Rate	94.48%	67.39%	34.89%
True Negative Rate	9.07%	45.46%	79.11%
False Positive Rate	90.93%	54.54%	20.89%
Precision	82.67%	85.02%	88.47%
F-score	88.18%	75.19%	50.04%

Accuracy is the percentage of observations that are classified correctly. It might be thought that maximizing accuracy must be the best strategy. But, as mentioned in the previous section, this is not necessarily the case. Indeed, in our example accuracy is maximized at 82.12% by simply classifying all observations as positive (i.e. always predicting no default and setting Z=0).

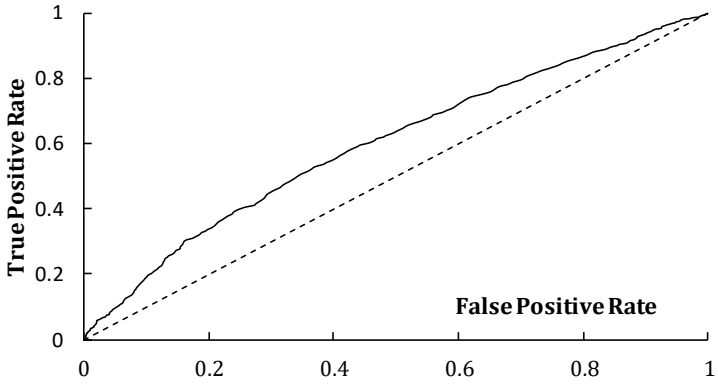
The *true positive rate*, which is also called the *sensitivity* or the *recall* is the percentage of positive outcomes that are correctly predicted. Like accuracy, this should not be the sole objective because it can be made one by classifying all observations as good.

The *true negative rate*, which is also called the *specificity*, is the proportion of negative outcomes that were predicted as negative. The *false positive rate* is one minus the true negative rate. It is the proportion of negative outcomes that were incorrectly classified. The *precision* is the proportion of positive predictions that prove to be correct.

There are a number of trade-offs. We can increase the true negative rate (i.e., identify a greater proportion of the loans that will default) only if we identify a lower proportion of the loans that prove to be good. Also, accuracy declines as the true negative rate increases.

The trade-offs are summarized in Figure 3.11 which plots the true positive rate against the false positive rate for every possible threshold, Z. This is known as the *receiver operating curve* (ROC). The *area under this curve* (AUC) is a popular way of summarizing the predictive ability of a model. If the AUC is 1.0, there is a perfect model where a 100% true positive rate can be combined with a 0% false positive rate. The dashed line in Figure 3.11 corresponds to an AUC of 0.5. This corresponds to models with no predictive ability. A model that makes random predictions would have an AUC of 0.5. Models with AUC < 0.5 are worse than random.

Figure 3.11 ROC curve showing the relationship between true positive rate and false positive rate for test set (see Excel or Python files)



For the data we have been considering, the Python implementation calculates an AUC of 0.6020 indicating that the model does has some small predictive ability. (Given that Lending Club has already used machine learning to make its lending decisions and we are using only four features, the AUC is quite encouraging.)

Note that, if we had set default as the positive outcome (perhaps because this is what we are trying to predict) and no-default as the negative outcome, the probabilities estimated for default and no-default would have been the same. The accuracy ratio would be the same, but the other ratios in Table 3.14 would change. The AUC would be the same. See Exercise 3.14 to understand this.

In deciding on the appropriate value for Z (i.e., positioning on the ROC) a lender has to consider the average profit from loans that do not default and the average loss from loans that default. Suppose for example that the profit from a loan that does not default is V, whereas the loss from cost of a defaulting loan is $4V$. The lender's profit is greatest when

$$V \times \text{TP} - 4V \times \text{FP}$$

is maximized. For the alternatives considered in Tables 3.10, 3.11, and 3.12 this is $12.55V$, $16.34V$, and $13.69V$, respectively. This indicates that of the three alternative Z values, $Z = 0.80$ would be most profitable.

3.12 The *k*-Nearest Neighbors Algorithm

Before closing this chapter, we mention a simple alternative to linear or logistic regression known as the *k*-nearest neighbors algorithm. This involves choosing a value for k and then finding the k observations whose features are most similar to the features from which we are making a prediction.

Suppose we are predicting the value of a house in a certain neighborhood from lot size and square feet of living area. We could set $k = 3$. We would then search for the three houses in our training set that are most similar to the house under consideration as far as lot size and living area are concerned. We could measure similarity by scaling the features and then using the Euclidean distance measure described in Chapter 2. Suppose that the prices of the three most similar houses are \$230,000, \$245,000, and \$218,000. The estimated value of the house would be set equal to the arithmetic average of these house prices, or \$231,000.

The algorithm can also be used for classification. Suppose that we are predicting whether a loan will be good from the four features in Table 3.8 and set $k = 10$. We would search for the 10 loans in our training set whose features are most similar to the loan under consideration. If eight of those loans proved to be good and two defaulted our prediction for the probability of no default would be 80%.

Summary

Linear regression is of course not a new machine learning technique. It has played a central role in empirical research for many years. Data scientists have adopted it as a predictive tool.

Machine learning applications often have many features some of which are highly correlated. Linear regression is then liable to produce a result which gives a large positive coefficient to values for one feature and a large negative coefficient to values for another correlated feature. We illustrated this with the prediction-of-salary example that was considered in Chapter 1.

One approach to reducing the magnitude of weights in a regression model is Ridge regression. Another is Lasso regression. The latter has the effect of reducing the weights of unimportant variables to zero. Elastic Net regression uses the ideas underlying both Ridge and Lasso regression and can be used to achieve the advantages of both (i.e., coef-

ficients that are smaller in magnitude and the elimination of unimportant variables).

Categorical variables can be accommodated in linear regression by creating dummy variables, one for each category. The dummy variable for an observation is set equal to one if the observation falls into the category and zero otherwise.

Logistic regression, like regular linear regression, has been used in empirical research for many years. It has become an important classification tool for data scientists. Typically, there are two classes. One is designated as "positive"; the other is designated as "negative." The S-shaped sigmoid function is used to define the probability of an observation falling into the positive class. An iterative search procedure is used to find a linear function of the feature values that when substituted into the sigmoid function does the best job in assigning a high probability to positive outcomes and a low probability to negative outcomes. The results of using logistic regression on the test data set can be summarized with what is termed a confusion matrix.

Once a logistic regression has been carried out it is necessary to decide how the results will be used. We illustrated this with a lending decision. It is necessary for the decision maker to define a *Z*-value. When the probability of a positive outcome from the loan is estimated to be greater than *Z*, the loan is accepted. When it is less than *Z*, the loan is rejected. There is a trade-off between success at identifying good loans and success at identifying loans that will default. Improving the latter tends to worsen the former, and vice versa. This trade-off can be summarized by a receiver operating curve (ROC) which relates the true positive rate (i.e., the percentage of the time a positive outcome is classified as positive) and the false positive rate (the percentage of the time a negative outcome is classified as positive).

A general point is we should not expect a machine learning model to make perfect predictions. The key test is whether their predictions are as good as, or better than, the predictions made by a human being. The popularity of machine learning models in a variety of different fields indicates that they must be passing this test.

SHORT CONCEPT QUESTIONS

3.1 What is the objective function in a "plain vanilla" linear regression?

3.2 How is the objective function changed for (a) Ridge regression, (b) Lasso regression, and (c) Elastic Net regression?

3.3 What is the main advantage of (a) Ridge regression and (b) Lasso Regression?
3.4 In predicting house prices, how would you handle a feature which is "yes" if a house has air conditioning and "no" if it does not?
3.5 In predicting house prices, how would you handle a feature which describes the lot as "no slope", "gentle slope", "moderate slope", and "severe slope."
3.6 In predicting house prices how would you handle a feature which identifies the neighborhood of the house.
3.7 Explain the meaning of the term "regularization." What is the difference between L1 and L2 regularization?
3.8 What is the sigmoid function?
3.9 What is the objective function in a logistic regression?
3.10 What is the definition of (a) the true positive rate, (b) the false positive rate, and (c) the precision?
3.11 What is plotted in an ROC? Explain the trade-offs it describes.
3.12 Explain what is meant by the dummy variable trap.

EXERCISES

3.13 Using the validation set in Table 1.2, calculate tables similar to Tables 3.2 and 3.3. Using scaled data calculate biases, weights, and mean squared errors for a
(a) Plain vanilla linear regression of salary on X, X^2, X^3, X^4, and X^5, where X is age.
(b) Ridge regression of salary on X, X^2, X^3, X^4, and X^5 with $\lambda = 0.02$, 0.05, and 0.1.
(c) Lasso regression of salary on X, X^2, X^3, X^4, and X^5 with $\lambda = 0.02$, 0.05, and 0.1.

3.14 Suppose that, in the Lending Club data, we define default as the positive outcome and no default as negative outcome.
(a) What effect does this have on the bias and weights? Show that the probability of default and no-default are unchanged.
(b) Choose Z values of 0.25, 0.20 and 0.15 for predicting defaults. Calculate confusion matrices and the ratios in Table 3.14.
(c) Use the Python implementation to confirm your answers to (a) and (b) and verify that the AUC is still 0.6020.

3.15 Extend the Iowa house price example by including additional features from the Original_Data.xlsx file in

www-2.rotman.utoronto.ca/~hull

As in the analysis in the text, choose the first 1,800 observations as the training set, the next 600 as the validation set, and the remainder as the test set. One additional feature should be Lot Frontage and you should consider alternative approaches for dealing with the missing observations. Another additional feature should be the categorical feature Lot Shape. Choose a model for prediction and calculate the accuracy for the test set. Repeat your analysis by randomly spitting data into training set, validation set, and test set.

3.16 The full data set for Lending Club is in the file Full_Data_Set.xlsx (see www-2.rotman.utoronto.ca/~hull). In the analysis in this chapter "good loans" are those listed as "Current" and defaulting loans are those listed as "Charged Off" (see column O in Full_Data_Set.xlsx). Other loans are not considered. Repeat the analysis in this chapter assuming that "good loans" are those listed as "Fully Paid" and defaulting loans are those listed as "Charged Off." Are your results better than those in this chapter? Choose additional features from the full data set and report on any improvements in your logistic regression results. (Make sure the features you choose would have known values at the time the loan was made.)

Chapter 4

Supervised Learning: Decision Trees

In this chapter, we continue our discussion of supervised learning by considering how decision trees can be used for prediction. Decision trees have a number of potential advantages over linear or logistic regression. For example:

- They correspond to the way many human beings think about a problem and are easy to explain to non-specialists.
- There is no requirement that the relationship between the target and the features be linear.
- The tree automatically selects the best features to make the prediction.
- A decision tree is less sensitive to outlying observations than a regression

The first part of this chapter will focus on the use of decision trees for classification. We use the Lending Club data, introduced in Chapter 3, to illustrate the methodology. We build on the Bayes' theorem material in Chapter 1 to explain what is known as the *naïve Bayesian classifier*. We then show how decision trees can be used for targets that are continuous variables using the Iowa house price data. After that we explain how different machine learning algorithms can be combined to

produce composite predictions. An important example of this is a *random forest*, which is created by generating many different decision trees and combining the results.

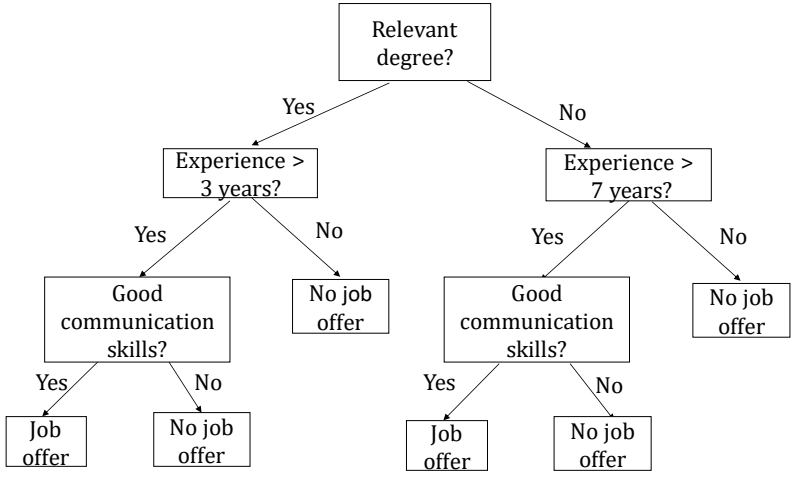
4.1 Nature of Decision Trees

A decision tree shows a step-by-step process for making predictions. Figure 4.1 shows a simple example concerned with the classification of applicants for a job into two categories:

- those who should get a job offer
- those who should be told "thanks but no thanks"

The example illustrates a key feature of decision trees. The decision is made by looking at features one at a time rather than all at once. The most important feature, relevant degree in our example, is considered first. After that experience and communication skills are used by the decision maker.

Figure 4.1 Simple example of a decision tree



An employer might subconsciously use a decision tree such as that in Figure 4.1 without ever formalizing it. When decision trees are used as a machine learning tool, the tree is constructed from historical data using an algorithm, as we will explain.

4.2 Information Gain Measures

What is the best feature to select at the first (root) node of a tree? Let's suppose that the purpose of our analysis is to replicate how employment decisions have been made in the past.[1] The feature to put at the root node is the one with the most information gain. Suppose we have lots of data on job applicants and find that we made a job offer to 20% of them. Suppose further that 50% of job applicants have a relevant degree. If both those with a relevant degree and those without a relevant degree had a 20% chance of receiving a job offer, there would be no information gain to knowing whether an applicant has a relevant degree. Suppose instead that

- 30% of those with relevant degrees received a job offer
- 10% of those without a relevant degree received a job offer

There is then clearly some information gain to knowing whether an applicant has a relevant degree.

One measure of information gain is based on *entropy*. This is a measure of uncertainty. If there are n outcomes and each outcome has a probability of $p_i (1 \le i \le n)$, entropy can be defined as

$$\text{Entropy} = -\sum_{i=1}^{n} p_i \log(p_i)$$

Here, to be consistent with the machine learning literature, we define "log" as log to the base 2.[2] Initially in our example there is a 20% chance of a job offer and an 80% chance of no job offer, so that:

$$\text{Entropy} = -[0.2 \times \log(0.2) + 0.8 \times \log(0.8)] = 0.7219$$

If a candidate has a relevant degree, this becomes

$$\text{Entropy} = -[0.3 \times \log(0.3) + 0.7 \times \log(0.7)] = 0.8813$$

If a candidate does not have a relevant degree, it becomes

$$\text{Entropy} = -[0.1 \times \log(0.1) + 0.9 \times \log(0.9)] = 0.4690$$

---

[1] A more sophisticated analysis might try to relate the performance of employees to the features known at the time the employment decision was made.

[2] The base used for the logarithm does not make a difference to the results as changing the base merely multiplies $\log(x)$ by the same constant for all x. When the base is 2, $\log(x) = y$ when $2^y = x$.

Because 50% of candidates have a relevant degree, the expected value of entropy, assuming that information on whether a candidate has a relevant degree will be obtained, is

$$0.5 \times 0.8813 + 0.5 \times 0.4690 = 0.6751$$

A measure of the information gain from finding out whether a candidate has a relevant degree is the expected uncertainty reduction. If uncertainty is measured by entropy, this is

$$0.7219 - 0.6751 = 0.0468$$

When constructing the decision tree, we first search for the feature that has the biggest information gain. This is put at the root of the tree. For each branch emanating from the root we then search again for the feature that has the biggest information gain. For both the "has relevant degree" and "does not have relevant degree," the feature that maximizes the expected information gain (reduction in expected entropy) in our example is the number of years of business experience. When the candidate has a relevant degree, the threshold for this feature that maximizes the expected information gain is 3 years. At the second level of the tree, the "has relevant degree" is therefore split into "experience > 3 years" and "experience ≤ 3 years" branches. For the branch corresponding to the candidate not having a relevant degree the threshold that maximizes the expected information gain is 7 years. The two subsequent branches therefore are "experience > 7 years" and "experience ≤ 7 years." We use the same procedure for building the rest of the tree. Note that numeric features can be used more than once. For example, the "Experience > 3 years" branch could lead to a further split into "Experience between 3 and 6 years" and "Experience greater than 6 years."

An alternative to entropy for quantifying information gain is the Gini measure. This is:

$$\text{Gini} = 1 - \sum_{i=1}^{n} p_i^2$$

It is used in the same way as entropy. In the example considered earlier, initially

$$\text{Gini} = 1 - 0.2^2 - 0.8^2 = 0.32$$

The expected Gini measure after finding out whether the candidate has a relevant degree is

$$0.5 \times (1 - 0.1^2 - 0.9^2) + 0.5 \times (1 - 0.3^2 - 0.7^2) = 0.30$$

The information gain (reduction in the expected Gini measure) is 0.02. Most of the time, the entropy and Gini measures give rise to similar trees.

4.3 Application to Credit Decisions

We now apply the decision tree approach using the entropy measure to the Lending Club data introduced in Chapter 3. It will be recalled that there are 8,695 observations in the training set and 5,916 in the test set. Of those in the training set, 7,196 were for good loans and 1,499 were for loans that defaulted. Without any further information, the probability of a good loan is therefore estimated from the training set as 7,196/8,695 or 82.76%. The initial entropy is therefore:

$$-0.8276 \times \log(0.8276) - 0.1724 \times \log(0.1724) = 0.6632$$

We will consider the same four features as in Chapter 3:

- A home ownership variable (= 1 if home owned; = 0 if rented)
- The applicant's income
- The applicant's debt to income ratio (dti)
- The applicant's credit score (FICO)

The first step in constructing a tree is to calculate the expected information gain (reduction in expected entropy) from each feature. Of the applicants, 59.14% own their own home while 40.86% rent. Loans were good for 84.44% of those who owned their own homes and 80.33% of those who rented. The expected entropy if home ownership (but no other feature) becomes known is therefore:

$$0.5914 \times [-0.8444 \times \log(0.8444) - 0.1556 \times \log(0.1556)]$$
$$+0.4086 \times [-0.8033 \times \log(0.8033) - 0.1967 \times \log(0.1967)] = 0.6611$$

The expected reduction in entropy is therefore a modest 0.6632−0.6611 = 0.0020.

The calculation of the expected entropy from income requires the specification of a threshold income. Define:

P_1: Probability that income is greater than the threshold
P_2: Probability that, if income is greater than the threshold, the borrower does not default
P_3: Probability that, if income is less than the threshold, the borrower does not default

The expected entropy is

$$P_1[-P_2\log(P_2) - (1-P_2)\log(1-P_2)]$$
$$+(1-P_1)[-P_3\log(P_3) - (1-P_3)\log(1-P_3)]$$

We carry out an iterative search to determine the threshold income that minimizes this expected entropy for the training set. It turns out that this is \$85,202. For this value of the threshold, P_1 = 29.93%, P_2 = 87.82%, and P_3 = 80.60% and expected entropy is 0.6573.

The results of all the information gain calculations are shown in Table 4.1. It can be seen that the FICO score with a threshold of 717.5 has the greatest information gain. It is therefore put at the root node of the tree. The initial branches of the tree correspond to FICO > 717.5 and FICO ≤ 717.5.

Table 4.1 Information gain from features to determine the root node (see Excel decision tree file for Lending Club case)

Feature	*Threshold value*	*Expected entropy*	*Information gain*
Home Ownership	N.A.	0.6611	0.0020
Income (\$'000s)	85.202	0.6573	0.0058
Debt to income (%)	19.87	0.6601	0.0030
Credit score (FICO)	717.5	0.6543	0.0088

For the next level of the tree we repeat the process. Table 4.2 shows the calculations for FICO > 717.5. In this case, the starting expected entropy is 0.4402. We must calculate the information gain of each of the three remaining features and consider the possibility of a further branch involving the FICO score (i.e., splitting the range of FICO scores above 717.5 into two categories) It turns out that income has the highest information gain and is therefore the feature that should be considered next, The threshold value of income is \$48,750.

Table 4.2 Information gain from features to determine the second level of the tree when FICO > 717.5 (see Excel file and Python implementation)

Feature	*Threshold value*	*Expected entropy*	*Information gain*
Home ownership	N.A.	0.4400	0.0003
Income ($'000s)	48.75	0.4330	0.0072
Debt to income (%)	21.13	0.4379	0.0023
FICO score	789	0.4354	0.0048

Table 4.3 shows the results when FICO ≤ 717.5. In this case the starting entropy is 0.7043. Income proves to be the feature with the most information gain and the threshold is $85,202. (Note that it is not always the case that the feature chosen is the same for both branches emanating from a node. Also, when the feature chosen does happen to be the same for both branches, it will not in general have the same threshold for both branches.)

Table 4.3 Information gain of features to determine the second level of tree when FICO ≤ 717.5 (see Excel file and Python implementation)

Feature	*Threshold value*	*Expected entropy*	*Information gain*
Home ownership	N.A.	0.7026	0.0017
Income ($'000s)	85.202	0.6989	0.0055
Debt to income (%)	16.80	0.7013	0.0030
FICO score	682	0.7019	0.0025

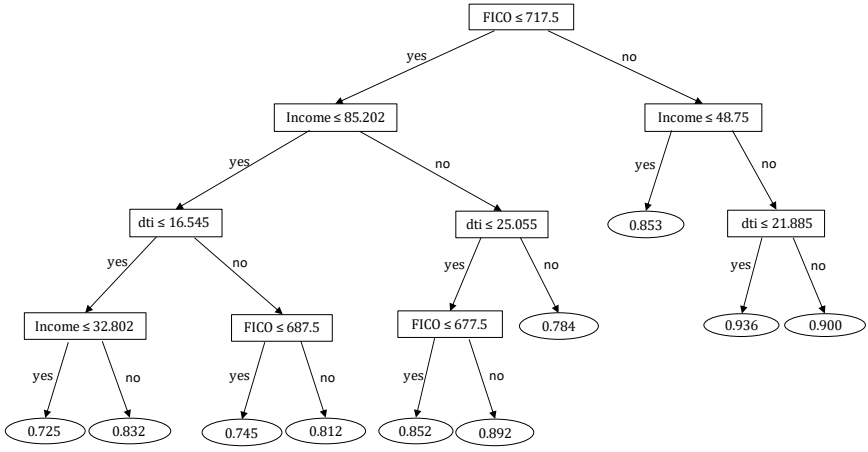
The construction of the tree continues in this way. The full tree produced by Sklearn's DecisionTreeClassifier is summarized in Figure 4.2. The final points reached by the tree (shown as ovals) are referred to as *leaves*. The numbers shown in the leaves in Figure 4.2 are the predicted probabilities of default. For example, if FICO > 717.5, Income > 48.75, and dti > 21.885, the estimated probability of no-default is 0.900. This is because there were 379 observations in the training set satisfying these conditions and 341 of them were good loans (341/379 = 0.900).

In determining the tree, it is necessary to set a number of hyperparameters. In this case:

- The maximum depth of the tree was set equal to 4. This means that there were at most four levels at which the tree splits.
- The minimum number of observations necessary for a split was set to 1,000.

The second hyperparameter explains why the tree sometimes stops before reaching the fourth level. For example, there are only 374 observations where FICO > 717.5 and income ≤ 48.75; there are only 893 observations where FICO > 717.5, income > 48.75, and dti ≤ 21.885; and there are only 379 observations where FICO > 717.5, income > 48.75, and dti > 21.885.

Figure 4.2 Decision tree for Lending Club. Numbers at the end of the tree are the probability of a good loan for training set



As with logistic regression we need a *Z*-value to define whether a loan is acceptable. Similarly to Section 3.11, we consider *Z*-values of 0.75, 0.80, and 0.85. An examination of Figure 4.2 shows that these three *Z*-values correspond to the following criteria:

- *Z* = 0.75: predict that all loans are good except those for which (a) income ≤ $85,202, dti > 16.545, and FICO ≤ 687.5 and (b) income ≤ $32,802, dti ≤ 16.545, and FICO ≤ 717.5.
- *Z* = 0.80: Same as Z = 0.75 except that loans where FICO ≤ 717.5, income > $85,202, and dti > 25.055 are not considered good.

- Z=0.85: Predict loans are good loans when (a) FICO > 717.5 or (b) FICO ≤ 717.5, income > $85,202, and dti ≤ 25.055

It is interesting to note that a tree can give inconsistent predictions. For example, when FICO=700, dti=10, and Income =30, the no-default probability predicted by the tree is 0.725. But when the dti value is changed from 10 to 20 (a worse value) the no-default probability increases to 0.812.

Tables 4.4 to 4.6 give confusion matrices for the test set when Z is 0.75, 0.80, and 0.85, respectively, and Table 4.7 provides the ratios introduced in Section 3.11.

Figure 4.3 shows the ROC curve. The marked points correspond to 11 ranges within which the threshold Z can be chosen. If $Z \leq 0.725$, all loans are accepted; if $0.725 < Z \leq 0.745$, we accept all loans except those corresponding to the 0.725 leaf in Figure 4.2; if $0.745 < Z \leq 0.784$ we accept all loans except those corresponding to the 0.725 and 0.745 leaves in Figure 4.2; and so on. Finally, if $Z > 0.936$ we accept no loans. The AUC calculated in the Python implementation is 0.5948, marginally worse than the 0.6020 calculated for the logistic regression model.

Table 4.4 Confusion matrix for test set when Z = 0.75 (See Python or Excel files)

	Predict positive (no default)	*Predict negative (default)*
Outcome positive (no default)	62.42%	19.69%
Outcome negative (default)	11.07%	6.81%

Table 4.5 Confusion matrix for test set when Z = 0.80 (See Python or Excel files)

	Predict positive (no default)	*Predict negative (default)*
Outcome positive (no default)	59.47%	22.65%
Outcome negative (default)	10.45%	7.44%

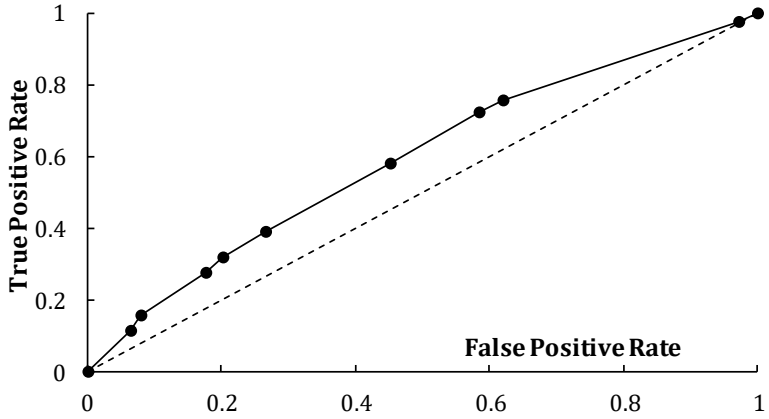
Table 4.6 Confusion matrix for test set when $Z = 0.85$ (See Python or Excel files)

	Predict positive (no default)	*Predict negative (default)*
Outcome positive (no default)	32.15%	49.97%
Outcome negative (default)	4.73%	13.15%

Table 4.7 Ratios calculated from Tables 4.4 to 4.6 (See Python or Excel files)

	Z=0.75	*Z=0.80*	*Z=0.85*
Accuracy	69.24%	66.90%	45.30%
True Positive Rate	76.02%	72.42%	39.15%
True Negative Rate	38.09%	41.59%	73.53%
False Positive Rate	61.91%	58.41%	26.47%
Precision	84.94%	85.06%	87.17%
F-score	80.23%	78.23%	54.03%

Figure 4.3 Trade-off between true positive rate and false positive rate for decision tree approach (see Python or Excel files)



4.4 The Naïve Bayes Classifier

We introduced Bayes' theorem in Chapter 1. Bayesian learning involves using Bayes' theorem to update probabilities. For example, Chapter 1 showed how Bayes' theorem can be used in the identification of fraudulent transactions.

The tree in Figure 4.2 can be regarded as an example of Bayesian learning. The probability of a good loan in the training set with no information about the features is 0.8276. The probability that the FICO score will be greater than 717.5, conditional on the loan being good, is 0.2079 and the unconditional probability that FICO is greater than 717.5 is 0.1893. From Bayes' theorem the probability of a good loan conditional on FICO > 717.5 is:

$$= \frac{\text{Prob}(\text{FICO} > 717.5|\text{good loan}) \times \text{Prob}(\text{good loan})}{\text{Prob}(\text{FICO} > 717.5)}$$

$$= \frac{0.2079 \times 0.8276}{0.1893} = 0.9089$$

Other (more complicated) Bayesian calculations can be used to update probabilities further. For example, at the next step we can calculate the probability of a good loan conditional on both FICO > 717.5 and Income > $48,750.

The *Naïve Bayes Classifier* is a procedure that can be used if the values of the features for observations classified in a particular way can be assumed to be independent. If C is a classification result and x_j is the value of the jth feature ($1 \leq j \leq m$), we know from Bayes theorem that

$$\text{Prob}(C|x_1, x_2, \ldots, x_m) = \frac{\text{Prob}(x_1, x_2, \ldots, x_m|C)}{\text{Prob}(x_1, x_2, \ldots x_m)}\text{Prob}(C)$$

Because of the independence assumption this reduces to

$$\text{Prob}(C|x_1, x_2, \ldots, x_m) = \frac{\text{Prob}(x_1|C)\text{Prob}(x_2|C)\ldots\text{Prob}(x_m|C)}{\text{Prob}(x_1, x_2, \ldots x_m)}\text{Prob}(C)$$

This shows that if we know the probability of each feature conditional on the classification, we can calculate probability of the classification conditional on a particular mixture of features occurring.

As a simple example of this, suppose that the unconditional probability of a good loan is 85% and that there are three independent features

when a loan is being assessed. These are:

- Whether the applicant owns a house (denoted by H). The probability of the applicant owning a house if the loan is good is 60% whereas the probability of the applicant owning her own house if the loan defaults is 50%.
- Whether the applicant has been employed for more than one year (denoted by E). The probability of the applicant being employed for more than one year if the loan is good is 70% whereas the probability of this if the loan defaults is 60%.
- Whether there are two applicants or only one (denoted by T). The probability of two applicants when the loan is good is 20% whereas the probability of two applicants when the loan defaults is 10%.

Consider an applicant that is able to check all three boxes. She owns a house, she has been employed for more than one year, and she is one of two applicants for the same loan. Assuming the features are independent across good loans and across defaulting loans:

$$\text{Prob}(\text{Good Loan}|\text{H, E, T}) = \frac{0.6 \times 0.7 \times 0.2}{\text{Prob}(\text{H and E and T})} \times 0.85$$

$$= \frac{0.0714}{\text{Prob}(\text{H and E and T})}$$

$$\text{Prob}(\text{Defaulting Loan}|\text{H, E, T}) = \frac{0.5 \times 0.6 \times 0.1}{\text{Prob}(\text{H and E and T})} \times 0.15$$

$$= \frac{0.0045}{\text{Prob}(\text{H and E and T})}$$

Because the probability of a good loan and the probability of a defaulting loan must add to one, we do not need to calculate the value of Prob (H and E and T). The probability of a good loan is

$$\frac{0.0714}{0.0714 + 0.0045} = 0.941$$

and the probability of a defaulting loan is

$$\frac{0.0045}{0.0714 + 0.0045} = 0.059$$

For an applicant who checks all three boxes the probability of a good loan rises from 85% to just over 94%.

We can also use the naïve Bayes classifier with continuous distributions. Suppose we want to use the data in Chapter 3 to produce a loan forecast using two features: FICO score and income. We assume that these features are independent both for data on good loans and data on defaulting loans.[3] Table 4.8 shows the mean and standard deviation of the FICO score and income conditional on a good loan and a defaulting loan.

Table 4.8 Statistics on FICO score and income conditional on loan result. Income is measured in $'000s

Loan result	*Mean FICO*	*SD FICO*	*Mean Income*	*SD Income*
Good loan	696.19	31.29	79.83	59.24
Defaulting loan	686.65	24.18	68.47	48.81

Consider an individual who has a FICO score of 720 and an income ('000s) of 100. Conditional on a loan being good the FICO score has a mean of 696.19 and a standard deviation of 31.29. Assuming a normal distribution, the probability density for the individual's FICO score conditional on the loan being good is

$$\frac{1}{\sqrt{2\pi} \times 31.29} \exp\left(-\frac{(720 - 696.19)^2}{2 \times 31.29^2}\right) = 0.00954$$

Similarly, assuming a normal distribution, the probability density for income conditional on the loan being good is[4]

$$\frac{1}{\sqrt{2\pi} \times 59.24} \exp\left(-\frac{(100 - 79.83)^2}{2 \times 59.24^2}\right) = 0.00636$$

[3] The independence assumption is an approximation. For loans that defaulted, the correlation between credit score and income is about 0.07 and, for loans that are good, it is about 0.11.

[4] It would be better to assume a lognormal distribution for income. We have not done this to keep the example simple.

The probability density for the credit score conditional on the loan defaulting is

$$\frac{1}{\sqrt{2\pi} \times 24.18} \exp\left(-\frac{(720 - 686.65)^2}{2 \times 24.18^2}\right) = 0.00637$$

Similarly, the probability density for the income conditional on the loan defaulting is

$$\frac{1}{\sqrt{2\pi} \times 48.81} \exp\left(-\frac{(100 - 68.47)^2}{2 \times 48.81^2}\right) = 0.00663$$

The unconditional probability of the loan being good is 0.8276 and the unconditional probability of it being bad is 0.1724. The probability of a loan being good conditional on a credit score of 720 and an income ($'000) of 100 is

$$\frac{0.00954 \times 0.00636 \times 0.8276}{Q} = \frac{5.020 \times 10^{-5}}{Q}$$

where Q is the probability density of the observation ($100,000 income and credit score equal to 720).

The corresponding conditional probability of a bad loan is

$$\frac{0.00637 \times 0.00663 \times 0.1724}{Q} = \frac{0.729 \times 10^{-5}}{Q}$$

Because the two probabilities must add up to one we know that the probability of the loan being good is 5.020/(5.020+0.729) or 0.873. (One of the attractive features of the naïve Bayes classifier is that we do not need to calculate Q to obtain this result.)

The naïve Bayes classifier is easy to use when there are a large number of features. It makes a simple set of assumptions. These assumptions are unlikely to be completely true in practice. However, the approach has been found to be useful in a variety of situations. For example, it is quite effective in identifying spam when word frequencies are used as features. (See Chapter 8 for the use of the naïve Bayes Classifier in natural language processing.)

4.5 Continuous Target Variables

So far we have considered the use of decision trees for classification. We now describe how they can be used to predict the value of a continuous variable. Suppose that the feature at the root node is X and the threshold value for X is Z. We choose X and Z to minimize the expected mean squared error (mse) in the prediction of the target for the training set. In other words, we minimize

$$\text{Prob}(X \geq Z) \times (\text{mse if } X \geq Z) + \text{Prob}(X < Z) \times (\text{mse if } X < Z)$$

The feature at the next node and its threshold are chosen similarly. The value predicted at a tree leaf is the average of the values for the observations corresponding to the leaf.

We will illustrate this procedure for the Iowa house price data considered in Chapter 3. To keep the example manageable, we consider only two features:

- Overall quality (scale 1 to 10)
- Living area (square feet)

(These were identified as the most important features by linear regression in Chapter 3.) As in Chapter 3 we divide up the data (2,908 observations in total) so that there are 1,800 observations in the training set, 600 in the validation set, and 508 in the test set. The mean and standard deviation of the prices of houses in the training set ('000s) are \$180.817 and \$77.201.

First, we determine the feature to put at the root node and its threshold. For each of the two features, we use an iterative search procedure to calculate the optimal threshold. The results are shown in Table 4.9. The expected mse is lowest for overall quality which has an optimal threshold of 7.5. This feature and its threshold therefore define the root node. (Because overall quality is an integer all thresholds between 7 and 8 are equivalent. A similar point applies to living area.)

Table 4.10 considers the best feature when overall quality ≤ 7.5. It turns out that, even though overall quality has been split at the root node, it is best to split it again at the second level using a threshold of 6.5. Table 4.11 shows that when overall quality > 7.5, it is also best to split overall quality again, this time with a threshold of 8.5. Following the two splits on overall quality, it is optimal to split living area at each of the decision points encountered at the third level.

Table 4.9 Expected mean squared error at root node. House prices are measured in thousands of dollars for the purposes of calculating mse (see Excel decision tree file for Iowa house price case). Threshold = *Z*.

Feature	*Z*	*No. of obs < Z*	*mse of obs < Z*	*No. of obs ≥ Z*	*mse of obs ≥ Z*	*E(mse)*
Overall Quality	7.5	1,512	2,376	288	7,312	3,166
Living (sq. ft.)	1,482.5	949	1,451	851	6,824	3,991

Table 4.10 Expected mean squared error at second level when overall quality ≤ 7.5. House prices are measured in thousands of dollars for the purposes of calculating mse (see Excel decision tree file for Iowa house price case). Threshold = *Z*.

Feature	*Z*	*No. of obs < Z*	*mse of obs < Z*	*No. of obs ≥ Z*	*mse of obs ≥ Z*	*E(mse)*
Overall Quality	6.5	1,122	1,433	390	1,939	1,564
Living (sq. ft.)	1,412.5	814	1,109	698	2,198	1,612

Table 4.11 Expected mean squared error at second level when overall quality >7.5. House prices are measured in thousands of dollars for the purposes of calculating mse (see Excel decision tree file for Iowa house price case). Threshold = *Z*.

Feature	*Z*	*No. of obs < Z*	*mse of obs < Z*	*No. of obs ≥ Z*	*mse of obs ≥ Z*	*E(mse)*
Overall Quality	8.5	214	3,857	74	8,043	4,933
Living (sq. ft.)	1,971.5	165	3,012	123	8,426	5,324

The tree produced by Sklearn's DecisionTreeRegressor is shown in Figure 4.4. The maximum depth of the tree was specified as three. The average house prices and root mean squared errors are shown on the

tree for each of the leaf nodes (see circles). The overall root mean squared error of the predictions for the training set, validation set, and test set are shown in Table 4.12. It can be seen that the model generalizes quite well.

The root mean squared errors are quite large because the tree in effect divides all houses into only eight clusters. With more features and more depth, more clusters of houses would be considered. However, the number of houses in some clusters would then be quite small and the calculated average house price might be unreliable. For better results a larger data set is likely to be necessary.

Figure 4.4 Decision tree for calculating house prices. The house price predictions and the rmse's are shown in the final (circular) leaf nodes)

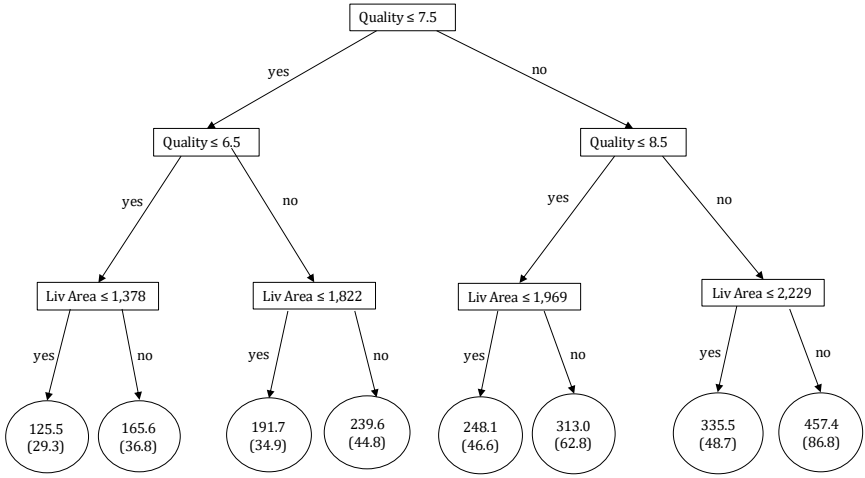


Table 4.12 Comparison of results for training set, validation set, and test set (see Python)

	Root mean squared error of house price ($'000s)
Training set	38.66
Validation set	40.46
Test set	39.05

Finally, note that we can adjust our calculations so that more than two branches are considered at a node. This is appropriate for a categorical feature with more than two possible values. For features that can take a range of values, we can consider N branches ($N > 2$) at a node and maximize the information gain over $N - 1$ thresholds.

4.6 Ensemble Learning

When the predictions from several different machine learning algorithms are aggregated, the result can be better than the predictions from any one of the algorithms. The improvement in the predictions depends on the correlation between the estimates produced from the algorithms. If two algorithms always produce the same predictions, there is clearly nothing to be gained by using both of them. But, if this is not the case, there is potentially some value in producing a composite prediction that uses the results from both algorithms. Using two or more algorithms to make predictions is known as *ensemble learning*.

Suppose that you have a biased coin which when tossed has a 52% chance of giving one result (heads or tails, you do not know which) and a 48% of giving the other result. If you want to know whether it is heads or tails that is more likely, you could toss the coin once, but this would not give much guidance. If you toss the coin 1,000 times there is a probability of about 90% that, if heads has a probability of 0.52, you will see more heads than tails. Similarly, if it is tails that has a probability of 0.52, there is a probability of about 90% that you will see more tails than heads. This illustrates that 1,000 weak learners can be combined to produce a learner where the predictions are reliable. Of course, in this example the learners are independent of each other. In machine learning different learning algorithms are unlikely to be totally independent and so the prediction improvement will not in general be as good as for the coin tossing example.

There are a number of ways of combining predictions. Sometimes a (possibly weighted) average of the predictions is appropriate. When each learner recommends a particular classification, we can use majority voting (i.e., the class that is recommended most can be chosen).

Bagging

Bagging involves using the same algorithm but training it on different random samples of the training set or the features. We might have 200,000 observations in the training set and randomly sample 100,000 observations 500 times to get 500 subsets of the training set. We then train the model on each subset in the usual way. The sampling is nor-

mally done with replacement so that the same observation may appear more than once in a subset. (If the sampling is done without replacement the method is referred to as *pasting*.)

We can also create many new models by sampling (without replacement) from the features. For example, if there are 50 features, we could create 100 models each with 25 features. Sometimes models are created by random sampling from both features and observations.

Random Forests

A random forest as its name implies is an ensemble of decision trees. The trees are often created by sampling from the features or observations using the bagging approach just mentioned. Each tree gives a suboptimal result but overall the prediction is usually improved.

Another approach to creating a random forest is to randomize the thresholds used for features rather than search for the best possible threshold. This can be computationally efficient as finding the optimal feature threshold at each node can be time consuming.

The importance of each feature in a random forest can be calculated as the weighted average information gain (as measured by entropy or Gini) with weights that are proportional to the number of observations considered at a node.

Boosting

Boosting refers to an ensemble method where prediction models are used sequentially, each trying to correct the errors in the previous one.

Consider the loan classification problem we looked at earlier. We might create a first classification in the usual way. We then increase the weight given to misclassified observations and create a new set of predictions; and so on. The predictions are combined in the usual way except that the weight given to a prediction depends on its accuracy. This procedure is known as *AdaBoost*.

A different approach from AdaBoost is *gradient boosting*. At each iteration, gradient boosting tries to fit a new predictor to the errors made by the previous predictor. Suppose there are three iterations. The final prediction is the sum of the three predictors. This is because the second predictor estimates errors in the first predictor and the third predictor estimates errors in a predictor that equals the sum of the first two predictors.

Summary

A decision tree is an algorithm for classification or predicting the value of variable. Features are considered in order of the information gain they provide. For classification, two alternative measures of information gain are entropy and Gini. When the value of a variable is being predicted, the information gain is measured by the improvement in expected mean squared error.

In the case of a categorical feature, the information gain arises from knowledge of the feature's label (e.g., whether a potential borrower's home is owned or rented). In the case of numerical features, it is necessary to determine one (or more) thresholds defining two (or more) ranges for the feature's value. These thresholds are determined so that the expected information gain is maximized.

A decision tree algorithm first determines the optimal root node of the tree using the "maximize information gain" criterion we have just described. It then proceeds to do the same for subsequent nodes. The ends of the final branches of the tree are referred to a leaf nodes. When the tree is used for classification, the leaf nodes contain the probabilities of each category being the correct. When a numerical variable is being predicted, the leaf nodes give the average value of the target. The geometry of the tree is determined with the training set, but statistics concerned with its accuracy should (as always in machine learning) come from the test set.

Sometimes more than one machine learning algorithm is used to make predictions. The results can then be combined to produce a composite prediction. This is referred to as the use of ensemble methods. A random forest machine learning algorithm is created by building many different trees and combining the results. The trees can be created by sampling from the observations or from the features (or both). They can also be created by randomizing the thresholds.

Bagging is the term used when different subsets of the observations or features in the training set are used to create multiple models. Boosting is a version of ensemble where prediction models are chosen sequentially, and each model is designed to correct errors in the previous model. In classification one way of doing this is to increase the weight of observations that are misclassified. Another is to use one machine learning model to predict the errors given by another model.

SHORT CONCEPT QUESTIONS

4.1 What are the main differences between the decision tree approach to prediction and the regression approach?
4.2 How is entropy defined?
4.3 How is the Gini measure defined?
4.4 How is information gain measured?
4.5 How do you choose the thresholds for a numerical variable in a decision tree?
4.6 What is the assumption underlying the naïve Bayes classifier?
4.7 What is an ensemble method?
4.8 What is a random forest?
4.9 Explain the difference between bagging and boosting.
4.10 "The decision tree algorithm has the advantage that it is transparent." Explain this comment.

EXERCISES

4.11 What strategy corresponds to a *Z*-value of 0.9 in Figure 4.2? What is the confusion matrix when this *Z*-value is used on the test data?
4.12 For the naïve Bayes classifier data in Table 4.8, what is the probability of a default when the credit score is 660 and the income is $40,000?
4.13 Python exercise: Similarly, to Exercise 3.16, determine the effect on the decision tree analysis of defining good loans as "Fully Paid" rather than "Current." Investigate the effect of adding more features to the analysis.
4.14 Test the effect of changing (a) the maximum depth of the tree and (b) the minimum number of samples necessary for a split on the decision tree results for Lending Club using Sklearn's DecisionTreeClassifier.
4.15 Choose an extra feature in addition to the two considered in Figure 4.4 and construct a tree using all three features. Compare the results with those in Section 4.5. Use Sklearn's DecisionTreeRegressor.

Chapter 5

Supervised Learning: SVMs

This chapter considers another popular category of supervised learning models known as a support vector machines (SVMs). Like decision trees, SVMs can be used for either classification or for the prediction of a continuous variable.

We first consider linear classification where a linear function of the feature values is used to separate observations into two categories. We then move on to explain how non-linear separation can be achieved. Finally, we show that, by reversing the objective, we can use SVM for predicting the value of a continuous variable, rather than for classification.

5.1 Linear SVM Classification

To describe how linear classification works, we consider a simple situation where loans are classified into good loans and defaulting loans by considering only two features: credit score and income of the borrower. A small amount of illustrative data is shown in Table 5.1. This is a balanced data set in that there are five good loans and five loans that defaulted. SVM does not work well for a seriously imbalanced data set

and procedures such as those mentioned in Section 3.10 sometimes have to be used to create a balanced data set.

Table 5.1 Loan data set to illustrate SVM

Credit score	*Adjusted credit score*	*Income ('000s)*	*Default = 0; good loan = 1*
660	40	30	0
650	30	55	0
650	30	63	0
700	80	35	0
720	100	28	0
650	30	140	1
650	30	100	1
710	90	95	1
740	120	64	1
770	150	63	1

The first step is to normalize the data so that the weight given to each feature is the same. In this case, we will take a simple approach and subtract 620 from the credit score. This provides approximate normalization because the adjusted credit scores range from 30 to 150 while incomes range from 28 to 140.

Figure 5.1 gives a scatter plot of the loans. Defaulting loans (represented by the circles) tend to be closer to the origin than the good loans (represented by the squares). We can separate the observations into two groups by drawing a straight line such as the one shown in the figure. Loans that are to the north-east of the line are good. Those to the south-west of the line default. (Note that this is an idealized example. As we discuss later, the sort of perfect separation indicated in Figure 5.1 is not usually possible.)

There is some uncertainty about where the line in Figure 5.1 should be positioned. We could move it sideways or change its gradient a little and still achieve perfect separation. SVM handles this uncertainty by using a pathway. The optimal pathway is the one that has maximum width. The line separating the observations is the middle of the pathway.

Figure 5.2 shows the optimal pathway for the data in Table 5.1. Note that adding more observations that are correctly classified by the pathway (i.e., good loans to the north-east of the pathway or defaulting loans to the south-west) does not affect the optimal pathway. The critical points are those on the edge of the pathway. These are referred to as

support vectors. For the data we are considering the support vectors are those defined by the third, seventh, and ninth observations.

Figure 5.1 Data set in Table 5.1. Circles represent loans that defaulted while squares represent good loans

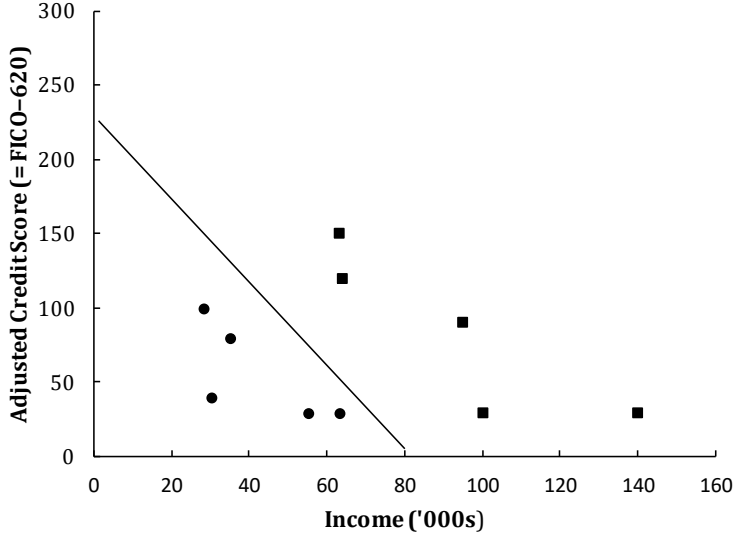
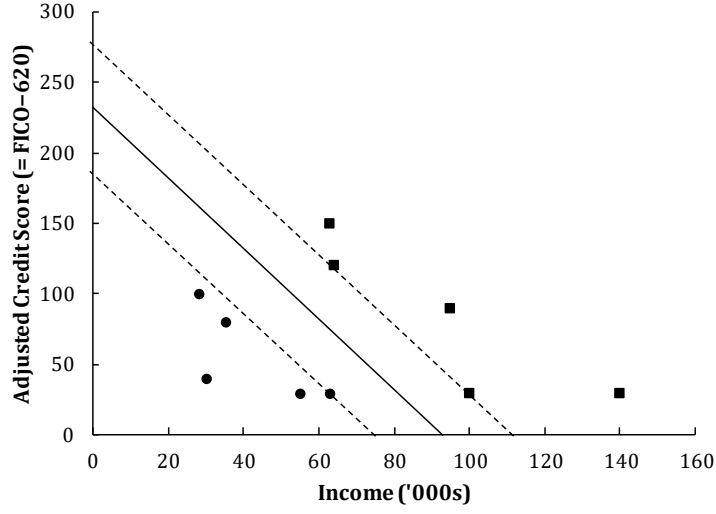


Figure 5.2 Optimal pathway for data in Table 5.1



The solid line in Figure 5.2 is the center of the pathway. It is the line that would be used to separate new observations into those that are predicted to be good loans and those that are expected to default.

To show how the optimal pathway can be determined in the two-feature case, we suppose that the features are x_1 and x_2, and the equations for the upper and lower edges of the pathway are:[1]

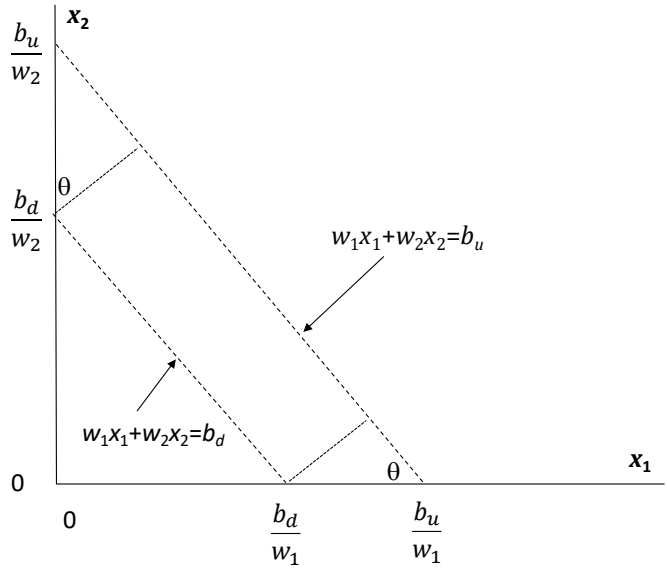
$$w_1 x_1 + w_2 x_2 = b_u \tag{5.1}$$

and

$$w_1 x_1 + w_2 x_2 = b_d \tag{5.2}$$

where $w_1, w_2, b_u,$ and b_d are constants. These definitions are illustrated in Figure 5.3.

Figure 5.3 Calculation of path width in general situation



Defining the angle θ as indicated, we see from the lower triangle involving θ in Figure 5.3 that the width of the path, *P*, can be written as

[1] Note that this material is often presented with b_u and b_d having the opposite sign to that here. This is just a notational issue and makes no difference to the model.

$$P = \left(\frac{b_u}{w_1} - \frac{b_d}{w_1}\right)\sin\theta = \left(\frac{b_u - b_d}{w_1}\right)\sin\theta$$

so that

$$\sin\theta = \frac{Pw_1}{b_u - b_d}$$

From the upper triangle involving θ, the width of the path, *P*, can also be written as

$$P = \left(\frac{b_u}{w_2} - \frac{b_d}{w_2}\right)\cos\theta = \left(\frac{b_u - b_d}{w_2}\right)\cos\theta$$

so that

$$\cos\theta = \frac{Pw_2}{b_u - b_d}$$

Because $\sin^2\theta + \cos^2\theta = 1$,

$$\left(\frac{Pw_1}{b_u - b_d}\right)^2 + \left(\frac{Pw_2}{b_u - b_d}\right)^2 = 1$$

so that

$$P = \frac{b_u - b_d}{\sqrt{w_1^2 + w_2^2}}$$

We can multiply $w_1, w_2, b_u,$ and b_d by the same constant without changing the equations (5.1) and (5.2) for the upper and lower boundary of the path. We can choose this constant so that $b_u - b_d = 2$. We can then set

$$b_u = b + 1 \tag{5.3}$$

and

$$b_d = b - 1 \tag{5.4}$$

The equation for the line defining the middle of the pathway becomes

$$w_1 x_1 + w_2 x_2 = b$$

and the width of the pathway becomes

$$P = \frac{2}{\sqrt{w_1^2 + w_2^2}}$$

This shows that the width of the path can be maximized by minimizing by $\sqrt{w_1^2 + w_2^2}$, or equivalently $w_1^2 + w_2^2$, subject to the constraint that the pathway separates the observations into the two classes and equations (5.1) to (5.4) are satisfied.

For the example in Table 5.1, we can set x_1 equal to income and x_2 equal to credit score. All good loans must be to the north-east of the pathway while all defaulting loans must be to the south-west of the pathway. This means that, if a loan is good, its income and credit score must satisfy

$$w_1 x_1 + w_2 x_2 \geq b + 1$$

while if the loan defaults they must satisfy

$$w_1 x_1 + w_2 x_2 \leq b - 1$$

From Table 5.1, we must therefore have

$$\begin{aligned}
30w_1 + 40w_2 &\leq b - 1 \\
55w_1 + 30w_2 &\leq b - 1 \\
63w_1 + 30w_2 &\leq b - 1 \\
35w_1 + 80w_2 &\leq b - 1 \\
28w_1 + 100w_2 &\leq b - 1 \\
140w_1 + 30w_2 &\geq b + 1 \\
100w_1 + 30w_2 &\geq b + 1 \\
95w_1 + 90w_2 &\geq b + 1 \\
64w_1 + 120w_2 &\geq b + 1 \\
63w_1 + 150w_2 &\geq b + 1
\end{aligned}$$

Minimizing $w_1^2 + w_2^2$ subject to these constraints gives b = 5.054, w_1 = 0.05405, w_2 = 0.02162. The middle of pathway in Figure 5.2 is therefore

$$0.05405x_1 + 0.02162x_2 = 5.054$$

This is the line that would be used to separate good loans from bad loans. The width of the pathway, *P*, is 34.35.

The analysis we have given can be extended to more than two features. If we have *m* features, the objective function to be minimized is

$$\sum_{j=1}^{m} w_j^2$$

If x_{ij} is the value of the *j*th feature for the *i*th observation, the constraint that must be satisfied is

$$\sum_{j=1}^{m} w_j x_{ij} \geq b + 1$$

when there is a positive outcome for observation *i* (which in our example occurs when the loan does not default) and

$$\sum_{j=1}^{m} w_j x_{ij} \leq b - 1$$

When there is a negative outcome for observation *i* (which in our example occurs when a loan defaults).

Minimizing the objective function subject to the constraints involves a standard numerical procedure known as *quadratic programming*.

5.2 Modification for Soft Margin

What we have considered so far is referred to as *hard margin classification* because the pathway divides the observations perfectly with no violations. In practice, there are usually some violations (i.e., observations that are within the pathway or on the wrong side of the pathway) The problem is then referred to as *soft margin classification* and there is a trade-off between the width of the pathway and the severity of violations. As the pathway becomes wider the violations become greater.

Continuing with the notation that x_{ij} is the value of the *j*th feature for the *i*th observation, we define:

$$z_i = \max\left(b + 1 - \sum_{j=1}^{m} w_j x_{ij}, 0\right) \quad \text{if positive outcome}$$

$$z_i = \max\left(\sum_{j=1}^{m} w_j x_{ij} - (b - 1), 0\right) \quad \text{if negative outcome}$$

The variable z_i is a measure of the extent to which observation i violates the hard margin conditions at the end of the previous section.

The machine learning algorithm involves a hyperparameter, C, which defines the trade-off between the width of the pathway and violations. The objective function to be minimized is

$$C\sum_{i=1}^{n} z_i + \sum_{j=1}^{m} w_j^2$$

where n is the number of observations. Like the hard margin case, this can be set up as a quadratic programming problem.

To illustrate the soft margin classification problem, we change the example in Table 5.1 so that (a) the adjusted credit score for the second loan is 140 rather than 30 and (b) the income for the eighth loan is 60 rather than 95. The new data is in Table 5.2.

Table 5.2 Loan data set for illustrating soft margin classification

Credit score	*Adjusted credit score*	*Income ('000s)*	*Default =0; good loan=1*
660	40	30	0
650	140	55	0
650	30	63	0
700	80	35	0
720	100	28	0
650	30	140	1
650	30	100	1
710	90	60	1
740	120	64	1
770	150	63	1

Figure 5.4 shows the new data together with the optimal path when C = 0.001. It shows that one observation (the first one) is to the south-west of the pathway and another observation (the sixth one) is to the north-east of the pathway. Three observations (the fifth, seventh and tenth) are support vectors on the pathway edges and the remaining five observations are within the pathway.

It is the center solid line in Figure 5.4 that defines the way loans are classified. Only one observation (the second one) is misclassified by the SVM algorithm.

The results for different values of C are shown in Table 5.3. When C = 0.001, as we have just seen, one loan (10% of the total) is misclassified. In this case, w_1 = 0.0397, w_2 = 0.0122, and b = 3.33. When C is decreased to 0.0003 so that violations are less costly, two loans (20% of the total) are misclassified. When it is decreased again to 0.0002, three loans (30% of the total) are misclassified.

This baby example shows that SVM is a way of generating a number of plausible boundaries for a data set. As with other machine learning algorithms, it is necessary to use a training set, validation set and a test set to determine the best model and its errors.

Figure 5.4 Optimal pathway for the data in Table 5.2 when C=0.001 (See Excel or Python file)

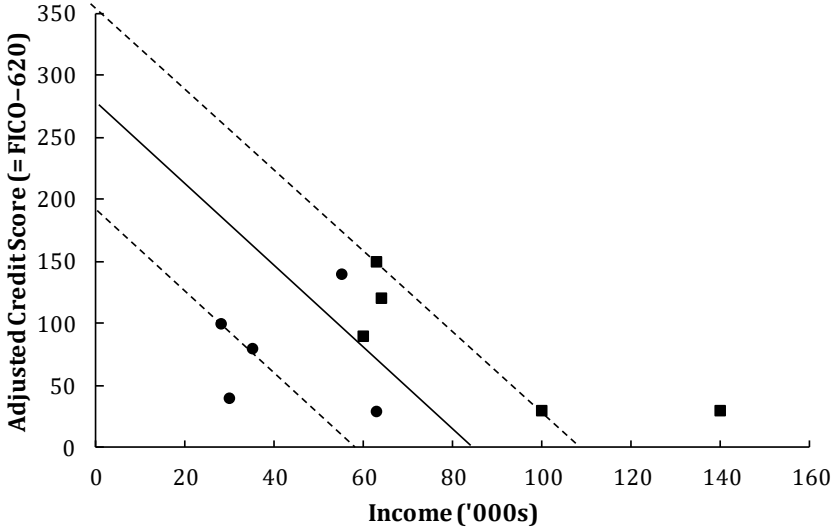


Table 5.3 Results of applying SVM to the data in Table 5.2. See Python results. Excel results are not as accurate.

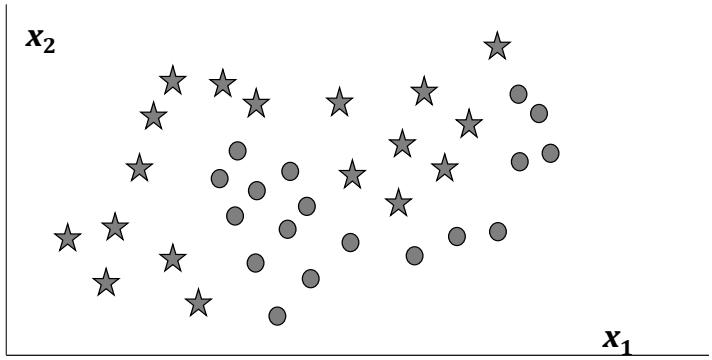
C	w_1	w_2	b	*Loans mis-classified*	*Width of pathway*
0.01	0.054	0.022	5.05	10%	34.4
0.001	0.040	0.012	3.33	10%	48.2
0.0005	0.026	0.010	2.46	10%	70.6
0.0003	0.019	0.006	1.79	20%	102.2
0.0002	0.018	0.003	1.69	30%	106.6

5.3 Non-linear Separation

So far, we have assumed that the pathway separating observations into two classes is a linear function of the feature values. We now investigate how this assumption can be relaxed.

Figure 5.5 provides an example of a situation where there are only two features, x_1 and x_2. It appears that a non-linear boundary would work better than a linear boundary. The general approach to finding a non-linear boundary is to transform the features so that the linear model presented so far in this chapter can be used.

Figure 5.5 Example of data where a non-linear separation would be appropriate (Circles and stars represent observations in different classes.)



As a simple example of this approach, suppose that we introduce A, the age (yrs.) of the borrower, as a feature for classifying loans. We sup-

pose that for $A < 23$ and $A > 63$, the impact of age is negative (so that it is more likely that the loan will default) while for $23 \leq A \leq 63$ the impact of age is positive. A linear pathway will not handle the age variable well.

One idea is to replace A by a new variable:

$$Q = (43 - A)^2$$

If the dependence of creditworthiness on age is closer to quadratic than linear the transformed variable Q will work better as a feature than the original variable A.

We can extend this idea by creating several features that are powers of the existing features. For example, for loan classification we could create features that are the square of income, the cube of income, the fourth power of income and so on.

Another way of transforming features to achieve linearity is by using what is known as a *Gaussian radial bias function* (RBF). Suppose that there are m features. We choose a number of landmarks in m-dimensional space. These could (but do not need to) correspond to observations on the features. For each landmark, we define a new feature that captures the distance of an observation from the landmark. Suppose that the values of the features at a landmark are $\ell_1, \ell_2, \ldots, \ell_m$ and that the values of the features for an observation are $x_1, x_2, \ldots, x_m$. The distance of the observation from the landmark is

$$D = \sqrt{\sum_{j=1}^{m} (x_j - \ell_j)^2}$$

and the value of the new RBF feature that is created for the observation is

$$\exp(-\gamma D^2)$$

The parameter γ determines how the value of the RBF feature for an observation declines as its distance from the landmark increases. (As γ increases, the decline becomes more rapid.)

Using many landmarks or introducing powers of the features as new features will usually lead to linear separation in a situation such as that

in Figure 5.5. The downside is that the number of features increases, and the model becomes more complex.[2]

5.4 Predicting a Continuous Variable

SVM can be used to predict the value of a continuous variable. It is then referred to as *SVM regression*. Instead of trying to fit the largest possible pathway between two classes while limiting violations, we try to find a pathway with a pre-specified width that contains as many of the observations as possible.

Consider a simple situation where a target, y, is being estimated from only one feature, x. The value of the target is on the vertical axis and the value of the feature is on the horizontal axis. The vertical half-width of the pathway is specified by a hyperparameter, e. We assume that the center of the path is

$$y = wx + b$$

The situation is illustrated in Figure 5.6. If an observation, i, lies within the pathway, there is considered to be no error. If it lies outside the pathway, the error, z_i, is calculated as the vertical distance from the edge of the pathway. We could choose the pathway to minimize

$$C\sum_{i=1}^{n} z_i + w^2$$

where C is a hyperparameter. The w^2 term is a form of regularization. It is not necessary when there is only one feature but becomes more relevant as the number of features is increases.

When there are m features with values x_j ($1 \le j \le m$), the pathway is formed by two hyperplanes separated vertically by $2e$.[3] The equations of the hyperplanes are

---

[2] The complexity can be reduced by what is known as the kernel trick. See, for example, J. H. Manton and P.-O. Amblard, "A primer on reproducing Hilbert spaces," https://arxiv.org/pdf/1408.0952v2.pdf

[3] For example, when there are two features there is a three-dimensional relationship between the target and the features and the pathway consists of two parallel planes. When there are m (>2) features the relationship between the target and the features is in m+1 dimensional space.

$$y = \sum_{j=1}^{m} w_j x_j + b + e$$

and

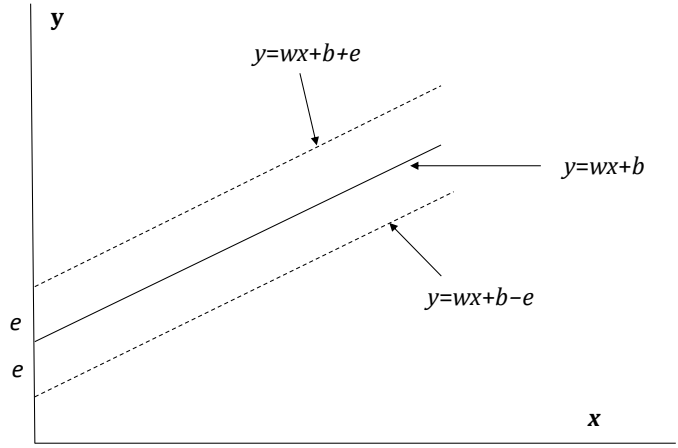
$$y = \sum_{j=1}^{m} w_j x_j + b - e$$

The objective function to be minimized is

$$C \sum_{i=1}^{n} z_i + \sum_{j=1}^{m} w_j^2$$

Here the regularization aspect of the second term becomes clear. Its aim is to prevent large positive and negative weights.[4]

Figure 5.6 Pathway for SVM regression



Consider the task of estimating the price of a house from Living Area (square feet). We are searching for a pathway such as that shown in Figure 5.6. Figure 5.7 shows the result for the training set when e = 50,000 and C = 0.01 while Figure 5.8 shows results for the training set

[4] Its effect is similar to that of the extra term included in a Ridge regression (see Chapter 3).

when e = 100,000 and C = 0.1. The line used for predicting house prices is the solid line in the figures.

Figure 5.7 Results for training set when house prices are predicted from living area with e = 50,000 and C = 0.01 (See SVM regression Excel file for calculations)

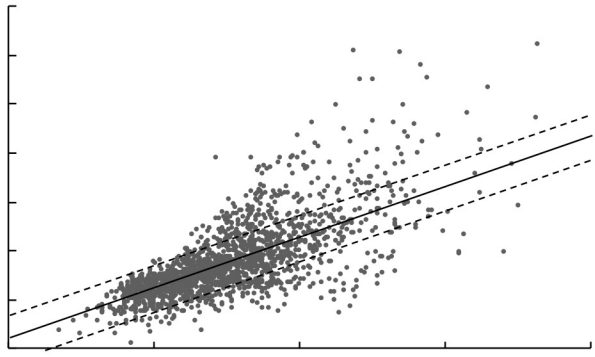
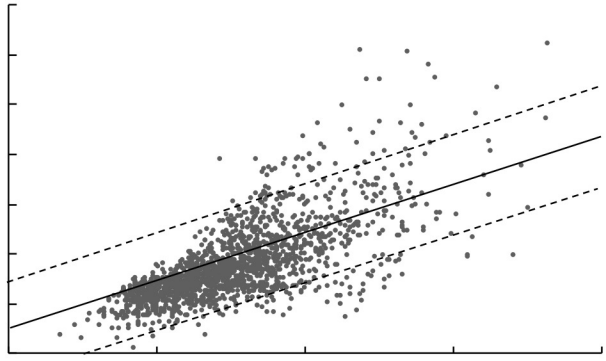


Figure 5.8 Results for training set when house prices are predicted from living area with e = 100,000 and C = 0.1 (See SVM regression Excel file for calculations)



The two SVM regression lines have slightly different biases and weights. In Figure 5.7 the b = 21,488 and w = 104.54, whereas in Figure 5.8 the b = 46,072 and w = 99.36. (This means that the slope of the regression line in Figure 5.7 is slightly greater than in Figure 5.8.)

Which is the better model? As usual this must be determined out-of-sample. Table 5.4 shows that the standard deviation of the prediction errors for the validation set is slightly lower for the model in Figure 5.7. In this case, plain vanilla linear regression has better prediction errors than both models. This is not too surprising as the regularization inherent in the SVM objective function does not improve the model when there is only one feature. (But see Exercise 5.11 for an extension of the example).

Table 5.4 Validation set results for the two SVM regression models considered

	Standard deviation of prediction error ($)
Model in Figure 5.7	58,413
Model in Figure 5.8	58,824
Linear regression	57,962

To summarize, the SVM approach is different from a simple linear regression because

- The relationship between the target and the features is represented by a pathway rather than a single line.
- The prediction error is counted as zero when an observation lies within the pathway.
- Errors for observations outside the pathway are calculated as the difference between the target value and the closest point in the pathway that is consistent with the feature values.
- There is some regularization built into the objective function as explained above.

We can extend SVM regression to allow the pathway to be non-linear by creating new features as functions of the original features in a similar way to that discussed in Section 5.3 for non-linear classification.

Summary

SVM seeks to classify observations by deriving a pathway between observations in the training set. The center of the pathway is the boundary used to assign new observations to a class. In the simplest situation, the pathway is linear function of the features and all observations are correctly classified. This is referred to as a hard margin classification. However, perfect separation is usually not possible and alternative boundaries can be developed by considering alternative trade-offs between the width of the pathway and the extent of violations.

By working with functions of the feature values rather than the feature values themselves the pathway for separating the observations into two classes can be made non-linear. We have discussed the possibility of creating new features that are squares, cubes, fourth powers, etc. of the feature values. Alternatively, landmarks can be created in the feature space with new features that are functions of the distances of an observation from the landmarks.

SVM regression uses the ideas underlying SVM classification to predict the value of a continuous variable. A pathway through the observations for predicting the target is created. If the value of the target for an observation is inside the pathway there is assumed to be no prediction error. If it is outside the pathway the prediction error is the difference between the target value and what the target value would be if it were just inside the pathway. The width of the pathway (measured in the direction of the target value) is specified by the user. There is a trade-off between the average pathway prediction error and the amount of regularization.

SHORT CONCEPT QUESTIONS

5.1 What is the objective in SVM classification?

5.2 What is the difference between hard and soft margin classification?

5.3 What are the equations for the upper and lower edges of the pathway in linear classification with m features in terms of the weights w_j and the feature values x_j?

5.4 What happens to the width of the path as the cost assigned to violations increases?

5.5 How is the extent of the violation measured in soft margin linear classification?

5.6 How is the methodology for linear classification extended to non-linear classification?
5.7 What is a landmark and what is a Gaussian radial bias function (RBF)?
5.8 Explain the objective in SVM regression.
5.9 What are the main differences between SVM regression and simple linear regression?

EXERCISES

5.10 Produce a table similar to Table 5.3 for the situation where the data in Table 5.1 is changed so that the third loan is good and the eighth loan defaults.
5.11 Use Sklearn.svm.LinearSVR to extend the SVM regression analysis in Section 5.4 for the Iowa house price example so that other features are considered.

Chapter 6

Supervised Learning: Neural Networks

Artificial Neural Networks (ANNs) are powerful machine learning algorithms that have found many applications in business and elsewhere. The algorithms learn the relationships between targets and features using a network of functions. Any continuous non-linear relationship can be approximated to arbitrary accuracy using an ANN.

In this chapter, we first explain what an ANN is. We then move on to provide an application and explain extensions of the basic idea to what are known as autoencoders, convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

6.1 Single Layer ANNs

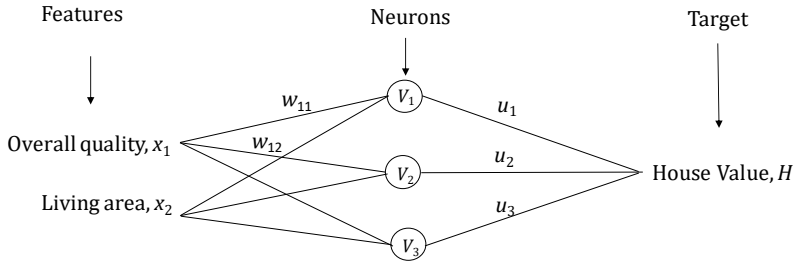
Consider the problem we considered in Section 4.5 of predicting the value of a house from just two features:

- Overall quality
- Living area (sq. ft.)

A simple ANN for doing this is shown in Figure 6.1. It has three layers: the *input layer* consisting of the two features, an *output layer* consisting of the house value and a *hidden layer* consisting of three neurons. The parameters and variables in Figure 6.1 are as follows:

- w_{jk} is a model parameter. It is the weight linking the *j*th feature to the *k*th neuron. As there are two features and three neurons, there are a total of 6 of these weights in Figure 6.1. (Only two are marked on the figure.)
- u_k is a model parameter. It is the weight linking the *k*th neuron to the target.
- V_k is the value at neuron *k*. It is calculated by the model.

Figure 6.1 A simple single-hidden-layer ANN for predicting the value of a house



Functions are specified relating the V_k to the x_j and the value of the house, H, to the V_k. The key point is that the ANN does not relate H directly to the x_j. Instead it relates H to the V_k and the V_k to the x_j. The functions that define these relationships are referred to as *activation functions*. A popular activation function is the sigmoid function which was introduced in Chapter 3 in connection with logistic regression (see Figure 3.10).[1] This is the function

$$f(y) = \frac{1}{1 + e^{-y}}$$

[1] Examples of other popular activation functions are:
(a) the hyperbolic tangent function $\tanh(y) = (e^{2y} - 1)/(e^{2y} + 1)$ which gives values between −1 and +1; and
the Relu function max(y, 0).

For all values of the argument *y,* the function lies between zero and one. We set

$$V_k = f(a_k + w_{1k}x_1 + w_{2k}x_2)$$

where the *a*'s are biases and the *w*'s are weights. This means that

$$V_1 = \frac{1}{1 + \exp(-a_1 - w_{11}x_1 - w_{21}x_2)}$$

$$V_2 = \frac{1}{1 + \exp(-a_2 - w_{12}x_1 - w_{22}x_2)}$$

$$V_3 = \frac{1}{1 + \exp(-a_3 - w_{13}x_1 - w_{23}x_2)}$$

To relate a numerical target such as H to the V_k, we typically use a linear activation function so that

$$H = c + u_1V_1 + u_2V_2 + u_3V_3$$

where c is a bias and the u's are weights.

The model in Figure 6.1 has six weights w_{jk}, three biases $a_{k,}$ three weights u_k, and one additional bias c for a total of 13 parameters. The objective is to choose the parameters so that the predictions given by the network are as close as possible to the target values for the training set. Typically, this is done by minimizing an objective function such as mean squared error (mse) or mean absolute error (mae) across all observations. This objective function is referred to as a *cost function.*[2]

For another simple example of an ANN consider the problem in Chapter 3 of classifying loans into "good" and "default" categories using four features:

- Credit score
- Income ($'000s)
- Debt to income ratio
- Home ownership (1 = owns; 0 = rents)

[2] The cost function terminology is used throughout machine learning when a numerical value is being predicted. For example, the mse in a linear regression is referred to as a cost function.

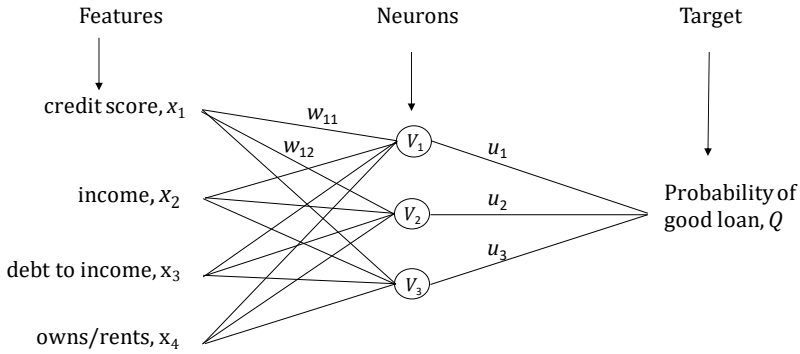
An ANN with one hidden layer is shown in Figure 6.2. This works in the same way as the one in Figure 6.1 except that we do not use a linear activation function to relate the target to the V_k. We require the target to be between zero and one because it is a probability and so it is natural to use another sigmoid function to calculate it from the V_k. The probability of a good loan is given by

$$Q = \frac{1}{1 + \exp(-c - u_1 V_1 - u_2 V_2 - u_3 V_3)}$$

In Figure 6.2, the number of parameters is 19.

The objective function for a neural network, such as that in Figure 6.2 where a probability is being predicted, can be the one based on maximum likelihood that we introduced in connection with logistic regression in equation (3.8).

Figure 6.2 A simple single-hidden-layer ANN for classifying loans



The ANNs in Figures 6.1 and 6.2 have a single hidden layer with three neurons. In practice, single-hidden-layer ANNs usually have many more than three neurons. There is a result known as the *universal approximation theorem* which states that any continuous function can be approximated to arbitrary accuracy with an ANN that has a single hidden layer.[3] However, this may require a very large number of neurons and it can be more computationally efficient to use multiple hidden layers.

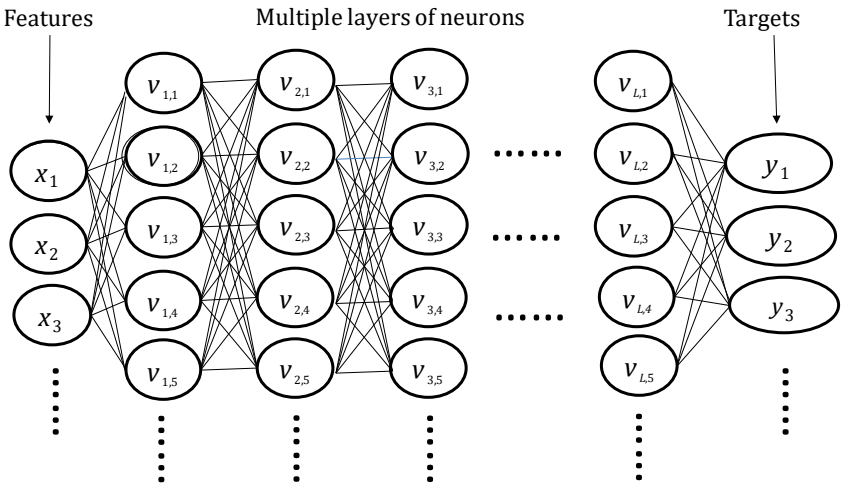
[3] See K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, 1991, 4, 251–257.

6.2 Multi-layer ANNs

Figure 6.3 shows the general configuration of a multi-layer ANN. In each of Figure 6.1 and 6.2 there is one set of intermediary variables (i.e., one hidden layer) between the features and the target. In Figure 6.3, there are a total of L hidden layers and therefore $L + 1$ sets of biases and weights. We have labeled the feature values, x_j $(1 \le j \le m)$ similarly to before. We assume K neurons per layer. The values at the neurons comprising the lth layer are labeled V_{lk} $(1 \le k \le K; 1 \le l \le L)$.

As indicated in Figure 6.3 there can be multiple targets in the output layer. The objective function can then be set equal to the sum of the (possibly weighted) objective functions that would be used for each target. (For targets that are probabilities, the maximum likelihood objective function in equation (3.8) can have its sign changed so that it is an expression to be minimized rather than maximized.)

Figure 6.3 A multi-layer neural network



There are weights associated with each of the lines in Figure 6.3. Activation functions are used to:

1. Relate values at the neurons of the first hidden layer to the feature values, i.e., to relate the V_{1k} to the x_j
2. Relate the values at neurons in hidden layer l+1 to the values at neurons in hidden layer l $(1 \le l \le L-1)$

3. Relate the target values to the values in the final hidden layer (i.e., to relate the y's to the V_{Lk})

The sigmoid function, used in the way described in the previous section, is a popular choice for the activation function for 1 and 2. In the case of 3, as explained in the previous section, linear activation functions are usually used for numerical targets while the sigmoid function can be used for classification.

The number of hidden layers and number of neurons per hidden layer necessary to solve a particular problem is usually found by trial and error. Typically, layers and neurons are increased until it is found that further increases produce little increase in accuracy.

A neural network can easily give rise to a very large number of model parameters. If there are F features, H hidden layers, M neurons in each hidden layer, and T targets there are

$$(F+1)M + M(M+1)(H-1) + (M+1)T$$

parameters in total. For example, in a four-feature model with three hidden layers, 80 neurons per layer and one target there are 13,441 parameters. This naturally leads to over-fitting concerns, as we discuss later.

6.3 Gradient Descent Algorithm

When neural networks are used, the minimization of the objective function is accomplished using the gradient descent algorithm. We briefly outlined how this works in Chapter 3. First, an initial set of parameter values is chosen. An iterative procedure is then carried out to gradually improve the objective function by changing these parameters.

To illustrate the gradient descent algorithm, we take a simple example. Consider again the data introduced in Table 1.1 of Chapter 1 for salary as a function of age. This is reproduced in Table 6.1. We assume a very simple (and not particularly good) linear model

$$y = bx + \varepsilon$$

where y is salary, x is age, and ε is the error. There is only one parameter b. The mean squared error, E, is given by

$$E = \frac{1}{10}\sum_{i=1}^{10}(y_i - bx_i)^2 \tag{6.1}$$

where x_i and y_i are age and salary for the *i*th observation.

Table 6.1 Salaries for a random sample of ten people working in a particular profession in a certain area

Age (years)	*Salary ($'000)*
25	135
55	260
27	105
35	220
60	240
65	265
45	270
40	300
50	265
30	105

The value of the parameter b that minimizes E can of course be calculated analytically, as we explained in Chapter 3. Here we show how the gradient descent algorithm can be used. Figure 6.4 shows the mean squared error, E, as a function of b. The aim of the algorithm is to find the value of b at the bottom of the valley in Figure 6.4.

We might arbitrarily set $b = 1$ initially. Using calculus, it can be shown that the gradient of E with respect to b is[4]

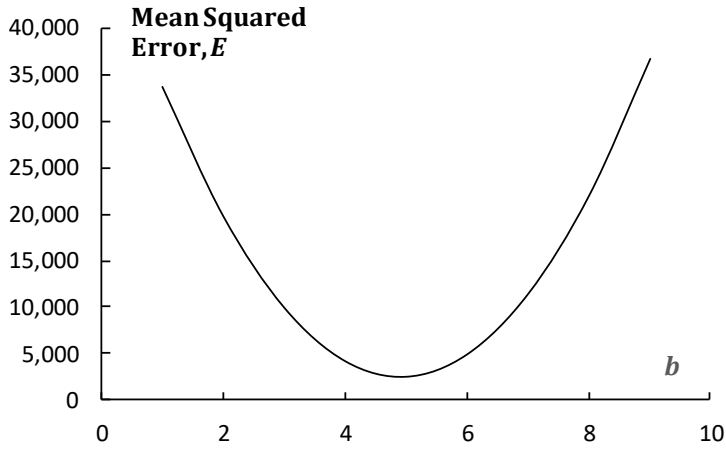
$$-\frac{1}{5}\sum_{i=1}^{10} x_i(y_i - bx_i) \tag{6.2}$$

Substituting $b = 1$ and using the values of x_i and y_i in Table 6.1, this formula gives −15,986.2. This means that when b increases by a small amount e, E increases by −15,986.2 times e.

[4] Without knowing any calculus, we could calculate this gradient by

- substituting $b = 1.01$ into equation (6.1) to get E^+,
- substituting $b = 0.99$ into equation (6.1) to get E^-
- calculating the gradient as $(E^+ - E^-)/(2 \times 0.01)$

Figure 6.4 Mean squared error as a function of the value of the parameter b



Once we have calculated the gradient at $b = 1$, we take a step down the valley. The size of the step is referred to as the *learning rate*. The new value of b is calculated from the old value of b as

$$b^{\text{new}} = b^{\text{old}} - \text{learning rate} \times \text{gradient} \qquad (6.3)$$

In our example, we choose a learning rate equal to 0.0002 so that we change $b = 1$ to

$$b = 1 - 0.0002 \times (-15{,}986.2) = 4.1972$$

We then calculate the gradient from equation (6.2) when $b = 4.1972$. This turns out to be −2,906.9. Using equation (6.3) we calculate a new value of b on the second iteration to be

$$b = 4.1972 - 0.0002 \times (-2906.9) = 4.7786$$

We continue in this way, improving the value of b at each step. As indicated in Table 6.2, the value of b quickly converges to 4.9078 which (as a simple linear regression verifies) is the value that minimizes *E.*

Table 6.2 Value of *b* in successive iterations when learning rate is 0.0002

Iteration	*Value of b*	*Gradient*	*Change in b*
0	1.0000	−15,986.20	+3.1972
1	4.1972	−2,906.93	+0.5814
2	4.7786	−528.60	+0.1057
3	4.8843	−96.12	+0.0192
4	4.9036	−17.48	+0.0035
5	4.9071	−3.18	+0.0006
6	4.9077	−0.58	+0.0001
7	4.9078	−0.11	+0.0000
8	4.9078	−0.02	+0.0000
9	4.9078	0.00	+0.0000

In Table 6.2, we chose a learning rate of 0.0002 which proves to work well. A learning rate that is too low will lead to very slow convergence. A learning rate that is too high will lead to no convergence at all. Tables 6.3 and 6.4 illustrate this by showing what happens when learning rates of 0.00001 and 0.0005 are used. As we explain later, methods for optimizing learning rates have been developed.

Table 6.3 Value of *b* in successive iterations when learning rate is 0.00001

Iteration	*Value of b*	*Gradient*	*Change in b*
0	1.0000	−15,986.20	+0.1599
1	1.1599	−15,332.24	+0.1533
2	1.3132	−14,705.03	+0.1471
3	1.4602	−14,103.47	+0.1410
4	1.6013	−13,526.53	+0.1353
5	1.7365	−12,973.18	+0.1297
6	1.8663	−12,442.48	+0.1244
7	1.9907	−11,933.48	+0.1193
8	2.1100	−11,445.31	+0.1145
9	2.2245	−10,977.10	+0.1098

Table 6.4 Value of b in successive iterations when learning rate is 0.0005

Iteration	*Value of b*	*Gradient*	*Change in b*
0	1.0000	−15,986.20	+7.9931
1	8.9931	16,711.97	−8.3560
2	0.6371	−17,470.70	+8.7353
3	9.3725	18,263.87	−9.1319
4	0.2405	−19,093.05	+9.5465
5	9.7871	19,959.87	−9.9799
6	−0.1929	−20,866.05	+10.4330
7	10.2401	21,813.37	−10.9067
8	−0.6665	−22,803.69	+11.4018
9	10.7353	23,838.98	−11.9195

Multiple Parameters

When many parameters have to be estimated, all the parameter values change on each iteration. For the gradient descent algorithm to work efficiently, the feature values should be scaled as described in Section 2.1. The change in the value of a parameter, θ, equals

$$-\text{Learning rate} \times \text{gradient}$$

as before. In this case, the gradient used is the rate of change of the value the objective function with respect to θ. To use the language of calculus, the gradient is the partial derivative of E with respect to θ.

Suppose that there are two parameters and at a particular point in the gradient descent algorithm, the gradient in the direction of one parameter is ten times the gradient in the direction of the other parameter. If the same learning rate is used for both parameters, the change made to the first parameter will be ten times as great as that made to the second parameter.

When there are a large number of parameters, determining the gradient applicable to each one is liable to be prohibitively time consuming. Luckily a shortcut has been developed.[5] This is known as *backpropagation* and involves working back from the end of the network to the be-

[5] See D. Rumelhart, G. Hinton, and R. Williams, "Learning internal representations by error propagation," *Nature*, 1986, 323, 533–536.

ginning calculating the required partial derivatives. A technical note explaining this is on the author's website:

www-2.rotman.utoronto.ca/~hull

It is also worth noting that the partial derivatives of the target with respect to each of the features can be calculated from the neural network by working forward through the network.[6]

6.4 Variations on the Basic Method

As already indicated, neural networks learn much faster when the features being input are scaled (see Section 2.1). Also, some regularization is usually desirable. Similarly to linear regression, L1 regularization involves adding a constant times the sum of the absolute values of all the weights to the target; L2 regularization involves adding a constant times the sum of the squares of all weights to the target. Analogously to linear regression (see Chapter 3), L1 regularization zeroes out some weights whereas L2 regularization reduces the average magnitude of the weights.

The gradient descent algorithm can lead to a local minimum. Consider for example the situation in Figure 6.5. If we start at point A, we could reach the local minimum at point B when the better (global) minimum is at point C. To speed up the learning process and attempt to avoid local minima several variations on the basic gradient descent algorithm have been developed. For example,

- *Mini-batch stochastic gradient descent.* This randomly splits the training data set into small subsets known as *mini-batches*. Instead of using the whole training data to calculate gradients, it updates model parameters based on the gradients calculated from a single mini-batch with each of the mini-batches being used in turn. Because the algorithm estimates the gradient using a small sample of the training data, it is faster. An *epoch* is a set of iterations that make one use of the whole training set.
- *Gradient descent with momentum.* This calculates a gradient as an exponentially decaying moving average of past gradients. This

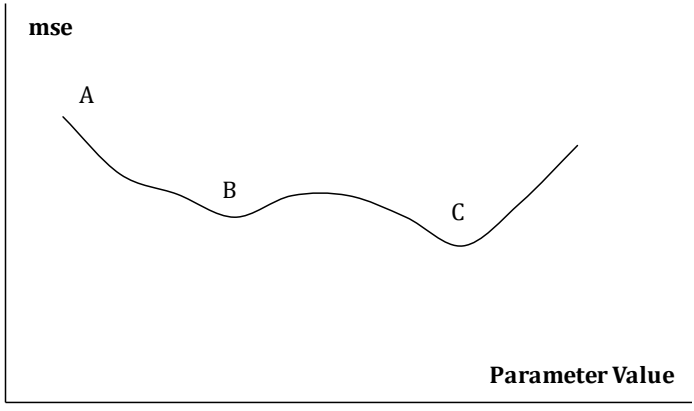
[6] This assumes that the activation functions are differentiable everywhere. The sigmoid and hyperbolic tangent function do have this property. The relu function does not. Both backpropagation and the calculation of partial derivative involve an application of the chain rule for differentiation.

approach helps to speed up learning in any direction that has a consistent gradient.

- *Gradient descent with adaptive learning rates.* As Tables 6.2, 6.3, and 6.4 illustrate it is important to choose a good learning rate. A learning rate that is too small will result in many epochs being required to reach a reasonable result. A learning rate that is too high may lead to oscillations and a poor result. Different model parameters may benefit from different learning rates at different stages of training. A popular adaptive learning rate algorithm is *Adam* which stands for Adaptive moment estimation. It uses both momentum and an exponentially decaying average of past squared gradients.
- *Learning rate decay.* In addition, using adaptive learning rates it generally makes sense to reduce the learning rate as the algorithm progresses. This is known as learning rate decay. (The shape of the curve in Figure 6.4 indicates that it would work well for that example.) To avoid local minima periodic increases in the learning rate are sometimes made.
- *Gradient descent with dropouts.* Training can be faster if some nodes, chosen at random from each hidden layer, are removed from the network on each iteration. The number of iterations necessary for convergence is increased but this is more than offset by a reduction in the run time for each iteration

.

Figure 6.5 Situation where there is a local minimum at B



6.5 The Stopping Rule

It might be thought that the algorithm should continue until the values of the parameters can be improved no more, i.e., until we have reached the bottom of the valley that defines the objective function, E, in terms of the parameters. In the simple example we considered earlier, the algorithm does this. Indeed, the optimal value of b was found in Table 6.2 (to four places of decimals) after only seven iterations.

In practice, as we have pointed out, there can be tens of thousands of parameters in a neural network. Continuing to change the parameters until the error, E, is minimized for the training set, even if that were possible, would result in a very complex model and over-fitting. As we pointed out in Chapter 1, a good practice is to continue fitting the machine learning model to the data, making it more complex, until the results for the validation set start to get worse.

When implementing a neural network, we therefore calculate the cost function for the both the training set and the validation set after each epoch of training. The usual practice is to stop training when the cost function for the validation set starts to increase. We then choose to use the model that gives the lowest cost function for the validation set. As training progresses, neural network software must therefore calculate the cost function for the validation set after each epoch and remember all the weights and biases associated with the model that gives the lowest cost function.

6.6 The Black–Scholes–Merton Formula

The Black–Scholes–Merton (or Black–Scholes) formula is one of the most famous results in finance. It gives the value of a call option on an asset as

$$Se^{-qT}N(d_1) - Ke^{-rT}N(d_2) \tag{6.4}$$

where

$$d_1 = \frac{\ln(S/K) + (r - q + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\ln(S/K) + (r - q - \sigma^2/2)T}{\sigma\sqrt{T}}$$

The inputs to this formula are as follows. S is the stock price, K is the strike price, r is the risk-free rate, q is the dividend yield (i.e., income as a percent of the price of the stock), σ is the stock price volatility, and T is the option's time to maturity.

We will use the formula to provide an application of neural networks.[7] (See www-2.rotman.utoronto.ca/~hull) Note that it is not necessary to understand how call options work in order appreciate this application of neural networks. However, interested readers will find that Chapter 10 provides a discussion of this as well as some further applications of machine learning to derivative markets.

We assume $q = 0$ and create a data set of 10,000 observations by randomly sampling from uniform distributions for the other five inputs to the Black–Scholes–Merton formula.[8] The lower and upper bounds of the uniform distributions are as indicated in Table 6.5. For each set of parameters sampled, we calculate the Black–Scholes–Merton price using equation (6.4). To make the illustration more interesting, we then add a random error to each observation. The random error is normally distributed with a mean of zero and standard deviation of 0.15. The observations were split as follows: 60,000 to the training set, 20,000 to the validation set, and 20,000 to the test set. *Z*-score scaling, based on the mean and standard deviation of the observations in the training set, was used for all the features.

Table 6.5 Upper and lower bounds used for Black-Scholes parameters to create the data set

	Lower bound	*Upper bound*
Stock price, S	40	60
Strike price, K	$0.5S$	$1.5S$
Risk free rate, r	0	5%
Volatility, σ	10%	40%
Time to maturity, T	3 months	2 years

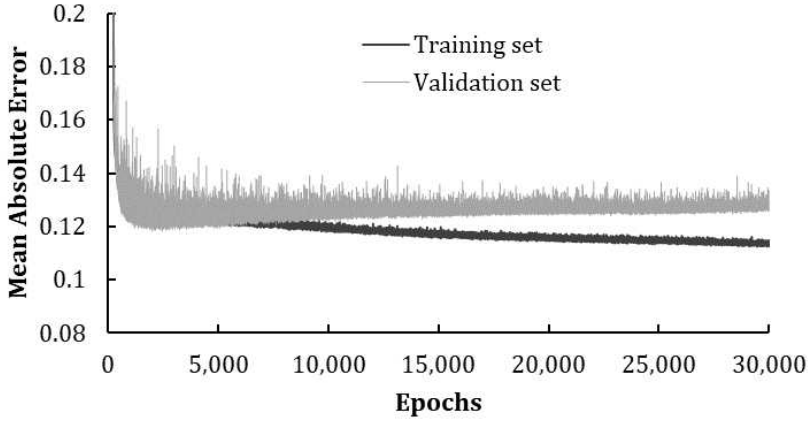
[7] Black–Scholes–Merton model was used to illustrate neural networks many years ago by J. M. Hutchinson, A. W. Lo, and T. Poggio, "A Nonparametric Approach to Pricing and Hedging Derivative Securities Via Learning Networks," *Journal of Finance*, (July 1994), 49(3): 851–889. A more recent similar implementation is R. Culkin and S. R. Das, "Machine Learning in Finance: The Case of Deep Learning for Option Pricing," *Journal of Investment Management* (2017) 15 (4): 92–100.

[8] The $q = 0$ assumption means that we are using the formula corresponding to the original Black–Scholes result.

We use mean absolute error (mae) as the cost function. The neural network has three hidden layers and 20 neurons per layer for a total of about 1,000 parameters. Similarly to Figure 6.1, the sigmoid function is used as the activation function, except for the calculation of the option price from the values in the final hidden layer where a linear activation function is used. Learning rates are determined using Adam.

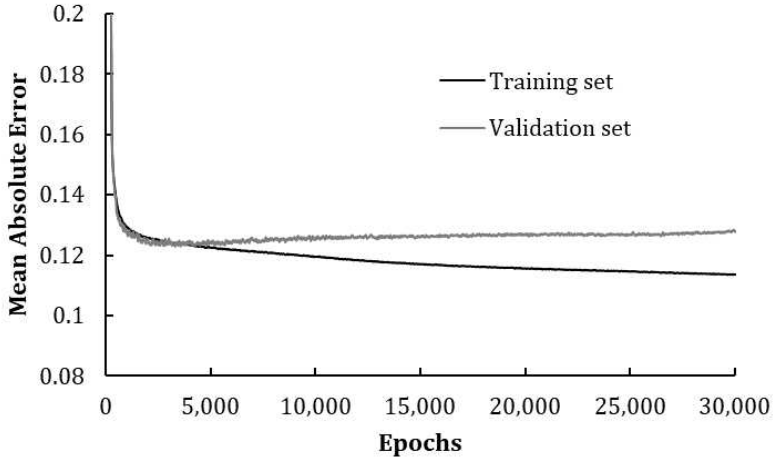
Figure 6.6 shows the mean absolute error for the training set and the validation set as the number of epochs is increased. The results for the validation are more noisy than those for the training set, but it is clear that, whereas the training set results continue to improve, the validation set results start deteriorating after a few thousand epochs. Figure 6.7 shows the same results with the errors shown for each epoch being the average error over the subsequent 50 epochs. This moving-average calculation eliminates most of the noise and shows more clearly that the validation set average errors are similar to the training set errors for the first few thousand epochs and then start to worsen.

Figure 6.6 Training set and validation set mean absolute errors as a the number of epochs of training is increased



As mentioned earlier, once an increase in the cost function for the validation set is observed, normal practice is to go back to the model that gave the best result for the validation set. In this case, the best result for the validation set was after 2,575 epochs of training.

Figure 6.7 Mean absolute errors averaged over 50 epochs



How well has the model fitted the data? The mean absolute error for the test set is 0.122. This is about what one would expect. The mean absolute value of a normally distributed variable with a mean of zero and a standard deviation of 0.15 is

$$\sqrt{\frac{2}{\pi}} \times 0.15 = 0.120$$

It is interesting to compare three values:

1. The true Black–Scholes–Merton price of an option
2. The price of the option after noise is added
3. The predicted price given by the neural network

The averages of the three prices are very close to each other. The standard deviation of the difference between 1 and 2 and between 2 and 3 was almost exactly 0.15 (which is as expected because the standard deviation of the noise added to the Black–Scholes–Merton price was 0.15). The standard deviation of the difference between 1 and 3 was about 0.04. This indicates that the neural network model gets rid of the noise reasonably well. If the size of the data set were increased, we would expect 1 and 3 to become closer.

6.7 Extensions

The example in the previous section might seem a little artificial. The Black-Scholes price can be calculated quickly and accurately, and so there is little point in using a neural network to estimate it! However, this is not true for the prices of all derivatives that are traded. Some must be valued using Monte Carlo simulation or other numerical procedures that are computationally slow. This creates problems because analysts, for a number of reasons, have to carry out scenario analyses, involving Monte Carlo simulations, to explore how the values of portfolios of derivatives can change through time. If the procedure for calculating the prices of some of the instruments in the portfolio involves Monte Carlo simulation or another slow numerical procedure, the scenario analyses can be impossibly slow.

To solve this problem, it is useful for analysts to replace slow numerical pricing procedures with neural networks.[9] Once the neural network has been constructed the valuation is very fast. All that is involved is working forward through the neural network, starting with the inputs, to get the target price. This can be several orders of magnitude faster than a slow numerical procedure.

The first stage is to create a large data set relating the derivative's value to inputs such as the price of the underlying asset, interest rates, volatilities, and so on (as we did in the example in the previous section). This is done using the standard (slow) numerical procedure and can take a long time. But it only has to be done once. The data set is then divided into a training set, a validation set, and a test set in the usual way. A neural network is trained on the training set. The validation set is used to determine when training should stop (as in our Black–Scholes example) and the test set is used to quantify the model's accuracy.

An interesting aspect of this type of application of neural networks is that the analyst does not have to collect and clean a large amount of data. The analyst generates the data needed from a model. It is possible to replicate the relationship between the output and input with very little error. One important point is that the model is only reliable for the range of values of the data in the training set. It is liable to give very poor answers if extrapolated to other data.[10]

---

[9] See R. Ferguson and A. Green, "Deeply Learning Derivatives," October 2018, ssrn 3244821.

[10] However, there have been some attempts to overcome this problem. See A. Antonov, M. Konikov, and V. Piterbarg, "Neural Networks with Asymptotic Controls," ssrn 3544698.

6.8 Autoencoders

An autoencoder is a neural network designed to reduce the dimensionality of data. Its objective is similar to that of principal components analysis (PCA), which we discussed in Section 2.7. It is designed to explain most of the variation in a data set that has *m* features with a new set of less than *m* manufactured features.

In an autoencoder's neural network, the output is the same as the input. The simplest type of autoencoder has one hidden layer and linear activation functions. The number of neurons in the hidden layer is less than the number of features. The first part of the network where values at the neurons are determined from the inputs is known as *encoding*. The second part of the network where the output is determined from the values at neurons is known as *decoding*. The neural network tries to minimize the total mean squared error (or other cost function) between the predicted output values and the actual output/input values. If the mean squared error is small, the weights used to calculate the predicted output from the hidden layer provide manufactured features that are smaller in number than the original features but contain similar information.

In Section 2.7 we showed how PCA can be used to create a small number of features describing interest rate changes. Figure 6.8 shows the design of an autoencoder to do the same thing. The aim is to determine two features that capture most of the variation in eight interest rates.

Define r_{ij} as the *j*th rate for the *i*th observation, and $\hat{r}_{ij}$ as the prediction of the *j*th rate made by the neural network for this observation. With the notation indicated on Figure 6.8[11]

$$V_{i1} = \sum_{j=1}^{8} r_{ij} w_{j1}$$

$$V_{i2} = \sum_{j=1}^{8} r_{ij} w_{j2}$$

where V_{i1} and V_{i2} are the values of V_1 and V_2 for the *i*th observation. Also

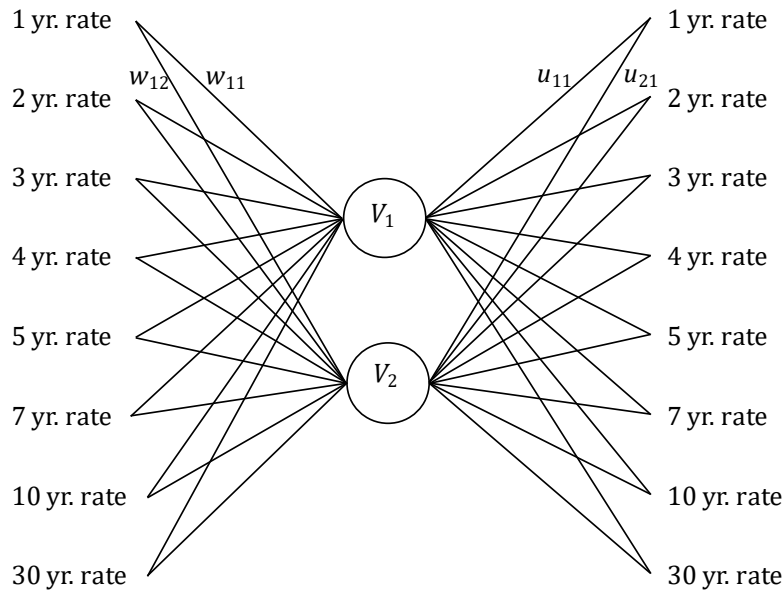
[11] To keep the example simple, we assume zero bias throughout the network. This has been suggested by R. Memisevic, K. Konda, and D. Krueger, "Zero-bias Autoencoders and the Benefits of Co-adapting Features" arXiv:1402.3337.

$$\hat{r}_{ij} = V_{i1}u_{1j} + V_{i2}u_{2j}$$

The objective is the minimize

$$\sum_{i,j}\left(r_{ij} - \hat{r}_{ij}\right)^2$$

Figure 6.8 Design of autoencoder to find two features describing interest rate changes



The first manufactured feature involves the *j*th rate moving by u_{1j} and the second one involves it moving by u_{2j}. There are many different possible sets of values for the *w*'s and the *u*'s that are equally good and give the same predictions. Results from one implementation using Excel and Solver (see www-2.rotman.utoronto.ca/~hull) are in Table 6.6.

A comparison with Table 2.9 shows that the two manufactured features are quite different from PC1 and PC2. However, they are equiva-

lent. If we transform the features in Table 6.6 so that the first manufactured feature involves a movement in the *j*th rate of

$$0.518u_{1j} + 0.805u_{2j}$$

and the second manufactured feature involves a movement in the *j*th rate of

$$2.234u_{1j} - 2.247u_{2j}$$

we do get PC1 and PC2.

Table 6.6 One possible output from the autoencoder in Figure 6.8

	First manufactured feature (the u_{1j})	*First manufactured feature (the u_{2j})*
1 yr. rate	0.0277	0.2505
2 yr. rate	0.1347	0.3248
3 yr. rate	0.2100	0.3276
4 yr. rate	0.2675	0.3147
5 yr. rate	0.3118	0.3017
7 yr. rate	0.3515	0.2633
10 yr. rate	0.3858	0.2186
30 yr rate	0.3820	0.1331

The advantage of PCA is that the features produced are uncorrelated and their relative importance is clear from the output. The advantage of autoencoders is that they can be used with non-linear activation functions. To use the PCA terminology, they allow data to be explained with non-linear factors. Autoencoders have been found to be useful in image recognition and language translation.

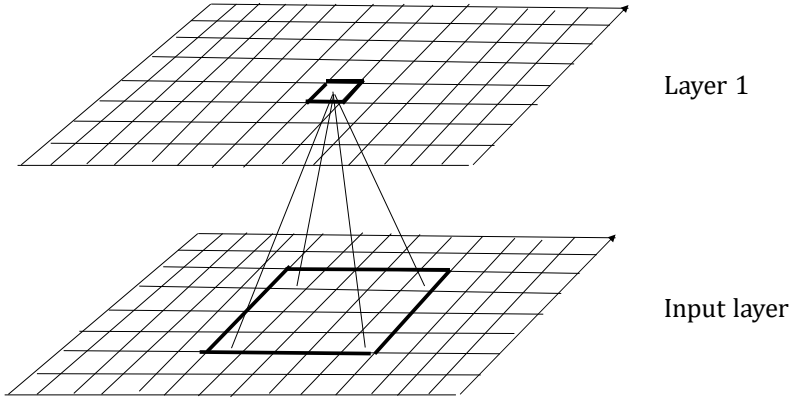
6.9 Convolutional Neural Networks

In the ANNs we have presented so far, a neuron in one layer is connected to every neuron in the previous layer. For very large networks this is infeasible. A convolutional neural network (CNN) solves this problem by connecting the neurons in one layer to only a subset of the neurons in the previous layer.

CNNs are used in image recognition, voice recognition, and natural language processing. Consider the task of processing an image through a neural network for the purposes of facial recognition. The image is divided into many small rectangles by drawing horizontal and vertical lines. The small rectangles are referred to as pixels. Each pixel has a certain colour and a number is associated with that colour. Even the simplest image is liable to have 10,000 pixels (formed from 100 horizontal and 100 vertical lines). This creates a large number of inputs to a neural network.

In Figure 6.3, the layers are columns of numbers. When a CNN is used to process images, the inputs are a rectangular array of numbered pixels. Subsequent layers are a rectangular grid of numbers. Figure 6.9 shows how the values at grid points in the first layer depend on the values at the pixels in the input layer. The rectangle in layer one denotes a single grid point (or neuron) and the bolded rectangle in the input layer shows what is termed the *receptive field*. It is all the pixels to which the grid point is related. The values at grid points in subsequent layers depend on the values at grid points in the previous layer in a similar way.[12]

Figure 6.9 Relationship of value at grid point in layer one to values at grid point in the input layer in a CNN



[12] There may be "padding" where extra observations are added at the edges to avoid successive layers becoming smaller.

The two-dimensional set of points in layer one in Figure 6.9 is referred to as a *feature map*. A complication is that each layer comprises several feature maps so that it must be represented in three dimensions. However, within a feature map, all neurons share the same set of weights and biases. As a neuron within a feature map and its associated receptive field are changed, the biases and weights stay the same. This reduces the number of parameters. It also has the advantage that the identification of an object in image recognition does not depend on the part of the input layer where it appears.

Consider a black and white image that is 100×100 so that there are 10,000 pixels. A regular neural network would lead about $10{,}000^2$ or 100 million parameters to define the neurons in Layer 1. In a CNN where the receptive field is 10×10 and there are six feature maps, this is reduced to 6×101 or 606 parameters.

6.10 Recurrent Neural Networks

In a plain vanilla ANN each observation is considered separately from every other observation. In a recurrent neural network (RNN) we preserve the temporal sequence in which observations occur because we want to allow for changes in our prediction model through time. This can be particularly important in business where relationships between variables tend to change through time.

In a plain vanilla ANN, as we have explained, the value at a neuron in layer l is calculated by applying an activation function to a linear combination of the values at the neurons of layer $l-1$. When this is done for one observation, we proceed to do the same thing for the next observation. But the algorithm has no memory. When doing calculations for an observation at time t, it does not remember anything about the calculations it did for an observation at time $t-1$.

In an RNN, the activation function at time t is applied to a total of:

- a linear combination of the values at the neurons of layer $l-1$ at time t; and
- a linear combination of the values at the neurons of layer l for the observation at the previous time $t-1$.

This gives the network memory. The values in the network at time t depend on the values at time $t-1$ which in turn depend on the values at time $t-2$, and so on.

One issue with this is that the values from several time periods earlier are liable to have very little effect because they get multiplied by rel-

atively small numbers several times. A *Long Short-Term Memory* (LSTM) approach has been developed to overcome this problem.[13] Data from the past has the potential to flow straight through to the current network. The algorithm decides which data should be used and which should be forgotten.

Summary

An artificial neural network is a way of fitting a non-linear model to data. Outputs are not related directly to inputs. There are a number of intervening hidden layers which contain neurons. The values at the neurons in the first hidden layer are related to the inputs; the values at the neurons in the second hidden layer are related to the values at the neurons in the first hidden layer; and so on. The outputs are calculated from values at the neurons in the final hidden layer.

The functions defining the relationships are referred to as activation functions. The sigmoid function, which was introduced in connection with logistic regression, is often used as an activation function to relate (a) the values at neurons in the first hidden layer to the input values and (b) values at neurons in hidden layer l to values at neurons in hidden layer $l-1$. When numerical values are being estimated the activation function relating the output to the neurons in the final hidden layer is usually linear. When data is being classified, a sigmoid function is more appropriate for this last step.

The gradient descent algorithm is used to minimize the objective function in a neural network. Calculating the minimum can be thought of as finding the bottom of a valley. The algorithm takes steps down the valley where at each stage it follows the line of steepest descent. Choosing the correct size for the step, which is referred to as the learning rate, is an important aspect of the gradient descent algorithm. A number of procedures for improving the efficiency of gradient descent algorithms have been mentioned.

It is not unusual for neural networks to involve tens of thousands of parameters. Even if it were possible to find the values of the parameters that totally minimize the objective function for the training set, this would not be desirable as it would almost certainly result in overfitting. In practice, a stopping rule is applied so that training using the

[13] See S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, 9(8): 1735–1780.

gradient descent algorithm is halted when the results for the validation set depart from those for the training set.

An autoencoder is a neural network where the output is the same as the input. It is designed to replace the input features with a smaller number of almost equivalent features. A convolutional neural network is a neural network where the neurons in one layer are related to a subset of the neurons in the previous layer rather than all of them. It is particularly useful for image recognition where the inputs are defined by the colors of tens of thousands (or even millions) of pixels. A recurrent neural network is a version of an ANN that is particularly suitable for situations where the model for forecasting the output is expected to evolve through time.

SHORT CONCEPT QUESTIONS

6.1 Explain what is meant by (a) a hidden layer and (b) a neuron, and (c) an activation function.
6.2 Explain how a sigmoid function relates the values at the neurons in one layer to the values at neurons in the previous layer.
6.3 What is the universal approximation theorem?
6.4 What activation function is suggested in the chapter for relating the target to the values in the final layer when the objective is (a) to predict a numerical variable and (b) to classify data?
6.5 What is meant by the learning rate in a gradient descent algorithm?
6.6 What problems arise if the learning rate is too high or too low?
6.7 Explain how a stopping rule is chosen when an ANN is trained.
6.8 Explain how ANNs can be used in derivatives valuation.
6.9 Explain the key difference between a CNN and a plain vanilla ANN.
6.10 Explain the key difference between an RNN and a plain vanilla ANN.

EXERCISES

6.11 How many parameters are there when an ANN has five features, two hidden layers and ten neurons per hidden layer, and one target?

6.12 Produce tables similar to Table 6.2 for the validation set in Table 1.2 of Chapter 1. Assume the simple $y = bx$ model. Experiment with different starting points and learning rates.

6.13 In the model in Figure 6.1 assume that we start the gradient descent algorithm by setting all the w_{jk} weights equal to 0, all the u_k weights equal to 100, and the biases equal to 0. What does the initial network give for the price of a house with overall quality equal to 8 and living area equal to 3000 square feet?

6.14 Use the Python code on www-2.rotman.utoronto.ca/~hull to explore how well the Black–Scholes–Merton application in Section 6.6 works as the number of observations, the number of hidden layers, and the number of neurons per layer are changed.

6.15 Use neural networks for the Iowa house price data. Try different numbers of hidden layers and neurons per layer. Compare the results with those from linear regression.

6.16 Use neural networks for the Lending Club data. Try different numbers of hidden layers and neurons per layer. Compare the results with those from logistic regression.

Chapter 7

Reinforcement Learning

So far, we have considered situations where one decision is taken in isolation from other decisions. For example, the classification of a particular loan as "accept" or "reject" in earlier chapters was assumed to be independent of other decisions made about other loans. The models we developed implicitly assumed that, whether we accept or reject a loan today, does not in any way affect whether we will accept or reject a loan that comes across our desk tomorrow.

Some situations by their nature involve a series of decisions rather than a single one. Furthermore, as the decisions are taken, the environment may be changing. It is then necessary to determine the best action bearing in mind that further decisions have to be taken later.

Reinforcement learning is the branch of machine learning that deals with this type of sequential decision making.[1] The algorithm receives rewards when outcomes are good and incurs costs (negative rewards) when they are bad. The objective of the algorithm is to maximize expected future rewards possibly with discounting.

[1] A comprehensive treatment of reinforcement learning is provided by R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, 2nd edition, 2018, The MIT Press.

In this chapter we start with a simple sequential decision-making problem where the environment does not change. This provides an illustration of the exploitation vs. exploration trade-off which is central to reinforcement learning. We then move on to consider a more complicated situation where the environment does change. Finally, we mention how reinforcement learning can be used in conjunction with neural networks and discuss applications.

7.1 The Multi-armed Bandit Problem

Imagine a gambler in a casino that offers a game where one of several different levers can be pulled. Each lever provides a random payoff. The payoff from the kth lever is a sample from a normal distribution with mean m_k and standard deviation one. The m_k are liable to be different and are not known. However, the casino guarantees that they will not change. (In this simple example, the environment does not therefore change). The game can be played many times. What strategy should the gambler follow to maximize the expected payoff over many trials?

Clearly the gambler should keep careful records so that the average payoff realized so far from each of the levers is known at all times. At each trial of the game, the gambler has to decide between two alternatives:

- Choose the lever that has given the best average payoff so far
- Try a new lever

This is known as the exploitation vs. exploration choice. The first alternative is exploitation (also known as the *greedy action*). If, after choosing each lever a few times, the gambler only followed the first strategy, she would not find the best lever unless she was lucky. Some exploration, where another lever is chosen at random, is therefore a good idea. Exploitation maximizes the immediate expected payoff, but a little exploration may improve the long-run payoff.

A strategy for the gambler is to:

- Randomly choose a lever with probability ε
- Choose the lever that has given the best average payoff so far with probability $1-\varepsilon$

for some ε ($0 < \varepsilon \leq 1$). We can implement this strategy by sampling a random number between 0 and 1. If it is less than ε, the lever is chosen randomly; otherwise, the lever with the best average payoff so far is chosen. We might choose a value of ε equal to one initially and then slowly reduce it to zero as data on the payoffs is obtained.

Suppose that the *k*th lever has been chosen $n-1$ times in the past and the total reward on the *j*th time it was chosen was R_j. The best estimate of the expected reward from the *k*th lever (which we will refer to as the old estimate) is

$$Q_k^{\text{old}} = \frac{1}{n-1}\sum_{j=1}^{n-1} R_j$$

If the *k*th lever is chosen for the *n*th time on the next trial and produces a reward R_n, we can update our estimate of the expected reward from the *k*th lever as follows:

$$Q_k^{\text{new}} = \frac{1}{n}\sum_{j=1}^{n} R_j = \frac{n-1}{n} Q_k^{\text{old}} + \frac{1}{n} R_n$$

which can be written as

$$Q_k^{\text{new}} = Q_k^{\text{old}} + \frac{1}{n}\left(R_n - Q_k^{\text{old}}\right) \tag{7.1}$$

This shows that there is a simple way of updating the expected reward. We do not have to remember every reward on every trial.

Consider a situation where there are four levers and

$$m_1 = 1.2, \quad m_2 = 1.0, \quad m_3 = 0.8, \quad m_4 = 1.4$$

The gambler would of course choose the fourth lever every time if these values were known. However, the m_k have to be inferred from the results of trials. We first suppose that ε is kept constant at 0.1. This means that there is always a 90% chance that the gambler will choose the lever that has given the best result so far and a 10% chance that a lever will be chosen at random. The results from a Monte Carlo simulation run are shown in Table 7.1 (see Excel file for calculations). We arbitrarily set the *Q*-values (i.e., average payoffs) for each lever equal to zero initially. Lever one is chosen on the first trial and gives a payoff of 1.293 (slightly

above average). The *Q*-value for the first lever therefore becomes 1.293 while those for the others stay at zero. On the second trial there is therefore a 10% chance that gambler will explore (i.e., choose a lever at random) and a 90% chance that she will choose the first lever. In fact, she chooses the first lever (but gets a particularly low payoff of 0.160). The first lever is still the best one with a *Q*-value of 0.726. The decision on the third trial is to exploit and so the first lever is chosen again. On the fourth trial, she explores (because a random number less than 0.1 is drawn) and randomly selects the second lever.

Table 7.1 Results from 5000 trials with four levers and ε = 0.1

		Lever		Lever 1 (stats)		Lever 2 (stats)		Lever 3 (stats)		Lever 4 (stats)		Ave Gain
Trial	Decision	Chosen	Payoff	Q-val	Nobs	Q-val	Nobs	Q-val	Nobs	Q-val	Nobs	per trial
				0		0		0		0		
1	Exploit	1	1.293	1.293	1	0.000	0	0.000	0	0.000	0	1.293
2	Exploit	1	0.160	0.726	2	0.000	0	0.000	0	0.000	0	0.726
3	Exploit	1	0.652	0.701	3	0.000	0	0.000	0	0.000	0	0.701
4	Explore	2	0.816	0.701	3	0.816	1	0.000	0	0.000	0	0.730
50	Exploit	1	0.113	1.220	45	-0.349	3	0.543	2	0.000	0	1.099
100	Exploit	4	2.368	1.102	72	0.420	6	0.044	3	1.373	19	1.081
500	Explore	3	1.632	1.124	85	1.070	17	0.659	11	1.366	387	1.299
1000	Exploit	4	2.753	1.132	97	0.986	32	0.675	25	1.386	846	1.331
5000	Exploit	4	1.281	1.107	206	0.858	137	0.924	130	1.382	4527	1.345

The "Nobs" column in Table 7.1 shows the number of times a lever has been chosen, and therefore the number of observations over which the average payoff is calculated. In the first 50 trials the best lever (Lever 4) is not chosen at all. However, in the first 500 trials it is chosen 387 times (i.e., on 77% of trials). In the first 5,000 trials it is chosen 4,527 times (i.e., on just over 90% of the trials). The table shows that, after what appears to be a rocky start, the algorithm finds the best lever without too much difficulty. The average gain per trial over 5,000 trials is 1.345, a little less than the 1.4 average payout per trial from the best (fourth) lever.

Tables 7.2 and 7.3 show the results of keeping ε constant at 0.01 and 0.5. There are a number of conclusions we can draw from these tables. Setting ε = 0.01 gives very slow learning. Even after 5,000 trials the first lever looks better than the fourth lever. What is more the average gain per trial over 5,000 trials is worse than when ε=0.1. When ε = 0.5 (so that there is always an equal chance of exploitation and exploration) the algorithm finds the best lever without too much difficulty, but the average gain per trial is inferior to that in Table 7.1 because there is too much exploration.

Table 7.2 Results from 5000 trials with four levers and ε = 0.01

				Lever 1 (stats)		*Lever 2 (stats)*		*Lever 3 (stats)*		*Lever 4 (stats)*		*Ave Gain*
Trial	*Decision*	*Lever Chosen*	*Payoff*	*Q-val*	*Nobs*	*Q-val*	*Nobs*	*Q-val*	*Nobs*	*Q-val*	*Nobs*	*per trial*
				0		0		0		0		
1	Exploit	1	1.458	1.458	1	0.000	0	0.000	0	0.000	0	1.458
2	Exploit	1	0.200	0.829	2	0.000	0	0.000	0	0.000	0	0.829
3	Exploit	1	2.529	1.396	3	0.000	0	0.000	0	0.000	0	1.396
4	Exploit	1	-0.851	0.834	4	0.000	0	0.000	0	0.000	0	0.834
50	Exploit	1	1.694	1.198	49	0.000	0	-0.254	1	0.000	0	1.169
100	Exploit	1	0.941	1.132	99	0.000	0	-0.254	1	0.000	0	1.118
500	Exploit	1	0.614	1.235	489	0.985	6	-0.182	2	0.837	3	1.224
1000	Exploit	1	1.623	1.256	986	0.902	7	-0.182	2	0.749	5	1.248
5000	Exploit	1	1.422	1.215	4952	1.022	18	0.270	8	1.148	22	1.213

Table 7.3 Results from 5000 trials with four levers and ε = 0.5

				Lever 1 (stats)		*Lever 2 (stats)*		*Lever 3 (stats)*		*Lever 4 (stats)*		*Ave Gain*
Trial	*Decision*	*Lever Chosen*	*Payoff*	*Q-val*	*Nobs*	*Q-val*	*Nobs*	*Q-val*	*Nobs*	*Q-val*	*Nobs*	*per trial*
				0		0		0		0		
1	Exploit	1	0.766	0.766	1	0.000	0	0.000	0	0.000	0	0.766
2	Explore	1	1.257	1.011	2	0.000	0	0.000	0	0.000	0	1.011
3	Exploit	1	-0.416	0.536	3	0.000	0	0.000	0	0.000	0	0.536
4	Explore	3	0.634	0.536	3	0.000	0	0.634	1	0.000	0	0.560
50	Explore	4	0.828	1.642	17	1.140	9	0.831	9	1.210	15	1.276
100	Explore	3	2.168	1.321	47	0.968	15	0.844	16	1.497	22	1.231
500	Explore	1	0.110	1.250	86	0.922	65	0.636	72	1.516	277	1.266
1000	Explore	4	1.815	1.332	154	1.004	129	0.621	131	1.394	586	1.233
5000	Explore	3	2.061	1.265	666	0.953	623	0.797	654	1.400	3057	1.247

As indicated earlier, the best strategy is to start with ε close to 1 and reduce it as data on the payoffs is accumulated. One approach is to set ε equal to 1 on the first trial and then multiply it by a decay factor slightly less than 1 on each subsequent trial. If the decay factor is β the probability of exploration on trial τ is $\beta^{\tau-1}$. Table 7.4 shows results for the multi-armed bandit problem we have been considering when β = 0.995. It can be seen that these results are superior to those in Table 7.1 where a constant ε equal to 0.1 is used. The algorithm quickly finds the best lever and produces a average gain per trial of 1.381 (compared with 1.345 for ε = 0.1).

Like most hyperparameters, the decay factor β must be chosen by trial and error. Reducing β to 0.99 often works well but occasionally fails to find the best lever. Increasing β to 0.999 does find the best lever, but not as quickly as β=0.995. (See Exercise 7.10.)

Table 7.4 Results from 5,000 trials with four levers when ε starts at 1 with a decay factor of 0.995

		Lever		Lever 1 (stats)		Lever 2 (stats)		Lever 3 (stats)		Lever 4 (stats)		Ave Gain
Trial	Decision	Chosen	Payoff	Q-val	Nobs	Q-val	Nobs	Q-val	Nobs	Q-val	Nobs	per trial
				0		0		0		0		
1	Explore	2	1.4034	0	0	1.403	1	0	0	0.000	0	1.403
2	Explore	1	0.796	0.796	1	1.403	1	0.000	0	0.000	0	1.100
3	Explore	2	0.499	0.796	1	0.951	2	0.000	0	0.000	0	0.900
4	Explore	1	0.407	0.601	2	0.951	2	0.000	0	0.000	0	0.776
50	Explore	3	-1.253	0.719	8	1.640	18	0.729	11	1.698	13	1.308
100	Explore	1	0.100	0.852	19	1.326	31	0.681	20	1.391	30	1.126
500	Exploit	4	-0.448	1.148	37	1.184	51	0.815	51	1.349	361	1.263
1000	Exploit	4	2.486	1.174	44	1.225	53	0.819	53	1.387	850	1.339
5000	Exploit	4	3.607	1.148	45	1.225	53	0.819	53	1.391	4849	1.381

7.2 Changing Environment

The multi-armed bandit problem provides a simple example of reinforcement learning. The environment does not change and so the Q-values are a function only of the action (i.e., the lever chosen). In a more general situation, there are a number of states and a number of possible actions. This is illustrated in Figure 7.1. The decision maker takes an action, A_0, at time zero when the state S_0 is known. This results in a reward, R_1, at time 1 and a new state, S_1, is encountered. The decision maker then takes another action at time 1 which results in a reward, R_2, at time 2 and a new state, S_2; and so on. In this more general situation, the Q-value is a function of both the state and the action taken.

Our aim is to maximize future expected rewards. A simple objective at time t is therefore to maximize the expected value of G where

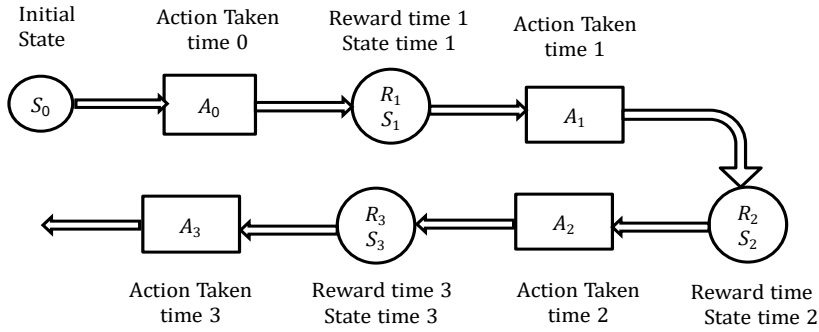
$$G = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

and T is a horizon date. In some cases, it is appropriate to maximize discounted rewards over a (possibly infinite) horizon. The reinforcement learning literature then uses the following expression for G:

$$G = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \tag{7.2}$$

where γ (< 1) is the discount factor.[2] Finance professionals are likely to interpret γ as $1/(1 + r)$ where r is the discount rate per period (possibly adjusted for risk). But this interpretation leads to the coefficient of R_{t+k} being γ^k rather than γ^{k-1}. The simplest way of reconciling equation (7.2) with the discount rates used in finance is to define the reward R_{t+k} as the cash received at time $t+k$ discounted by one period. (i.e., if the cash received at time $t + k$ is C then $R_{t+k} = \gamma C$.)

Figure 7.1 Reinforcement learning in a changing environment



Note that the states must include everything relevant to the action taken. For example, if we are developing a trading strategy and the past history of stock prices is relevant, they must be included in the "current" state.

The Q-values in the general situation reflect all future rewards, possibly discounted. Suppose that on a particular trial the total value of the future rewards from taking action A in state S is G. Suppose further that this is the nth trial where action A has been taken in state S. Analogously to equation (7.1), we might update as follows:

$$Q^{\text{new}}(S, A) = Q^{\text{old}}(S, A) + \frac{1}{n}\left[G - Q^{\text{old}}(S, A)\right]$$

In practice, in a changing environment it usually makes sense to give more weight to recent observations. We can do this by setting

[2] See, for example, R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, 2nd edition, 2018, The MIT Press, Chapter 3.

$$Q^{\text{new}}(S,A) = Q^{\text{old}}(S,A) + \alpha\left[G - Q^{\text{old}}(S,A)\right] \qquad (7.3)$$

The weight given to an observation then declines as it becomes older.

7.3 The Game of Nim

The game of Nim provides an illustration of the material in the Section 7.2. Imagine that there is a pile of matches. You and your opponent take turns to pick up one or two or three matches. The person who is forced to pick up the last match loses.

A little thought indicates that as the game nears the end you will win if you leave your opponent with five matches. Whatever your opponent picks up you will then be able to leave her with just one match on the next round. For example, if she picks up two matches you also pick up two matches; if she picks up three matches you pick up one match. How do you get to the stage where you can leave your opponent with five matches? The answer is that, if you leave her with nine matches, you will always be able to reduce the pile to five matches on your next turn. Continuing in this way, it is not difficult to see that the way to win is to always leave your opponent with $4n+1$ matches where n is an integer. Of course, if your opponent is savvy, she will try and do the same and so who wins will depend on the initial number of matches and who goes first.

Let us consider how you would analyze Nim with reinforcement learning. We assume that your opponent behaves randomly rather than optimally. The state, S, is the number of matches left and the action, A, is the number of matches picked up. With probability $1-\varepsilon$ the algorithm picks the best action identified so far for a particular state and with probability ε it randomly chooses an action. We start by setting all the Q's equal to 0. This is shown in Table 7.5. Somewhat arbitrarily we set the reward for winning the game to +1 and the reward for losing the game to −1. In this example there is no reward until the end. We assume that α in equation (7.3) equals 0.05.

Table 7.5 Initial Q-values

Matches picked up	*State (= number of matches left)*						
	2	*3*	*4*	*5*	*6*	*7*	*8*
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3		0	0	0	0	0	0

To keep the example simple, the games we consider start with only 8 matches. Suppose that on the first game you pick up 1 match, your opponent's (random) decision is to pick up 3 matches. You then pick up 1 match and your opponent picks up 3 matches. You win and obtain a reward of +1. Equation (7.3) gives

$$Q(8, 1) = 0+0.05(1-0) = 0.05$$

because, when there are 8 matches and you pick up 1, you end up getting a payoff of +1. Also

$$Q(4, 1) = 0+0.05(1-0) = 0.05$$

because, when there are 4 matches and you pick up 1, you end up getting a payoff of +1. This leads to Table 7.6.

Table 7.6 *Q*-values after one game

Matches	*State (= number of matches left)*						
picked up	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	0	0	0.05	0	0	0	0.05
2	0	0	0	0	0	0	0
3		0	0	0	0	0	0

Suppose that on the next game you initially pick up 1 match and your opponent picks up 2 matches. You pick up one match and your opponent picks up 3 matches. You have to pick up the remaining match and lose for a payoff of −1. This leads to Table 7.7 with $Q(8,1)$ and $Q(5,1)$ being updated as follows:

$$Q(8, 1) = 0.05+0.05(-1-0.05) = -0.0025$$

$$Q(5,1)= 0+0.05(-1-0) = -0.05$$

Table 7.7 *Q*-values after two games

Matches	*State (= number of matches left)*						
picked up	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	0	0	0.05	−0.05	0	0	−0.0025
2	0	0	0	0	0	0	0
3		0	0	0	0	0	0

Tables 7.8, 7.9, and 7.10 show the situation after 1,000, 5,000, and 25,000 games for one simulation assuming that the initial value of ε is 1 and the decay factor applied to it is 0.9995.[3] The algorithm requires more data than the multi-armed bandit example, but eventually it learns the correct strategy which is:

- When there are 8 matches, the correct strategy is to pick up 3 matches.
- When there are 6 matches, the correct strategy is to pick up 1 match.
- When there are 5 matches there are no good strategies.
- When there are 4 matches, the correct strategy is to pick up 3 matches.
- When there are 3 matches, the correct strategy is to pick up 2 matches.
- When there are 2 matches, the correct strategy is to pick up one match.

Table 7.8 *Q*-values as a function of state and action after 1,000 games (see Excel file for calculations)

Matches	*State (= number of matches left)*						
picked up	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	0.999	-0.141	0.484	-0.122	0.155	0.000	0.272
2	-0.994	0.999	-0.108	-0.276	-0.171	0.000	0.252
3	0.000	-0.984	1.000	-0.070	-0.080	0.000	0.426

Table 7.9 *Q*-values as a function of state and action after 5,000 games (see Excel file for calculations)

Matches	*State (= number of matches left)*						
picked up	*2*	*3*	*4*	*5*	*6*	*7*	*8*
1	1.000	-0.127	0.382	0.069	0.898	0.000	0.786
2	-1.000	1.000	0.222	0.297	-0.059	0.000	0.683
3	0.000	-1.000	1.000	-0.106	0.041	0.000	0.936

[3] In this case there are two key hyperparameters, α and β. Appropriate values can be determined using trial and error.

Table 7.10 Q-values as a function of state and action after 25,000 games (see Excel file for calculations)

Matches picked up	*State (= number of matches left)*						
	2	*3*	*4*	*5*	*6*	*7*	*8*
1	1.000	0.080	0.104	0.069	0.936	0.000	0.741
2	-1.000	1.000	0.103	0.412	-0.059	0.000	0.835
3	0.000	-1.000	1.000	-0.106	0.041	0.000	1.000

7.4 Temporal Difference Learning

The method in the previous section is referred to as the Monte Carlo method. We now present an alternative approach.

In the general situation, we can define $V_t(S)$ as the value at time t assuming that we are in state S and the actions taken subsequently are optimal. Assuming no discounting this means that

$$V_t(S) = \max_A E[R_{t+1} + V_{t+1}(S')]$$

where S' is the state at time t+1 assuming that action A is taken at time t. A similar equation can be used to relate $V_{t+1}$to V_{t+2}, V_{t+2} to V_{t+3}, and so on. This allows a dynamic programming method developed by Richard Bellman to be used in relatively simple situations. We start by considering all the states that could arise at the horizon time T and work backward. First, we calculate the optimal actions for states that could arise at time T−1. Given this we calculate the optimal actions for states at time T−2, and so on.

As mentioned earlier, the strategy for winning at Nim is to leave your opponent with $4n$+1 matches for some integer n. To prove this formally we can show (a) that we win if we leave the opponent with five matches and (b) that if we leave our opponent with $4n$+1 matches after one turn we can leave her with $4(n-1)$ +1 matches after the next turn (n > 1). This is a simple example of dynamic programming where we are in effect working back from the end of the game to find the optimal current decision.

Unfortunately, dynamic programming is not practical for many large problems, but reinforcement learning can use the ideas underlying dynamic programming. As before, we define $Q(S, A)$ as the current estimate of the value of taking action A in state S. The value of being in state S is

$$V(S) = \max_A Q(S, A))$$

For example, after 5,000 games we would calculate from Table 7.9 $V(8)$=0.936, $V(6)$=0.898, $V(5)$=0.297, $V(4)$=1.000, $V(3)$=1.000, and $V(2)$= 1.000.

In the Monte Carlo method, we update $Q(S, A)$ by observing the total subsequent gain, G, when a particular decision is taken in a particular state. We can use the ideas underlying dynamic programming and look just one time-step ahead. Suppose that when we take action A in state S we move to state S'. We can use the current value for $V(S')$ to update as follows:

$$Q^{\text{new}}(S, A) = Q^{\text{old}}(S, A) + \alpha\left[R + \gamma V(S') - Q^{\text{old}}(S, A)\right]$$

where R is the reward at the next step and γ is the discount factor.

In the Nim example, suppose that the current Q-values are those shown in Table 7.9. Suppose further that the results on the next game are as follows:

- You explore and choose 1match
- Your opponent chooses 1 match
- You exploit and choose 1 match
- Your opponent chooses 3 matches
- You exploit and choose 1 match
- Your opponent chooses 1 match
- You win

With α = 0.05 and γ=1, $Q(8,1)$ would be updated as follows

$$\begin{aligned} Q^{\text{new}}(8,1) &= Q^{\text{old}}(8,1) + 0.05\left[V(6) - Q^{\text{old}}(8,1)\right] \\ &= 0.786 + 0.05 \times (0.898 - 0.786) \\ &= 0.792 \end{aligned}$$

Also $Q(6,1)$ would be updated as follows

$$\begin{aligned} Q^{\text{new}}(6,1) &= Q^{\text{old}}(6,1) + 0.05\left[V(2) - Q^{\text{old}}(6,1)\right] \\ &= 0.898 + 0.05 \times (1.000 - 0.898) \\ &= 0.903 \end{aligned}$$

and $Q(2,1)$ would be updated as

$$\begin{aligned} Q^{\text{new}}(2,1) &= Q^{\text{old}}(2,1) + 0.05\big[1.000 - Q^{\text{old}}(2,1)\big] \\ &= 1.000 + 0.05 \times (1.000 - 1.000) \\ &= 1.000 \end{aligned}$$

This procedure is known as temporal difference learning. Here we are looking only one step ahead. (A "step" is a move by you and then by your opponent.) A natural extension of temporal difference learning is where we look *n* steps ahead. This is referred to as *n-step bootstrapping*.

7.5 Deep *Q*-Learning

The temporal difference approach we have described is referred to as *Q*-learning. When there are many states or actions (or both), the cells of the state–action table do not get filled up very quickly. It then becomes necessary to estimate a complete $Q(S, A)$ function from the results that have been obtained. As the $Q(S, A)$ function is in general nonlinear, an artificial neural network (ANN) is the natural tool for this. Using *Q*-learning in conjunction with an ANN is known as *deep Q-learning* or *deep reinforcement learning*.

7.6 Applications

One of the most widely publicized applications of reinforcement learning is AlphaGo. This is a computer program developed by Google to play the board game Go. In May 2017, it surprised professional Go players by beating the world champion Go player, Ke Jie, three games to zero. It generated data to improve its performance by playing against itself many times. (Exercise 7.12 asks you to do something analogous to this by deriving a Nim strategy where your opponent learns how to play the game, rather than making random decisions.)

Reinforcement learning has found applications other than playing games. For example, reinforcement learning is used for driverless cars, resource management, and the programming of traffic lights.[4] Healthcare is an area that has seen interesting applications of rein-

[4] See, for example, the work of H. Mao, M. Alizadeh, I. Menache, and S. Kandula, 2016, entitled "Resource Management with Deep Reinforcement Learning": https://people.csail.mit.edu/alizadeh/papers/deeprm-hotnets16.pdf; and I. Arel, C. Liu, T. Urbanik, and A.G.Kohls, 2010, "Reinforcement Learning-based Multi-agent System for Network Traffic Signal Control," *IET Intell. Transp. Syst.,* 4, 2: 128–135.

forcement learning.[5] Treating a patient is a multistage activity. The doctor chooses one action, observes the result, chooses another action, and so on. If enough data is available, algorithms should be able to determine the optimal action for any state. However, it is worth pointing out some of the problems that have been experienced because they are typical of those encountered in reinforcement learning:

- Data will tend to be biased toward the treatment option that is currently favored by physicians and so it may be difficult for the algorithm to identify treatments that are better than those currently used.
- It is difficult to come up with a reward function. How, for example, do you trade off quality of life with the length of a patient's life?
- A sufficient quantity of relevant data might not exist or if it does exist it might not have been collected in a way that can be used by a reinforcement learning algorithm.

Reinforcement learning generally requires much more data than supervised learning. Often the data is simply not available. An analyst can then try to determine a model of the environment and use that as a way of generating simulated data for input to a reinforcement learning algorithm,

There are a number of potential applications of reinforcement learning in finance. Consider a trader who wants to sell a large block of shares. What is the optimal strategy? If the trader chooses to sell all the shares in a single trade, she is likely to move the market and the price received may be less than that realized if a series of small trades are undertaken. But if the share price declines sharply, a series of small trades will not work out well.[6]

Another application is to portfolio management.[7] This is a multistage activity. Changing the composition of a portfolio too frequently

---

[5] See I. Godfried, 2018, "A Review of Recent Reinforcement Learning Applications to Healthcare" at https://towardsdatascience.com/a-review-of-recent-reinforcment-learning-applications-to-healthcare-1f8357600407.

[6] This is considered by a number of authors, for example, Y. Nevmyvaka, Y. Feng, and M. Kearns, "Reinforcement Learning for Optimized Trade Execution." https://www.seas.upenn.edu/~mkearns/papers/rlexec.pdf.

[7] For some examples of this, see Y. Huang, "Financial Trading as a Game: A Deep Reinforcement Learning Approach," arXiv:1807.02787**;** Z. Liang, H. Chen, J. Zhu, K. Jiang and Y. Li, "Adversarial Deep Reinforcement Learning in Portfolio Management," arXiv:1808.09940; Z. Jiang, D. Xu, and J. Liang, "A Deep Reinforcement Learn-

will involve transaction costs.[8] The past history of stock price returns can be used to evaluate the actions that should be taken in different circumstances. In this case, the reward function should be chosen carefully so that it penalizes risks as well as encouraging strategies that lead to high expected returns.

A further application is to hedging. There is a trade-off between the frequency of hedging and the reduction in risk. Risk can be reduced by increasing the frequency of trading but this also leads to an increase in transaction costs. Traditionally, derivatives have been hedged by calculating their theoretical sensitivity to the price of the underlying asset, the volatility of the underlying asset, and other risk factors. Reinforcement learning provides an alternative which we discuss further in Chapter 10.

Summary

Reinforcement learning is concerned with sequential decision making. The set-up involves actions and states. The actions taken lead to rewards and costs. The Q-function estimates the expected reward (net of costs) from taking a particular action when the environment is described by a particular state. The best action in a particular state is one for which the Q-function is greatest.

An important aspect of reinforcement learning is the exploitation vs. exploration choice. When learning from simulated or historical data, it is tempting to take an action that seems best based on the data that has been seen so far. However, if the algorithm always does this, it stops learning because it never tries a new action. A reinforcement learning algorithm therefore assigns a probability ε to an action that is chosen randomly and $1-\varepsilon$ to the best action identified so far. Typically, ε is close to one initially and declines as the model learns from data.

We have illustrated the exploitation vs. exploration trade-off with two examples. One involves the multi-armed bandit problem which is a well-known problem in statistics. A gambler attempts to learn which of a number of one-armed bandits in a casino gives the highest average payout. This is a relatively simple example of reinforcement learning

---

ing Framework for the Financial Portfolio Management Problem," arXiv: 1706.10059.pdf

[8] One source of transaction costs is the bid–ask spread. A portfolio manager typically has to buy at a market maker's ask price and sell at the market maker's bid price with the ask price being greater than the bid price.

because the environment (i.e., the state) never changes. The other example involves the game of Nim where the state is defined by the number of matches left and the action is the number of matches picked up. In both cases, we have shown that reinforcement learning provides a way of learning the best strategy.

The value of taking a particular action in a particular state is referred to as the Q-value. The best value for a state is the maximum of the Q-values over all possible actions. There are a number of different ways of updating Q-values. One is to base the updating on the total net reward (possibly discounted) between the current time and the horizon date. Another is to look only one action ahead and base the updating on the best value calculated so far for being in the state that exists at the time of the next action. Other updating procedures are between these two extremes where we look several actions ahead in calculating the consequences of an action.

In real-world applications of reinforcement learning there are usually a large number of states and actions. One way of coping with this is to use reinforcement learning in conjunction with artificial neural networks (ANNs). Reinforcement learning generates the Q-values for some state–action combinations and an ANN is used to estimate a more complete function.

SHORT CONCEPT QUESTIONS

7.1 How does reinforcement learning differ from supervised learning?
7.2 Explain why a reinforcement learning algorithm needs to involve both exploration and exploitation.
7.3 Explain how dynamic programming works.
7.4 What is the optimal strategy for playing Nim? To what extent has the Monte Carlo simulation found the best action after 1,000, 5,000, and 25,000 games in Tables 7.8 to 7.10?
7.5 Explain the Monte Carlo approach to reinforcement learning.
7.6 What is meant by temporal difference learning?
7.7 Why is it sometimes necessary to use an artificial neural network in conjunction with reinforcement learning?
7.8 What is meant by deep Q-learning?

EXERCISES

7.9 Suppose that Table 7.8 shows the current Q-values for Nim. In the next game you win because one match is always picked up by both you and your opponent. How would the table be updated for (a) the Monte Carlo approach and (b) the temporal difference learning approach?

7.10 Use the worksheet in www-2.rotman.utoronto.ca/~hull for the multi-armed bandit problem to investigate the impact of using different values for the decay factor, β.

7.11 Change the Nim Visual Basic program available at

www-2.rotman.utoronto.ca/~hull

so that it uses temporal difference learning rather than the Monte Carlo approach. Compare how quickly the two approaches find the best move when there are eight matches.

7.12 Change the Nim Visual Basic program available at

www-2.rotman.utoronto.ca/~hull

so that your opponent learns the best strategy rather than behaving randomly.

Chapter 8

Natural Language Processing

Up to now we have talked about applying machine learning algorithms to numerical or categorical data. We now move on to consider the way machine learning can handle language. This is known as *natural language processing* (NLP) or *computational linguistics*. It is becoming increasingly important because much of the data generated in the world is in the form of written or spoken words.

The development of NLP applications is challenging because the rules of language are difficult to communicate to a machine and words can have several meanings. Also, a human being can pick up nuances in language which are almost impossible for a machine to recognize. For example, it is difficult for a machine to recognize sarcasm or irony. In spite of this, a great deal of progress has already been made and we can expect to see exciting developments in NLP in the future.

In Chapter 1, we mentioned Google Neural Machine Translation which has been very successful in translating text from one language to another. Applications such as Siri on iPhones and Amazon's Alexa can recognize human speech to perform a variety of simple tasks. Programs for accurately converting speech to text work well. It seems likely that machines will soon take over the role of professional translators. Two individuals who speak different languages will then be able to communicate seamlessly.

There are many different natural language processing applications. This chapter will focus for the most part on what is referred to as *sentiment analysis*. This involves the processing of data from such sources as surveys and social media to determine whether it is positive, negative, or neutral about a particular product, company, person, event, etc. The sheer volume of the data available today often makes manual processing unrealistic.

NLP allows a company to monitor customer responses to its products and its actions. This can be done in real time and provides important inputs to the company's decision making. For example, when a company markets a new product, the comments from customers can be lead to timely decisions. In 1985, Coca-Cola changed its drink's formula for the first time in 99 years. The new product was not popular and the older formula was eventually reintroduced as Coca-Cola Classic. Today, NLP and the huge amount of data that is available on social media and elsewhere would enable the company to determine the market's response to the new formula very quickly.

When a company uses a new advertisement, it can use NLP to assess how well it is received by consumers. If the reaction of consumers is negative, the advertisement can be pulled quickly. This could have been useful for Gillette's "the best men can be" advertisement in January 2019, which was not at all well received. NLP can also be used to avoid public relations disasters. When United Airlines forcibly removed a passenger who was an Asian-American doctor from one of its planes in April 2017, the initial statements from the company only served to inflame the situation. NLP could have been used to assess the public's response very quickly. The company could then have issued an almost immediate unconditional apology to mitigate the event's negative impact (particularly on its Asian customers).

NLP has many applications for stock market investors. If news reports about a company or quarterly earnings calls to analysts are positive (negative), we might expect the stock price to increase (decrease). Zhang and Skiena were among the first researchers to investigate this.[1] They ranked companies by a sentiment measure derived from news reports each day. They then constructed a portfolio which was long stocks where the sentiment measure was positive and short stocks where the sentiment measure was negative. The value of the stocks in the long portfolio equaled the value of the stocks in the short portfolio

[1] See W. Zhang and S. Skiena, "Trading strategies to exploit blog and news sentiment," 2010, Proceedings of the 4th international AAAI Conference on Weblogs and Social Media.
www.aaai.org/ocs/index.php/ICWSM/ICWSM10/paper/viewFile/1529/1904

so that the portfolio was market neutral (i.e., its return was not affected by the performance of the stock market as a whole). Their results showed that trading relatively few stocks and holding them for short periods produced impressive returns.

Before readers rush out to develop their own trading strategies based on media reports, a word of caution is in order. An important theory concerning the way prices are formed in financial markets is the efficient markets hypothesis. This argues that financial markets reflect all known information. As NLP research findings, such as those of Zhang and Skiena just mentioned, get well known we can expect more traders to use NLP to guide their trading and as a result market prices will adjust almost immediately to news reports. It will then not be possible to generate the large returns realized by researchers in their experiments.

Does this mean that it is too late to profit from using NLP for investing? That is not necessarily the case. New data sources are becoming available all the time. One approach is to try and be one step ahead of most others in exploiting these new data sources. Another is to develop better models than those being used by others and then be very secretive about it. Renaissance Technologies, a hedge fund, provides an example of the second approach. It has been amazingly successful at using sophisticated models to understand stock price patterns. Other hedge funds have been unable to replicate its success. The average return of its flagship Medallion fund between 1988 and 2018 was 66% per year before fees. This included a return of close to 100% in 2008 when the S&P 500 lost 38.5%. Two senior executives, Robert Mercer and Peter Brown, are NLP experts and have been running the company following the retirement of the founder, Jim Simons, in 2009.[2]

Python contains a number of tools for NLP. Downloading data from the web is referred to as *web scraping* (and also as screen scraping, web data extraction, and web harvesting). Most data on the web is in the form of *.html files and Beautiful Soup is a useful resource for converting these to files than are better for analysis. Natural Language Toolkit (NLTK) is a platform for building Python programs. It contains many different tools to help out with the analyses that will be described in this chapter.

In the next few sections, we will assume that opinions about some activity of a company are to be classified. But the approaches suggested can be used in many other situations. For example, another popular ap-

[2] For more information about Renaissance Technologies, see G. Zuckerman, *The man who solved the market: How Jim Simons launched the quant revolution*, 2019, Penguin Random House.

plication is to distinguish emails that are spam from those that are non-spam (sometimes referred to as ham).

8.1 Sources of Data

In sentiment analysis, an analyst wants to use data that has been collected to predict the sentiment of new opinions. A conclusion from the analysis might be "the opinions being expressed at the moment about our product are 82% positive and 18% negative." If opinions are being classified as "positive", "negative", or "neutral", the conclusion might take the form "the opinions currently being expressed about our product are 60% positive, 20% negative and 20% neutral." Sometimes a numerical scale is involved, e.g., 1=very negative, 2=somewhat negative, 3= neutral, 4=somewhat positive, 5=very positive. An output from NLP could then be, "the average current sentiment is 3.9."

The general approach to sentiment analysis is similar to that for other machine learning applications that have been discussed in this book. We collect data that has been labeled in one of the ways just discussed (e.g., positive or negative). We divide the data into a training set and a test set. (If several different models are being compared a validation set is also a good idea, as discussed in Chapter 1.) We use the training set to develop the required model(s). A validation set can be used to choose between models. The test data evaluates the accuracy of the chosen model. The model is then used as a classification tool for new opinions.

Where do the labeled opinions come from? We have to base the labels on past opinions. There are publicly available data sets where opinions have been labeled and these are sometimes used to train and test a model. However, data sets that have been used for one situation may not be appropriate for another. For example, opinions about movies together with positive/negative labels may not be appropriate for assessing opinions about a consumer product used in the kitchen.[3]

The best (and most expensive) approach involves a company collecting a large number of opinions that customers have given for its products or actions in the past and asking one or more human beings to label them in one of the ways discussed above.

It is worth noting that that human beings agree on how an opinion should be labeled only about 80% of the time and so a model that agrees with human judgement 100% of the time is unrealistic. Typical-

[3] Movie reviews are a convenient source of data for sentiment analysis because they are usually labeled with between one and five stars.

ly a good model will agree with human judgement perhaps 70% of the time.

8.2 Pre-Processing

Suppose we have managed to obtain a large number of labeled opinions with which to construct a model. A first stage is often pre-processing, which is a type of data cleaning. The main objectives of pre-processing is to identify a vocabulary of words that will be considered.

The first stage in pre-processing is known as *word tokenization* and involves splitting text into a set of words. It involves looking for spaces and punctuation. For example,

"The product works well! I would recommend the product to someone else."

becomes

"The", "product", "works", "well", "!", "I", "would", "recommend", "the", "product", "to", "someone", "else", "."

Punctuation can be removed as it usually does not add much information. Also, we do not want an algorithm to consider "The" and "the" to be different words. It therefore makes sense to convert all upper case characters to lower case.

It is common to remove what are termed *stop words*. These are words like "the", "a", "in" and "an" which are very common but add very little to the meaning of text. NLTK has a list of stop words in different languages and provides a procedure for removing stop words from text. An analyst can change the list. This can be appropriate in some circumstances and the way it is done can depend on the nature of the text being considered. For example, words removed in legal documents might be different from those removed in a news article. One approach that can be used to identify stop words is to list the 10 or 20 most commonly occurring words in the opinions and then decide whether they should be retained.

Another type of pre-processing that is sometimes used involves what is referred to as *stemming*. This is the removal of suffices such as "s", "ing", "like", and "ly". Doing stemming for English text is not trivial. For example, we would like to replace "drinking" by "drink" but "sitting" by "sit" and lying" by "lie". A key objective of stemming is that related

words map to the same stem. The stem itself does not have to be a word. Thus "arguable", "argues", "argued", "arguing" might all be mapped to "argu". A related procedure is *lemmatization*. This uses a look up table to determine a mapping of a word into its root word. For example, "worse" would be mapped to "bad". Also, when a word has more than one meaning, lemmatization may attempt to use the context (i.e., the surrounding words) to find an appropriate root word.

Correcting spelling mistakes is desirable because it avoids duplicate words such as "machine" and "machne" or "learning" and "learnig". Fairly sophisticated spell checkers are now available, but they are not perfect and occasionally change the intended meaning of a sentence. It is also important to recognize abbreviations. For example, "u" in a text message should be changed to "you" and "approx" should be changed to "approximate".

It may also be appropriate to remove rare words that appear only once or twice in the training set. Suppose we are trying to predict whether a certain news report has a positive or negative effect on a stock price. If the news report includes the word "myopic", and that is the only time the word is used in the training set, it is unlikely that a model would be able to conclude with any confidence the word has information content.

The key point here is that the words we finally choose for the vocabulary will be used to classify opinions. Words which appear very occasionally are of little use as are words that appear in virtually all opinions. One approach to help identify the words that should be used is to retain only those that appear in between, say, 20% and 80% of opinions. (Some experimentation can be used to determine the right high and low percentages here.)

8.3 Bag-of-Words Model

Assume we have done pre-processing and formed a vocabulary that will be used for classification. Although we have done our best to reduce the number of different words that are considered, we may still have 10,000 or more words in the vocabulary. In the bag-of-words model each opinion is characterized by the number of times each word appears. To illustrate this with a simple example, suppose that our vocabulary consists of the following list of 10 words

"bad", "good", "great", "much", "never", "product", "recommend", "someone", "terrible", "well"

Suppose further that we want to classify the opinion mentioned earlier:

"The product works well! I would recommend the product to someone else"

A bag-of-words would convert this opinion to

(0, 0, 0, 0, 0, 2, 1, 1, 0, 1)

This indicates that the first word in the list, "bad", does not appear; the next four words in the vocabulary also do not appear; the word "product" appears twice; and so on. Note that words such as "the" and "works" appear in the opinion, but not in the vocabulary, and are therefore ignored.

How do we decide whether the opinion is positive or negative? A simple approach, which does not involve machine learning, would be to write down a list of positive and negative words and determine whether the opinion contains more positive words than negative words, or vice versa. The positive words could include "good", "great", "recommend", and "well". The negative words could include "bad", "never", and "terrible". If this sort of list making is too much work, there are a number of sentiment lexicons that can be used.

In the case of the opinion in our example, there are two positive words ("recommend" and "well") and no negative words and so the opinion would be classified as positive. Labeled data can be used to estimate the accuracy of the model.

There is no learning going on in this simple approach as the positive and negative words are provided externally. A more sophisticated approach is to use the training set to determine the words that most commonly occur in positive and negative opinions. The words that occur much more frequently in positive opinions than negative opinions would be categorized as "positive words" while those that occur much more frequently in negative opinions than positive opinions would be categorized as "negative words". We would have to decide what we mean here by "much more frequently." How much more frequently do words have to be in positive opinions than negative opinions in order to be "positive words". Similarly, how much more frequently do words have to be in negative opinions rather than positive opinions to be "negative words". The required difference between the frequencies are hyperparameters and it is likely to be desirable to use a validation set to choose good values for them.

One decision that has to be made in all approaches is whether positive or negative words that occur two or more times in an opinion should be given more weight. If in our example the word "well" appeared twice instead of once, should it be considered as being equivalent to two positive words or one positive word? There is some research indicating that the repetition of a word provides little additional information.[4]

The bag-of-words model takes no account of the order of words. It just counts the words. Unfortunately, simple word counts can give incorrect answers. One reason is the presence of negatives. Consider for example the opinion:

"I would not recommend this product"

Classifying this as a positive opinion because it includes the word "recommend" would give an incorrect result. One possible improvement is to consider word pairs as positive or negative. In the opinion just given the word pairs would be "I would", "would not" "not recommend", "recommend this", and "this product". The word pair "not recommend" would almost certainly be on the negative list.

A set of *n* consecutive words is referred to as an *n-gram*. One word is a *unigram*, two consecutive words is a *bigram*, three consecutive words is a *trigram*, and so on. We have just shown that bigrams can identify the negation of a word and avoid some incorrect signals. Trigrams can, in principle, work even better than bigrams. For example, the trigram "not so bad" could be a positive trigram. Of course, the problem here is that the number of possible bigrams is much greater than the number of possible unigrams, and the number of possible trigrams is greater again. In what follows we will assume that the analysis is based on unigrams (single words). But the approaches can be extended to cover bigrams and trigrams.

8.4 Application of Naïve Bayes Classifier

We introduced the naïve Bayes classifier in Section 4.4. It is a popular approach for sentiment analysis. Here we introduce an application of the classifier based on whether particular words occur in opinions.

[4] See for example, B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment classification using machine learning techniques" in Proceedings of Empirical Methods for Natural Language Processing, 2002.

The application can be extended so that it is based on a count of the number of times different words appear.[5] The naïve Bayes classifier assumes that the occurrence of word X in an opinion is uncorrelated with the occurrence of word Y for all the X and Y that are in the vocabulary being used for classification.

Suppose that there are m words in the vocabulary and our objective is to classify opinions two ways: positive or negative. If the jth word in the vocabulary is in a particular opinion, we define p_j as the probability that word j appears in positive opinions and q_j as the probability it appears in negative opinions. If the jth word in the vocabulary is not in a particular opinion, we define p_j as the probability that word j does not appear in positive opinions and q_j as the probability it does not appear in negative opinions. The naïve Bayes classifier gives the probability that the opinion is positive as

$$\text{Prob(Positive|words)} = \frac{p_1 p_2 \ldots p_m}{\text{Prob(words)}} \text{Prob (Positive)}$$

and the probability that it is negative as

$$\text{Prob(Negative|words)} = \frac{q_1 q_2 \ldots q_m}{\text{Prob(words)}} \text{Prob (Negative)}$$

where "words" refers to the list defining whether words are in an opinion or not, and Prob (Positive), Prob (Negative), and Prob (words) are unconditional probabilities. Because the two probabilities must add up to one, Prob(Positive|words) is

$$\frac{p_1 p_2 \ldots p_m \times \text{Prob (Positive)}}{p_1 p_2 \ldots p_m \times \text{Prob (Positive)} + q_1 q_2 \ldots q_m \times \text{Prob (Negative)}}$$

while Prob(Negative|words) is

$$\frac{q_1 q_2 \ldots q_m \times \text{Prob (Negative)}}{p_1 p_2 \ldots p_m \times \text{Prob (Positive)} + q_1 q_2 \ldots q_m \times \text{Prob (Negative)}}$$

The naïve Bayes classifier can easily be extended to the situation where there are more than two classifications. For example, if we are classifying an opinion as positive, negative, or neutral, we can define r_j

[5] As indicated in the previous section, multiple occurrences of a word in an opinion may not have more information than a single occurrence.

as the probability that word *j* appears in a neutral opinion if it is in the opinion under consideration and as the probability that word *j* does not appear in a neutral opinion if it is not in the opinion under consideration. Define *P*, *Q*, and *R* as the unconditional probability of positive, negative, and neutral observations. Then

$$\text{Prob(Positive|words)} = \frac{p_1 p_2 \ldots p_m P}{p_1 p_2 \ldots p_m P + q_1 q_2 \ldots q_m Q + r_1 r_2 \ldots r_m R}$$

$$\text{Prob(Negative|words)} = \frac{q_1 q_2 \ldots q_m Q}{p_1 p_2 \ldots p_m P + q_1 q_2 \ldots q_m Q + r_1 r_2 \ldots r_m R}$$

$$\text{Prob(Neutral|words)} = \frac{r_1 r_2 \ldots r_m R}{p_1 p_2 \ldots p_m P + q_1 q_2 \ldots q_m Q + r_1 r_2 \ldots r_m R}$$

To provide a simple example of these equations, suppose that there are 10 observations in the training set and only two words. Table 8.1 shows whether a particular word is in an observation (1 indicates that it is in the observation and 0 indicates that it is not.

Table 8.1 Example of a situation where there are 10 opinions in the training set and two words. 1 indicates that a word is present while 0 indicates that it is not present.

Opinion	1	2	3	4	5	6	7	8	9	10
Word 1	1	1	1	0	0	0	0	0	0	1
Word 2	0	0	1	1	1	1	0	1	0	0
Label	Pos	Pos	Pos	Pos	Neg	Neg	Neg	Neut	Neut	Neut

Consider an opinion that contains word 1 but not word 2. What is the probability that it is favorable? In this case, $p_1 = 3/4 = 0.75$ (because three of the four positive observations in the training set contain word 1) and $p_2 = 2/4 = 0.5$ (because two of the four positive observations do not contain word 2). Similarly, $q_1 = 0$, $q_2 = 0.33$, $r_1 = 0.33$, and $r_2 = 0.67$. The unconditional probability of a positive, negative, and neutral opinion are 0.4, 0.3, and 0.3, respectively. The equations given above show that the conditional probability that the opinion under consideration is positive is:

$$\frac{0.75 \times 0.50 \times 0.4}{0.75 \times 0.50 \times 0.4 + 0 \times 0.33 \times 0.3 + 0.33 \times 0.67 \times 0.3} = 0.69$$

The conditional probability that it is negative is

$$\frac{0 \times 0.33 \times 0.3}{0.75 \times 0.50 \times 0.4 + 0 \times 0.33 \times 0.3 + 0.33 \times 0.67 \times 0.3} = 0$$

The conditional probability that it is neutral is

$$\frac{0.33 \times 0.67 \times 0.3}{0.75 \times 0.50 \times 0.4 + 0 \times 0.33 \times 0.3 + 0.33 \times 0.67 \times 0.3} = 0.31$$

The performance of the model can be assessed with a test set.

Our baby example illustrates one problem with the naïve Bayes classifier. There is a chance that a particular word, *j*, appears in the opinion under consideration but does not appear in any of the training set observations that have a particular label. The probability of the opinion having that label is then zero. In our example, word 1 appears in the opinion under consideration but does not appear at all in the observations of the training set that are labeled negative. Hence q_1 =0 and the probability of the opinion being negative is bound to be calculated as zero. This would be the case even if the opinion contained many other words that were present in observations of the training set that were labeled negative. All other negative words have no weight if one particular word is not found in the negative training set observations.

Assigning a zero conditional probability may be too extreme. A way of making the zero probabilities slightly positive, so that a more reasonable result is obtained, is known as *Laplace smoothing*. In this case we can imagine adding two new observations for each of the three classes in such a way that each word is present in one of the two observations and not present in the other. This involves adding a total of six observations of which two are positive, two are negative and two are neutral. The unconditional of positive, negative and neutral opinions become 6/16, 5/16, and 5/16 instead of 4/10, 3/10, and 3/10. Furthermore, p_1 and p_2 become 4/6 and 3/6 instead of 3/4 and 2/4, respectively; q_1 and q_2 become 1/5 and 2/5 instead of 0/3 and 1/3, respectively; r_1 and r_2 become 2/5 and 3/5 instead of 1/3 and 2/3, respectively.

The new probability of that the opinion is positive is

$$\frac{0.667 \times 0.5 \times 0.375}{0.667 \times 0.5 \times 0.375 + 0.2 \times 0.4 \times 0.3125 + 0.4 \times 0.6 \times 0.3125} = 0.556$$

The probability that it is negative is

$$\frac{0.2 \times 0.4 \times 0.3125}{0.667 \times 0.5 \times 0.375 + 0.2 \times 0.4 \times 0.3125 + 0.4 \times 0.6 \times 0.3125} = 0.111$$

The probability that it is neutral is

$$\frac{0.4 \times 0.6 \times 0.3125}{0.667 \times 0.5 \times 0.375 + 0.2 \times 0.4 \times 0.3125 + 0.4 \times 0.6 \times 0.3125} = 0.333$$

In this simple example, the impact of Laplace smoothing on the probabilities is quite large. As the data set become bigger, its impact declines.

8.5 Application of Other Algorithms

The naïve Bayes classifier has the advantage that it is easy to implement and very fast. However, the independence assumption is imperfect because some words often occur together. For example, a commonly occurring phrase in opinions might be "easy to use". The words "easy" and "use" would then have a joint probability of occurring in a positive opinion that is much greater than the product of "easy" occurring and "use" occurring.

Other classification algorithms that have been described in this book are logistic regression, decision trees, SVM, and neural networks. When they are used for sentiment analysis, the set up is similar to that for the examples given earlier in the book. The features (i.e., the words in the vocabulary used for classification) have values of either 0 or 1 with 1 indicating that the word is present and 0 indicating that it is not present. We divide the available data into a training set and a test set. (If several different models are being compared a validation set is also a good idea as discussed in Chapter 1.) We use the training set (and possibly a validation set) to develop a classification model. The test data evaluates the accuracy of the model.

Consider first SVM. Unlike the other algorithms, this merely classifies opinions. It does not provide probabilities. In this respect, it is similar to the approaches suggested in Section 8.3. However, it is potentially superior to those approaches because it can detect that some words are more positive (negative) than others.

The SVM method described in Chapter 5 can be extended so that there are more than two classes. This involves the estimation of the positions of multiple hyperplanes rather than just one. Unlike logistic regression, the SVM algorithm has the advantage that it can be used even

when the number of features (i.e., words in the vocabulary) is greater than the number of observations in the training set.

Logistic regression is designed to accommodate two classes but there are ways of extending it to accommodate more than two classes. It has the advantage over SVM that it provides probabilities for the classes and has the advantage over naïve Bayes that it does not make the independence assumption.

Decision trees and neural networks can be used to handle multiple classes. In Section 3.9 we introduced a maximum likelihood objective function when there are only two classes. This can be generalized to more than two classes. If Q_i is the probability that we predict that observation i from a labeled set will fall into its correct class, we want to maximize

$$\sum_i \ln(Q_i)$$

The performance of a chosen algorithm can be assessed using measures similar to those presented in Section 3.11.

8.6 Information Retrieval

We now leave sentiment analysis to consider the application of NLP to information retrieval. This is important for search engines and it can also be important for businesses that want quick access to a library of documents that are stored electronically.

When words are input to a search engine by a user, how does it decide the most relevant documents to present to the user? Two measures that are commonly used are *term frequency* (TF) and *inverse document frequency* (IDF). The TF for a document and a word is defined as

$$\frac{\text{Number of times the word appears in the document}}{\text{Number of words in the document}}$$

The IDF for a word is defined as

$$\log\left(\frac{N}{n}\right)$$

where N is the total number of documents and n is the number of documents containing the word.[6] The TF-IDF is formed by multiplying TF and IDF and is a score for a particular word in a particular document.

Consider a corporate search engine where there are 10,000 documents. To ensure fast information retrieval, the documents are preprocessed and word counts are calculated. Suppose we input "the travel policy" into the search engine. The engine will calculate the TF-IDF for each of the words: "the", "travel", and "policy". The word "the" will have a relatively high TF for each document but its IDF will be zero because it will appear in every document so that $N = n$. Suppose that the word "travel" appears in 10% of the documents. Its IDF is

$$\log (10) = 3.32$$

(assuming that the logarithm is calculated using base 2). The TF-IDF of the word "travel" is calculated for each document by multiplying this by the proportion of words that are "travel" in the document. (For 90% of the documents this percentage is zero and so the TF-IDF is zero.) The word "policy" is handled similarly.

For each document, we can calculate a score as

$$\text{TF-IDF}(\text{“the”}) + \text{TF-IDF}(\text{“travel”}) + \text{TF-IDF}(\text{“policy”})$$

The documents are ranked by their score. The one with the highest score will be presented to the user first; the one will the next highest score will be presented second; and so on. Note that the word "the" plays no role in determining which documents are returned. Also, if a document contains neither "travel" nor "policy" it will have a score of zero. Interestingly, with this simple information retrieval algorithm, the order of the words input by the user makes no difference.

8.7 Other NLP Applications

There are many different NLP applications that have not been discussed in this chapter. In this section we briefly review a few of them.

If we give a machine two different words such as "many" and "numerous", how can it tell that they have a similar meaning? One way is by

[6] Logarithms are usually calculated using the base 2 in computer science. If another base is used all the IDFs are just multiplied by a constant and results are not affected.

looking at what other words they tend to be used with.[7] We could for example, look at a large number of documents and ask the question: if word X is in the document, what is the probability of word Y being close? By "close" we might mean that word Y has to be within five words of word X.

If there are 10,000 words in the vocabulary being used, this could lead to a 10,000 by 10,000 table showing the probability of any one word being close to any other word in a document. We would expect the rows for two words that have similar meanings to be similar to each other. It turns out that the information in such a table can be summarized by a rather smaller table that is 10,000 by 300.[8] (The procedure determining the smaller table is similar to autoencoding which was discussed in Chapter 6.) The meaning of words can therefore be represented by what are termed *word vectors* with 300 entries. This is known as *word embedding*. It turns out that these word vectors have nice addition and subtraction properties. For example, it is approximately true that the following relationship holds between vector representations:

$$\text{king} - \text{man} + \text{woman} = \text{queen}$$

Another application of NLP is to word sequences. It asks the question: What is the probability of a particular word sequence such as "I will give you the answer to the question later today" occurring in text? Clearly it is more likely than a sequence where the words have been jumbled such as "I you give will the answer later to the question today." This has important applications in

- translating from one language to another,
- speech recognition
- using NLP to summarize texts
- the conversion of speech to text.

One idea for estimating the probability of a word sequence is to see how often the text appears in a large number of documents. However, this is infeasible. The chance of a sequence of words such as the one just mentioned appearing in even millions of pages of text is virtually zero. What we have to do in practice is break the sentence down into subse-

[7] This was suggested by J.R. Firth in 1957, who is known for his famous quotation: "You shall know a word by the company it keeps."
[8] Tables which have 50, 100, or 200 columns instead of 300 are sometimes used.

quences of words. For example, we might consider the probability of occurrence of "I will", "will give", "give you", "you the", etc.

Translating from one language top another is a very challenging NLP application. There are a number of approaches. Google's GNMT system, which was mentioned in Chapter 1, uses a long short-term memory recurrent neural network (see Section 6.8). This proved to be a big improvement over its previous system which involved translating on a phrase-by-phrase basis. The phrase-by-phrase system was in turn an improvement over a previous word-by-word translation approach.

Summary

Natural language processing (NLP) involves translating words into numbers so that they can be analyzed. One important application of NLP is to sentiment analysis. This is concerned with determining the nature of opinions such as those in reviews and tweets. The opinions can be classified as positive or negative, with neutral being a possible third category. Alternatively, they can be expressed on a numerical scale (e.g. 1 to 5).

One of the most challenging aspects of sentiment analysis is obtaining relevant labeled opinions that can be used for training a model and testing it. Sometimes publicly available data (e.g., from movie reviews) can be used. When this is not appropriate it is necessary to collect opinions have that have been made in the past and undertake the laborious task of classifying them manually.

Opinions must be processed before they can be used in a model. This involves such tasks as separating out the words, eliminating punctuation, changing upper case letters to lower case, removing commonly occurring words, and removing words that are very rare. The result is a vocabulary of words that will be used for classification.

Bag-of-words models are commonly used for sentiment analysis. The models classifying an opinion depend on whether or not each word in the vocabulary is present in the opinion. Among the machine learning models that can be used are the naïve Bayes classifier, SVM, logistic regression, decision trees, and neural networks.

Search engines are an interesting application of NLP. The task is to choose the most relevant documents from a large number of possibilities from key words input by the user. Important statistics are the frequency with which a particular key word appears in each document and the proportion of all documents in which the word appears.

SHORT CONCEPT QUESTIONS

8.1 What is meant by "sentiment analysis"?
8.2 What are the alternative ways of creating labels for text in a sentiment analysis?
8.3 List five ways in which text can be pre-processed for an NLP application.
8.4 What is the difference between stemming and lemmatization?
8.5 What is a bag-of-words model?
8.6 Why do negative words such as "not" cause a problem in a bag-of-words model?
8.7 What assumption is made when the naïve Bayes classifier is used in sentiment analysis?
8.8 Explain what is meant by a "trigram."
8.9 Give one advantage of logistic regression over (a) SVM and (b) the naïve Bayes classifier.
8.10 What problem is Laplace smoothing designed to deal with?
8.11 Explain how TF and IDF are used in information retrieval.
8.12 What is a word vector?

EXERCISES

8.13 Suppose that there are three words in a vocabulary and we wish to classify an opinion that contains the first two words, but not the third, as positive or negative using the naïve Bayes classifier. The training set is as follows (1 indicates that the opinion contains the word, 0 indicates that it does not):

Opinion	1	2	3	4	5	6	7	8	9	10
Word 1	1	1	1	0	0	0	0	1	1	1
Word 2	0	0	1	1	1	0	0	1	0	0
Word 3	0	0	1	1	0	0	0	0	1	0
Label	Pos	Pos	Pos	Pos	Neg	Neg	Pos	Pos	Neg	Neg

Estimate the probability that the opinion under consideration is (a) positive and (b) negative.

8.14 Download 1000 positive and 1000 negative reviews from http://www.cs.cornell.edu/people/pabo/movie-review-data/

and polarity data set v2.0. Using the naïve Bayes classifier and logistic regression to develop a classification model for the movies.

Chapter 9

Model Interpretability

Supervised learning models produce predictions, but they do not explain their predictions to users. In some cases, this is not important. For example, when a search engine predicts that a particular document will be what the user wants, understanding the underlying model is not important and there is a low cost to the user when the model makes a mistake. But, in many other situations, understanding how predictions are made is desirable because mistakes are costly and confidence in the model is important. As machine learning has become more widely used, researchers have started to devote a great deal of effort to model interpretability issues.[1]

Consider the situation in Section 3.11 where machine learning is being used to accept or reject loan applications. If an applicant for a loan is rejected and asks the reason, it is not likely to be advisable for a bank representative to say: "The algorithm rejected you. Sorry I do not have any more information." The decision made by the algorithm has a potentially high cost to the would-be borrower and the bank's reputation

[1] A book providing an excellent discussion of model interpretability is C. Molnar, *Interpretable Machine Learning*, 2020, https://christophm.github.io/interpretable-ml-book/.

is likely to suffer if the representative blames an algorithm for lending decisions.

As another example, we can consider the predictions for the price of a house in Iowa made in Section 3.8. The person making use of a prediction might be a seller of the house, a buyer of the house, or a real estate agent. In all cases, mistakes are liable to be costly and an individual relying on the prediction is likely to want to know how it was obtained.

Model interpretability has been defined as the degree to which a human being can understand a decision made by a model.[2] One model is more interpretable than another if a human being can more easily understand its output.

Human beings are naturally curious about the predictions made by machine learning models and want to learn from them. In some cases, there may be biases in the model (e.g., involving race or gender) that are unacceptable. Understanding these biases can lead to the model being changed or used in a different way. Sometimes understanding an unfavorable prediction can lead to a decision to take some action that changes a feature so that the outcome being predicted is improved.

It is interesting to note that legislation can require model interpretability. The General Data Protection Regulation in the European Union, which is discussed in Chapter 11, includes a "right to explanation" with regard to machine learning algorithms applied to the data of citizens of the European Union. Specifically, individuals have the right to "meaningful information about the logic involved in, as well as the significance and the envisaged consequences of, such processing for the data subject."

Sometimes, understanding a model can lead one to understand the model's limitations. As a simple example of this, consider image recognition software that distinguishes between polar bears and dogs. It might be found that the model is making predictions by looking at the background (ice vs. grass/trees) rather than the characteristics of the animals. Understanding this clearly indicates a limitation of the model.

There is an amusing story concerning a German horse, named Hans, who in the early 20th century appeared to be intelligent and able to solve mathematical problems (for example, the horse could add, subtract, multiply, divide, and answer questions such as: "if the ninth day of the month is a Wednesday what day of the month is the following Friday?"). Hans indicated answers by stomping his hoof a number of times

---

[2] See T. Miller "Explanation in artificial intelligence: Insights from the social sciences." (2017) arXiv: 1706.07269.

and received a reward when the answer was correct. For some time, the horse was assumed to be intelligent and researchers studied the interesting phenomenon of a horse that could hear mathematical questions and correctly answer them. Eventually, it was found that the horse's real expertise was in reading the expressions on the face of the person asking the questions. Subtle changes in expression led the horse to know when to stop stomping. He did not actually have any mathematical intelligence. The horse can be considered analogous to a machine learning algorithm. Humans at first incorrectly interpreted why Hans gave the answer he did.

The task of understanding a model can be distinguished from the task of understanding a particular prediction made by the model. It is important for companies to have some understanding of a machine learning model so that they can have confidence in the results and know when the environment is such that the model is not applicable. It is also important that particular predictions are explainable.

In this chapter, we will distinguish between models that are intrinsically interpretable (white boxes) and models whose structures do not permit easy interpretation (black boxes). The *k*-nearest neighbors algorithm (see Section 3.12) is clearly in the first category. It is not difficult for someone to understand how the model works and any particular prediction is easily explained. It is essentially a "prediction-by-analogy" model and corresponds to how many human beings make predictions. (For example, a real estate agent in providing advice on the value of a house is likely to use the prices obtained for similar houses that have sold recently.)

As mentioned in Chapter 4, decision trees are fairly easy to explain because they also correspond to the way humans sometimes make predictions. (It is easier for a human to consider one feature at a time rather than all together.) Linear regression is also in the white-box category because the weights that are derived have a simple interpretation.

Models such as neural networks, SVM, and ensemble models such as random forests are in the black-box category. There is no easy way for us to understand how the models work or why the models make a particular prediction.

9.1 Linear Regression

Linear regression is fairly easy to interpret, which in part explains why it is so popular. The model is:

$$Y = a + b_1X_1 + b_2X_2 + \cdots + b_mX_m$$

where Y is the prediction of the target and X_j $(1 \le j \le m)$ are the features. The weight, b_j, can be interpreted as the sensitivity of the prediction to the value of the feature j. If the value of feature j increases by an amount u with all other features remaining the same, the value of the target increases by b_ju. In the case of categorical features that are 1 or 0, the weight gives the impact on a prediction of the target of changing the category of the feature when all other features are kept the same.

The bias, a, is a little more difficult to interpret. It is the value of the target if all the feature values are zero. However, feature values of zero make no sense in many situations. (For example, no houses have a lot size of zero or a first-floor square footage of zero.) For interpretation purposes, it is a good idea to redefine features so that the mean is subtracted. The value of the jth feature then becomes $X_j - \bar{X}_j$ rather than X_j, where $\bar{X}_j$ is the mean of the feature value calculated from all items in the training set.[3] The regression model becomes:

$$Y = a^* + b_1(X_1 - \bar{X}_1) + b_2(X_2 - \bar{X}_2) + \cdots + b_m(X_m - \bar{X}_m) \quad (9.1)$$

The new bias a^* is

$$a^* = a + b_1\bar{X}_1 + b_2\bar{X}_2 + \cdots + b_m\bar{X}_m$$

This bias does have an easy interpretation. It is the value of the target when all features have their average values. In the case of linear regression, it is also the average value of the target.

For a more sophistical interpretation of a linear regression model we can use the statistics mentioned in Section 3.2 that are calculated when linear regression models are implemented. R-squared is an indication of the overall performance of the model and therefore how much it can be relied upon. The t-statistics for the weights can be used to provide confidence limits for the sensitivities. Suppose b_j is 10 with a t-statistic of 5. Because the t-statistic is the weight divided by its standard error, the standard error is 2. We know that the best estimate of the effect of a change u in the value of the jth feature on the value of the target is $10u$ but it could be as low as $6u$ or as high as $14u$.[4]

---

[3] We could use Z-score scaling for the features so that, in addition to subtracting the mean, we divide by the standard deviation. However, this may make the weights more difficult to interpret.

[4] The value is within two standard deviations of the mean about 95% of the time when the data set is large.

So far we have focused on how the model can be understood. We now suppose that we want to explain a particular prediction. A natural approach is to use equation (9.1). The impact of feature j on a prediction, when the average value of the feature is used as a benchmark is

$$b_j\left(X_j - \bar{X}_j\right)$$

As the feature's value, X_j, moves further away from its average value of $\bar{X}_j$, its impact on the prediction made for the target increases. An analyst can take account of the standard error of b_j to produce a range of possible values for the impact of feature j on a particular prediction.

Model interpretability is an important reason for using regularization methods such as Ridge and Lasso. Consider the Iowa house price example in Chapter 3. The regression with 47 features in Table 3.6 does not include any regularization, but it happens to generalize quite well to the validation set with the result that the predictions it produces are quite reasonable.[5] However, the negative weights in the model (e.g., for number of bedrooms), which arise from correlations between features, would be difficult to explain. (Indeed, the model might be dismissed by some users as being ridiculous!) The regularized model in Table 3.7, which is produced using Lasso, is much easier to explain.[6]

Table 9.1 shows how results might be tabulated when the model in Table 3.7 is used to price a house. For completeness, two features appear in Table 9.1 that were not in Table 3.7. These are wood deck (sq. ft.) and open porch (sq. ft.). They had very small (but non-zero) weights of 0.001 and 0.002, respectively in the Python Lasso implementation on scaled data when λ=0.1.

The house of interest is assumed to have a lot area of 15,000 square feet, an overall quality of 6, and so on. To improve interpretability, the Z-score scaling used for Table 3.7 has been reversed in Table 9.1. (It will be recalled that scaling was necessary for Lasso.) This means that the feature weights in Table 3.7 are divided by the standard deviation of the feature and multiplied by the standard deviation of the house price to get the feature weights in Table 9.1.

The table shows that the average lot area for the houses in the training set is 10,249 square feet, compared with 15,000 square feet for the

---

[5] Note that it is not always the case that a model without any regularization generalizes well.

[6] Lasso regularization is also particularly useful from the perspective of model interpretability because it reduces the number of features making it easier for a user to understand the model.

house of interest. The weight for lot area is 0.3795 ($ per square foot). This means that the contribution of lot area to the value of the house is $\$0.3795 \times (15{,}000 - 10{,}249)$ or $1,803.

Table 9.1 Impact of different features on the value of a particular house in Iowa when the model in Table 3.7 is used. See Excel file.

Feature	*House value*	*Average value*	*Feature weight*	*Contribution ($)*
Lot area (sq. ft.)	15,000	10,249	0.3795	+1,803
Overall quality (1 to 10)	6.0	6.1	16,695	−1,669
Year built	1990	1972	134.4	+2,432
Year remodeled	1990	1985	241.2	+1,225
Finished base (sq. ft.)	1,200	444.0	20.42	+15,437
Total basement (sq. ft.)	1,300	1,057	19.08	+4,630
First floor (sq. ft.)	1,400	1,159	6.666	+1,609
Living area (sq. ft.)	2,000	1,503	46.79	+23,273
Number of fireplaces	0	0.6089	2,452	−1,493
Parking spaces	2	1.774	2,857	+646
Size of garage (sq. ft.)	600	474.8	24.42	+3,055
Wood deck (sq. ft.)	0	93.62	0.6364	−60
Open porch (sq. ft.)	0	46.59	2.562	−119
Neighborhood 1	0	0.02611	6,395	−167
Neighborhood 2	0	0.05944	27,523	−1,636
Neighborhood 3	0	0.01611	10,311	−166
Basement quality	5	3.497	1,820	+2,734
Total				+51,532

The model predicts that the price of the house of interest (with the feature values in the second column of Table 9.1) is $232,349. The value of an "average house" with the feature values in the third column of Table 9.1 is $180,817. This house is therefore worth $51,532 more than the average house. The final column of Table 19.1 shows the contribution of each feature to the difference between the value of the house and the value of an average house. (The calculation is shown for the first feature, lot area, above.) A key point is that for a linear model the sum of these contributions equals the difference. (As we discuss later nonlinear models do not have this property.)

The table shows that, for the particular house being considered, the finished basement and living area add most to the value. Many other results can be deduced from the table. For example, the table shows that, if the house had been in neighborhood 2 (the most expensive

neighborhood), the house would have been worth an extra $27,523. (This is because the feature value for neighborhood 2 would be 1 rather than 0.)

It should be noted that the model in Table 9.1, although much better than the original model in Table 3.6, is by no means perfect. It still has some features that are not independent of each other. For example, total basement (sq. ft.) and first floor (sq. ft.) are correlated. When one is high (low), the other is likely to be high (low).[7] In considering the effect of different features, it does not really make sense to consider the effect of the total basement (sq. ft.) increasing without first floor (sq. ft.) also increasing. There is no simple answer to this problem. We could consider grouping two or more features together when considering changes. Sometimes a principal components analysis or an autoencoder can suggest a way to redefine the features so that they are independent (or nearly independent).

9.2 Logistic Regression

The prediction from logistic regression is the probability of a positive outcome, not a value. As explained in Section 3.9,

$$\text{Prob (Positive Outcome)} = \frac{1}{1 + \exp[-(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)]}$$

where X_j is the value of feature j, b_j is its weight and a is the bias. It follows from this that

$$\text{Prob (Negative Outcome)} = \frac{\exp[-(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)]}{1 + \exp[-(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)]}$$

We can calculate the sensitivities of these probabilities to the value of a feature analytically. For example, using a little calculus, it can be shown that when feature j increases by a small amount u, the probability of a positive outcome increases by

$$\frac{\exp[-(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)]}{\{1 + \exp[-(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)]\}^2} b_j u$$

This is

[7] The correlation between the feature values is about 0.79.

$$\text{Prob (Positive Outcome)} \times \text{ Prob (Negative Outcome)} \times b_j u \quad (9.2)$$

Unfortunately, this is only accurate for small values of u because the relationship between probability and feature values is non-linear.

An approach to overcoming the non-linearity problem is to work in terms of odds. If the probability of an event is p, the odds against it happening are $(1 - p)/p$ to 1. "Odds against" tend to be used when the event has a probability less than 0.5. For example, the odds against the throw of a dice providing a six is 5 to 1. This would be stated as "5 to 1 against" and means that a fair bet of \$1 should (a) provide a payoff of \$5 and return the \$1 wager if a six is thrown and (b) lead to a loss of the \$1 wager in other circumstances.

The odds in favor of an event (often used when the event has a probability greater than 0.5) are $p/(1 - p)$ to 1. Thus, the odds in favor of a dice giving 1, 2, 3, or 4 is 2 to 1. This would be stated as "2 to 1 on" and indicates that a fair bet of \$2 will return \$1 plus the \$2 wager or nothing.

The logistic regression equations show that the odds of a positive result are

$$\exp[-(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)] \text{ to 1 against}$$

or

$$\exp(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m) \text{ to 1 on}$$

This shows that the natural logarithm of odds are linear in the features:

$$\ln(\text{odds against}) = -(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)$$

$$\ln(\text{odd on}) = a + b_1X_1 + b_2X_2 + \cdots + b_mX_m$$

As in the case of linear regression, it makes sense to redefine features so that the value of the *j*th feature is $X_j - \bar{X}_j$ rather than X_j where $\bar{X}_j$ is the mean of the feature value calculated from all items in the training set. We can analyze the logarithm of odds in the same way that predictions are analyzed in linear regression. When the analog of Table 9.1 is produced, we will be able to see the contribution of each feature's deviation from its average to the logarithm of the odds.

Working with the logarithm of odds is a little artificial. Instead, we can note that the percentage effect on "odds against" of increasing the value of a feature by u is $\exp(-b_j u) - 1$. Similarly, the percentage effect on "odds on" of increasing the value of a feature by u is $\exp(b_j u) - 1$. When the percentage effects from a number of different features are multiplied together we get the total percentage effect.

Odds can be converted to probabilities:

$$\text{Probability} = \frac{1}{1 + \text{odds against}}$$

and

$$\text{Probability} = \frac{\text{odds on}}{1 + \text{odds on}}$$

We can illustrate these results with the model in Section 3.11. The probability of a loan not defaulting (which we defined as the positive result) is

$$\frac{1}{1 + \exp[-(-6.5645 + 0.1395\text{X}_1 + 0.004107\text{X}_2 - 0.001123\text{X}_3 + 0.01125X_4)]} \tag{9.3}$$

where X_1 is a categorical variable indicating home ownership, X_2 is income in ($'000s), X_3 is debt to income ratio, and X_4 is credit score. Consider someone who does not own a home (X_1 = 0) with an income of $60,000, a debt to income ratio of 5, and a credit score of 670. Substituting these values into the formula, the probability that the loan will not default is estimated as 0.7711. Consider what happens if the credit score increases by 10 from 670 to 680. From equation (9.2), the increase in this probability can be estimated as

$$0.7711 \times 0.2289 \times 0.01125 \times 10 = 0.01986$$

This is an approximate answer because probability is a non-linear function of the features. Substituting into equation (9.3) shows that increasing X_4 to 680 changes the probability to 0.7903 so the exact increase in probability given by the model is $0.7903 - 0.7711$ or 0.01925.

The odds of a positive outcome are 3.369 to 1 on because 3.369 = 0.7711/(1 − 0.7711). We know that, when the credit score increases by 10, ln(odds on) increases by $0.01125 \times 10 = 0.1125$ (from 1.2145 to 1.3270). Alternatively, the percentage increase in the odds on could be calculated as $\exp(0.01125 \times 10) - 1$ or 11.91%. Either of these results can be used to calculate the new odds on and if desired this can be converted to a probability.

The same analysis can be applied to categorical features. Suppose that, in our example, we wonder how much better the lender's position would be if the borrower owned a house. The percentage increase in the odds on that the loan will not default is, from the coefficient of X_1 in

equation (9.3), $\exp(0.1395) - 1$ or 14.97%. This means that the odds increase from 3.369 to 1 on to 3.873 to 1 on. The new probability is $3.873/(1 + 3.873)$ or 0.7948.

9.3 Black-box Models

For a black-box model, the measures needed to understand the model must usually be calculated numerically.[8] The effect on a prediction of making a change to a feature value in a particular situation can be calculated by re-running the model. For the linear model in Section 9.1, this is independent of the values of the features, but for non-linear models it depends on the feature values.

Analogously to what we did in Section 9.1, we can provide a measure of the contribution of each feature in a particular situation by calculating the change in the prediction when the feature is changed from its average value to its actual value with all other features remaining at their average values. In Table 9.1, the sum of the contributions of different features equals the difference between the actual prediction and the average prediction. As a result, the difference between the prediction and one based on average feature values is neatly divided into a number of components. For a non-linear model, calculating contributions using the approach in Table 9.1 does not have this simple property and we must use a more complicated calculation, which will be explained in Section 9.4.

A key point about black-box models is that the relationship between the prediction and a feature's value may not be linear. It may not even be monotonic. For example, a store could find that one of the features affecting its sales is temperature and that the average effect of temperature has the form shown in Figure 9.1. This shows that, when the temperature is very low or very high, the volume of sales declines.

In a particular situation, the impact of feature *j* can be determined changing feature *j* while keeping all features except feature *j* fixed. An extension of this idea is where two features are considered simultaneously so that a three-dimensional plot is obtained.

To provide an overall understanding of the role of feature *j* in a model (as opposed to its role when other features have particular values), we can calculate many predictions where

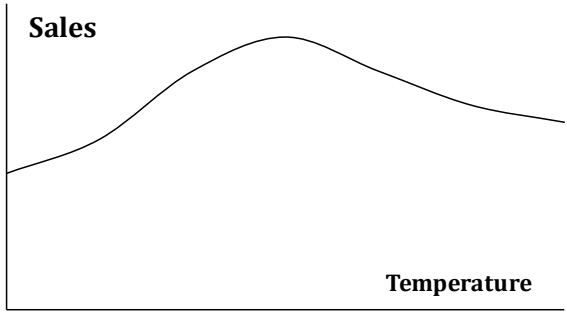
---

[8] But note that in the case of neural networks, partial derivatives can be calculated by working forward through the network and using the chain rule.

- Feature *j* has a particular value x_j.
- The values of the other feature are chosen randomly.

By averaging across all these predictions, we obtain the expected prediction conditional on feature *j* equaling x_j. By considering a number of different values for x_j we are then able to plot the expected prediction as a function of x_j. This is known as a *partial dependence plot*. (In the case of linear regression, it is a straight line.)

Figure 9.1 Possible effect of temperature on a store's sales



9.4 Shapley Values

Linear regression models have the property, illustrated in Table 9.1, that a simple calculation leads to the result that the sum of the contributions of the features to the change in a prediction equals the change. Non-linear models, as already mentioned, do not have this convenient property. However, what are known as Shapley values show that a more complicated calculation of the contributions of features leads to a result where the property does hold. Shapley values are based on the work of Lloyd Shapley in the early 1950s concerned with game theory.[9]

We can illustrate the nature of the calculations with a simple example. Suppose that there are three features. When the features have their

[9] See L. S. Shapley, "A value for *n*-person games." Rand Corporation, 1952.

average values a black-box model gives a prediction of 100. In a situation, which we will refer to as "current", the features have values that are different from average and the black-box model leads to a prediction of 140. What is the contribution of the features to the 40 increase?

To investigate this, we re-run the model to calculate the prediction for every situation where some features have their average values and some have their current values. We suppose that the results are as indicated in Table 9.2.

We next consider the sequence in which the features move from their average values to their current values. Denote XYZ by the situation where feature X changes first, then feature Y changes, and then feature Z changes. There are six possibilities in our example: 123, 132, 213, 231, 312, and 321. In the first two cases (123 and 132), feature 1 is changed from average to current while the other two features stay at their average values. From the first and fifth row of Table 9.2, the contribution of feature 1 is 110−100 = 10 in this situation. In the third case (213), feature 1 is changed after feature 2 has been changed, but before feature 3 is changed. From rows 3 and 7 of Table 9.2, the contribution of feature 1 is 137 −125 =12 in this situation. Other similar calculations are shown in Table 9.3. It can be seen that the total average contribution of the features (= 10 + 18.5 + 11.5) equals the total increase, 40, that has to be explained. This is always the case.

Table 9.2 Results from running model with different combinations of average and current values

Feature 1 Value	*Feature 2 Value*	*Feature 3 Value*	*Prediction*
Average	Average	Average	100
Average	Average	Current	120
Average	Current	Average	125
Average	Current	Current	130
Current	Average	Average	110
Current	Average	Current	128
Current	Current	Average	137
Current	Current	Current	140

Table 9.3 Contribution of feature for different sequences in which features are changed from the average value to the current value

Sequence	*Feature 1 Contribution*	*Feature 2 Contribution*	*Feature 3 Contribution*
123	10	27	3
132	10	12	18
213	12	25	3
231	10	25	5
312	8	12	20
321	10	10	20
Average	10	18.5	11.5

This calculation of the contribution of the features using Shapley values has nice properties. One is the property we have just illustrated that the sum of the contributions equals the total change. Other attractive properties are

- If a feature never changes, the prediction its contribution is zero.
- If two features are symmetrical in that they affect the prediction in the same way, they have the same contribution.
- For an ensemble model where predictions are the average of predictions given by several underlying models, the Shapley value is the average of the Shapley values for the underlying models.

As the number of features increases, the number of calculations to determine Shapley values increases quite fast making it computationally expensive to use them.[10] Shapley values can be used to explain why any two predictions are different (i.e., the benchmark does not have to be a prediction based on average feature values). However, they have a limited ability to explain the workings of the model as a whole. Also, interactions between features can cause unrealistic combinations of feature values to be considered.[11]

[10] When there are m features, there are $m!$ sequences and 2^n different contributions to calculate.

[11] All models have this problem. We explained in Section 9.1 that this happens in Table 9.1. Changing total basement (sq. ft.) without changing first floor (sq. ft.) creates an unrealistic set of feature values.

9.5 LIME

Local Interpretable Model-agnostic Explanations (LIME) are an alternative approach to explaining the predictions made by black-box models.[12] LIME tries to understand a model's prediction by creating a simpler more interpretable model that works well for values of the features that are close to those being considered.

The procedure is as follows:

- Perturb the current values of the features to create sample values for the features
- Run the black-box model to get predictions from the samples
- Train an easy-to-interpret model (e.g. linear regression or decision trees) using the samples and their predictions

Predictions in the region of interest can then be explained using the new model. The samples can be weighted according to their closeness to the current values of the features. The new model will often contain less features than the original model (e.g., because it uses Lasso).

Summary

An important aspect of machine learning is interpretability. Some models such as *k*-nearest neighbors, linear regression, and decision trees are fairly easy to interpret. Logistic regression is not quite as straightforward, but there is an analytic formula relating inputs to outputs and so any required property of the model can easily be derived. Models such as neural networks are black boxes. There is no simple way of understanding how outputs are related to inputs.

One question that can be asked of a model is "What happens if the value of feature *j* is changed with all other features remaining the same?" This is easy to answer in the case of linear regression. The weight of feature *j* times the change in the value of the feature equals the impact of the change on the prediction. This is true for both small

---

[12] See M.T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you? Explaining the predictions of any classifier." Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (2016).

and large changes and does not depend on current feature values. In the case of logistic regression, it is sometimes easier to work in terms of odds rather than probabilities when estimating the impact of feature-value changes.

In black-box models, the relationship between a feature's value and predictions can be highly non-linear. To understand the relationship for a particular prediction, an analyst can repeatedly run a model to investigate the effect of changing the value of the feature while keeping all other feature values fixed. To provide a broader understanding of the feature's effect, the values of the other features can be randomized in some way.

To understand why one set of feature values produces a different prediction from another set of feature values, one can consider the contribution of each feature separately. However, when the model is non-linear the sum of the contributions does not equal the total difference that is to be explained. Shapley values provide a way of overcoming this problem so that the overall change in a prediction is exactly allocated to the contributions of different features.

A recent idea in model interpretability is referred to as LIME. This involves understanding how a model works when features have values close to their current values. The approach is to perturb the current values and re-run the model so that a new data set describing output from the model in the region of interest is created. An interpretable model such as linear regression or decision trees is then fitted to this data set.

SHORT CONCEPT QUESTIONS

9.1 Which of the models introduced in this book are most difficult to interpret?

9.2 In what ways is a linear model simpler to interpret than a non-linear model?

9.3 How much is it worth to have an extra 5,000 square feet of back yard in Iowa?

9.4 In logistic regression, what is the equation for the sensitivity of the probability of a negative outcome to a very small change in the feature value?

9.5 Explain what is meant by "odds against" and "odds on." Why might they be useful concepts for interpreting logistic regression models?

9.6 "Interactions between features create problems when the contributions of features to the change in a prediction is calculated." Explain this statement.
9.7 Explain what a partial dependence plot is and how it is calculated.
9.8 What are the advantages of using Shapley values in model interpretability?
9.9 Explain the LIME approach to model interpretability.
9.10 How many different sequences have to be considered when Shapley values are calculated for four features?

EXERCISES

9.11 For the logistic regression model in Table 3.9, use Shapley values to calculate the contribution of each feature to the probability of a positive result for the person considered in Section 9.2 (no home, income equals $60,000, debt-to-income ratio is 5, and a credit score is 670). Use a person with average feature values as the benchmark.
9.12 Use the LIME approach to calculate a local model for the person considered in Section 9.2 (no home, income equals $60,000, debt-to-income ratio is 5, and a credit score is 670)

Chapter 10

Applications in Finance

Until a few years ago, my research was almost exclusively concerned with finance, in particular derivatives markets. It then became apparent to me that machine learning was having a bigger and bigger impact on finance and derivatives. I started to learn about machine learning and this book is one of the results of that.

This chapter will give a flavor for some of the applications of machine learning that are starting to revolutionize finance by describing two applications in detail. These applications are simplified versions of research that I have been involved in. (They involve derivatives, but readers with little knowledge of this area need not be concerned as the chapter provides the necessary background.) Data and Python code for the applications is at

www-2.rotman.utoronto.ca/~hull

The reader is encouraged to use this to explore the applications further.

The chapter concludes by summarizing a few of the many other ways that machine learning is used in finance.

10.1 Derivatives

Most financial and other transactions involve the immediate, or almost immediate, exchange of an asset for cash. Derivative transactions

are by their nature different. They involve two parties agreeing to an exchange in the future rather than immediately.

The exchange agreed to by the two parties in a derivative transaction usually involves one or more financial assets. The value of the transaction therefore depends on (or derives from) the value of these underlying financial assets. Examples of underlying financial assets that are frequently used in derivative transactions are stocks, portfolios of stocks, commodities, and currencies.

An important and popular derivative is an option. This gives one side the right (but not the obligation) to buy an asset from the other side, or sell an asset to the other side, for a certain price at a certain future time.[1] The price in the contract is known as the *strike price*.

The right to buy is termed a *call option*. An example of a call option is where Party A obtains the right to buy a certain stock for \$50 in six months from Party B. The current price of the stock might be greater than or less than \$50. The option will be exercised if the stock price in six months, S_T, is greater than \$50 for a gain equal to $S_T - 50$.[2] Party A is referred to as having bought the call option or having a long position in the call option. Party B is referred to as having sold the call option or having a short position in the call option. Party A would pay an amount upfront to Party B to acquire the option.

The right to sell is termed a *put option*. An example of a put option is where Party A obtains the right to sell a stock for \$50 in six months. (As in the case of the call option, the current price of the stock might be more or less than \$50.) The option will be exercised if the stock price in six months, S_T, is less than \$50 for a gain equal to $50 - S_T$.[3] Party A is referred to as having bought the put option or having a long position in the put option. Party B is referred to as having sold the put option or having a short position in the put option. As in the case of a call option, Party A would pay the cost of the option to Party B upfront.

Figure 10.1 shows the payoffs to the purchaser from call and put options as a function of the asset price at option maturity when the strike price is \$50. It can be seen that the payoff is fundamentally different from investing in the stock itself for the life of the option. In the case of an investment in the stock, the gain from a certain increase in the price equals the loss from the same decrease in price. In the case of an option this is

---

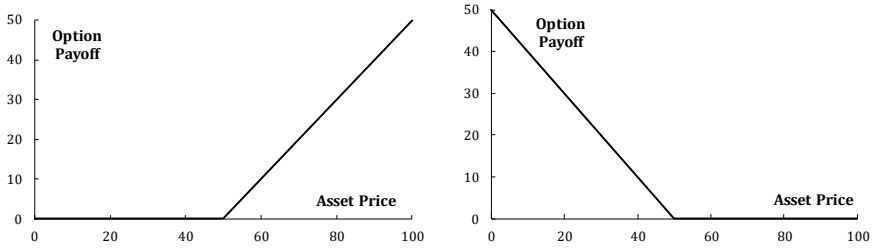
[1] The options that we will consider throughout this chapter are all what are termed *European options*. These can be exercised at only one future time.

[2] Party A could monetize this gain by buying the stock under the terms of the option and immediately selling it in the market.

[3] Party A could monetize this gain by buying the stock in the market and immediately selling it under the terms of the put option.

not so. There is a lack of symmetry. The payoff from favorable price movements during the life of the option can be very high whereas that from unfavorable movements is at worst zero. To put this another way, the purchaser of an option cannot lose more than the price paid for it while the possible gain can be substantial.

Figure 10.1 Payoff from (a) a call option to buy a stock for \$50 and (b) a put option to sell a stock for \$50



(a) (b)

When the volatility of the price of the underlying asset is high, big movements in the asset price are possible and the lack of symmetry just mentioned is valuable to the option holder. When the volatility is low, the lack symmetry is still present, but it is less valuable because big price movements are less likely.

The price that has to be paid for an option by the buyer to the seller therefore depends on an estimate of the volatility of the asset price. As the volatility increases, the price increases. This dependence of option prices on volatility is what makes the analysis of options and other derivatives more complicated (and more interesting) than that of other simpler instruments.

Define K as the strike price of an option and S as the price of the underlying asset. When $K < S$, a call option is commonly referred to as *in-the-money* because, if it were possible to exercise it immediately, there would be a positive payoff. When $K > S$, a call option is similarly referred to as *out-of-the-money*. For a put option, the reverse is true: when $K < S$, a put option is out-of-the-money and when $K > S$, it is in-the money. The extent to which an option is in- or out-of-the-money (i.e., the relative values of K and S) is referred to as the option's *moneyness*.

10.2 Delta

An important parameter in derivatives markets is *delta*. This is the sensitivity of the value of a derivatives portfolio dependent on a particular underlying asset to the price of that asset. If a small increase in the price of the underlying asset equal to ΔS causes the value the portfolio to increase by ΔP, the delta of the portfolio is $\Delta P/\Delta S$.

A portfolio with a delta of zero has the property that it is not sensitive to small changes in the price of the underlying asset and is referred to as *delta-neutral*. Traders usually try and make their portfolios delta-neutral, or close to delta-neutral, each day. They can do this by trading the underlying asset. Suppose that the delta measured for a derivatives portfolio is −4,000, indicating that a small increase, ΔS, in the price of the underlying asset will lead to a decrease in the value of the portfolio equal to $4{,}000 \times \Delta S$. Taking a long position in 4,000 units of the asset will lead to a delta-neutral portfolio. The gain (loss) on the derivatives is then offset by the loss (gain) on the new position taken.

The most famous model for valuing options is the Black–Scholes–Merton model, which we used to illustrate neural networks in Chapter 6. This is a relationship between the price of the option and the following variables.

- Asset price, S
- Strike price, K
- Risk-free interest rate, r
- Volatility, σ
- Life of the option, T
- The income expected by the market on the asset. We will assume that the underlying asset provides a constant yield (i.e., income as a percent of its price is constant) and will denote this yield by q

The Black–Scholes–Merton model assumes that the rate of return in a short period of time, Δt, is normally distributed with a mean of $\mu\Delta t$ and a standard deviation of $\sigma\sqrt{\Delta t}$. (Magically, as can be seen from equation (6.3), the mean return does not enter into the equation for the price of the option.) The delta of call and put options are

$$\text{Delta (call)} = e^{-qT}N(d_1)$$

$$\text{Delta (put)} = e^{-qT}[N(d_1) - 1]$$

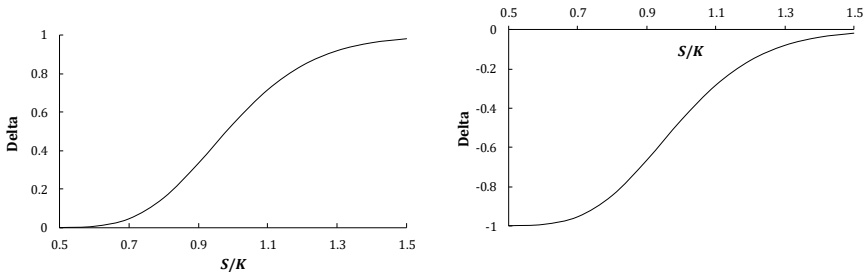
where

$$d_1 = \frac{\ln(S/K) + (r - q + \sigma^2/2)T}{\sigma\sqrt{T}}$$

The delta of a long position in a call option and a put option as a function of S/K is shown in Figure 10.2. Suppose for simplicity that $q = 0$. For a call option, delta is close to zero when it is deeply out of the money (S/K considerably less than 1) and close to one when it is deeply in the money (S/K considerably greater than one). For a put option, the pattern of deltas is similar except that, instead of ranging from 0 to 1, they range from −1 to 0

In addition to using delta for hedging, traders also use delta to measure moneyness.[4] We will do this in what follows. A call option with a delta of 0.5 is referred to as at-the-money. When delta is greater than 0.5, it is in-the-money and, when delta is less than 0.5, it is out-of-the-money. Similarly, a put option with a delta of −0.5 is referred to as at-the-money, one with a delta less than −0.5 is in-the-money, and one with a delta greater than −0.5 is out-of-the-money.

Figure 10.2 Variation of delta with S/K when $q = 0$ for (a) a long position in a call option and (b) a long position in a put option



(a) (b)

10.3 Volatility Surfaces

Of the six variables listed in the previous section that determine option prices in the Black–Scholes model, all except σ are known. (The income expected by the market on an asset can be estimated from futures

[4] As mentioned earlier, a common definition of moneyness simply reflects whether $S > K$ or $S < K$. Delta is a more sophisticated measure preferred by traders.

or forward contracts.) Option traders therefore play a game where they observe an option price in the market and then calculate from the Black–Scholes–Merton model the value of σ that is consistent with the price. This value of σ is known as the option's *implied volatility*.[5] Very often option prices are quoted in terms of their implied volatilities.[6] The price of the option must then be obtained from the implied volatility by substituting it into the Black–Scholes–Merton model, together with the other known parameters.

The Black–Scholes–Merton model assumes that the volatility, σ, is constant. From this, it follows that, if market participants used the Black-Scholes–Merton model with the same volatility when pricing all options on an asset, the implied volatilities calculated from the options would be the same and equal to the volatility used. In practice, the Black–Scholes –Merton model does not provide a perfect description of how traders price options. Nevertheless, implied volatilities continue to be calculated and quoted by market participants. At any given time, each option trading in the market has a price, and therefore an implied volatility, associated with it.

An important activity for traders is keeping track of implied volatilities. The implied volatility of an option at any given time is a function of two variables:

- its moneyness (which, as explained earlier, is measured as delta)
- its time to maturity, T.

The three-dimensional function relating implied volatility to delta and T is referred to as the *volatility surface*. The volatility surface, estimated from all options on the S&P 500 on two different dates, is shown in Figure 10.3.

10.4 Understanding Volatility Surface Movements

As illustrated in Figure 10.3, the volatility surface is highly non-linear. The change in the volatility surface from one day to the next is also highly non-linear. Understanding how the volatility surface moves is important for a number of reasons:

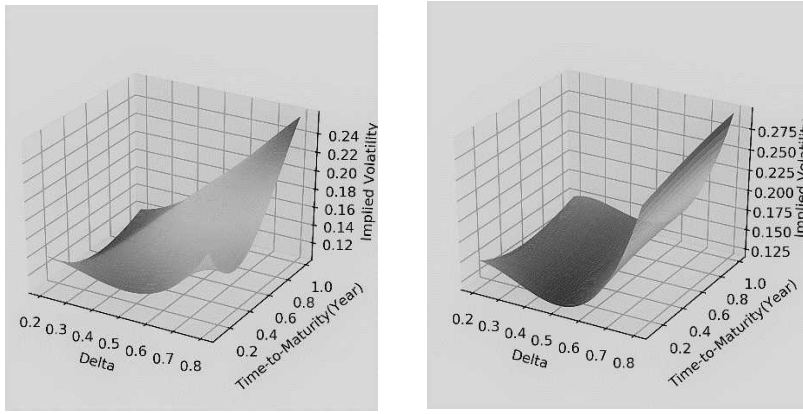
---

[5] The implied volatility for a call options with a certain strike price and maturity is the same as that for a put option with the same strike price and time to maturity.
[6] The key advantage of doing this is that, when the asset price changes, the implied volatility often stays the same. When quotes are in terms of implied volatility they are therefore more stable.

- It can help a trader hedge her exposure.[7]
- It can help a quant determine a stochastic volatility model reflecting how options are priced in the market.
- It can help a trader adjust implied volatilities in a market where asset prices are changing fast.

A neural network is a natural tool for using empirical data to model volatility surface movements.

Figure 10.3 Volatility surface observed for options on the S&P 500 on (a) January 31, 2019 and (b) June 25, 2019



(a) (b)

When the price of an asset declines, all implied volatilities calculated from options on the asset tend to increase, and vice versa. However, the implied volatilities do not all change by the same amount. This leads to many variations in the pattern of implied volatilities.

Interestingly, two competing theories about the phenomenon that implied volatilities tend to be negatively correlated with asset returns have been proposed by researchers:

[7] As the application in this section shows, when the asset price increases, volatility tends to decrease, and vice versa. A trader might want to use a "volatility-adjusted delta" for hedging that takes into account how volatility is expected to change when the price of the underlying asset changes.

- One theory, applicable to stocks and stock indices, is that when the asset price declines (increases), the impact of debt in the capital structure becomes more (less) pronounced and volatility increases (decreases).
- Another theory, known as the "volatility feedback effect hypothesis" argues that when volatility increases (decreases), investors in the asset require a higher (lower) return as compensation for risk. As a result the price of the asset declines.

In the first theory, the causality is from asset price changes to volatility. In the second theory, it is the other way around. On balance, the empirical evidence seems to favor the second theory, but for our purposes the reason for the relationship does not matter. We are merely interested in using data to quantify the relationship.

Our data consists of call options on the S&P 500 between 2014 and 2019. To keep the size of the data set manageable, we randomly sample 100 options per day, recording quotes for each option on both the day it is sampled and the next day.[8]

The neural network has the structure shown in Figure 6.3. There are three features:

- The percentage change in the S&P 500 from one day to the next
- The time to maturity
- The delta of the option

The target is the change in the implied volatility. The objective is to minimize the mean squared error between the predicted change in the implied volatility and the actual change. The results we will present here use three hidden layers and 20 nodes per layer.

Table 10.1 shows a sample of the data. There were 125,700 observations in total. These were split randomly into a training set (75,420 observations or 60% of total), validation set (25,140 observations or 20% of total), and test set (25,140 observations or 20% of total). As mentioned in Chapter 6, the performance of a neural network is improved when the inputs are scaled. We used *Z*-score scaling (with the mean and standard deviation taken from the training set). Table 10.2 shows the data from Table 10.1 after this scaling has been applied.

---

[8] The analysis presented here is a simplified version of that carried out by J.Cao, J. Chen, and J. Hull, "A neural network approach to understanding implied volatility movements," forthcoming, *Quantitative Finance*, available at ssrn 3288067. Their analysis used data between 2010 and 2017 and was based on 2.07 million observations on 53,653 call options.

In constructing a machine learning model, it is always useful to have a simpler model as a benchmark. In this case, we use the following model

$$\text{Expected change in implied volatility} = R\frac{a + b\delta + c\delta^2}{\sqrt{T}} \qquad (10.1)$$

where R is the return on the asset (= change in price divided by initial price), T is the option's time to maturity, δ is the option's moneyness (measured as delta) and a, b, and c are constants. This model was suggested by Hull and White (2017) and is quite popular with practitioners.[9] The a, b, and c can be estimated by regressing implied volatility changes against $R/\sqrt{T}$, $R\delta/\sqrt{T}$, and $R\delta^2/\sqrt{T}$.

Table 10.1 Sample of the original data. The return on the S&P 500 and the change in the implied volatility are between the day specified and the next business day. (See Excel file)

Date	*Return (%) on S&P 500*	*Maturity (years)*	*Delta*	*Change in implied volatility (bps)*
Jan 28, 2015	0.95	0.608	0.198	−31.1
Sept 8, 2017	1.08	0.080	0.752	−54.9
Jan 24, 2018	0.06	0.956	0.580	−1.6
Jun 24, 2019	−0.95	2.896	0.828	40.1

Table 10.2 Data in Table 10.1 after using Z-score scaling for the inputs. (See Excel file)

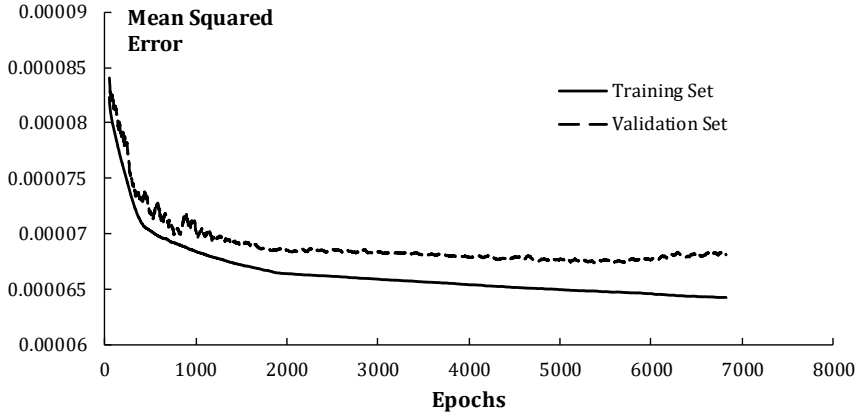
Date	*Return (%) on S&P 500*	*Maturity (years)*	*Delta*	*Change in implied volatility (bps)*
Jan 28, 2015	1.091	−0.102	−1.539	−31.1
Sept 8, 2017	1.247	−0.695	0.445	−54.9
Jan 24, 2018	0.027	0.289	−0.171	−1.6
Jun 24, 2019	−1.176	2.468	0.716	40.1

Figure 10.4 shows results obtained by the neural network for the training set and the validation set. (The mean squared errors have been smoothed in producing Figure 10.4 by averaging them over 50 successive epochs.)

[9] See J. Hull and A. White, "Optimal delta hedging for options," *Journal of Banking and Finance*, Sept 2017: 180-190.

Following the approach explained in Chapter 6, we continue training until there is no improvement in the results from the validation set. In our example this happens after the 5,826th epoch. The test set results indicated a modest 14% improvement over the model in equation (10.1).

Figure 10.4 Mean squared errors as the number of epochs is increased for the training set and the validation set



Once a first model has been implemented, it is natural to look for additional features that can improve results further. In this case, the VIX index, observed on day t to predict changes between day t and t+1, is found to produce a considerable improvement.[10] (By trying Exercise 10.12, readers can verify this for themselves). We can deduce from this that movements in the implied volatility surface tend to be different in high and low volatility environments.

10.5 Using Reinforcement Learning for Hedging

In Section 10.2, we explained how delta is used for hedging a portfolio of derivatives dependent on an underlying asset. For the purposes of our next application, we suppose that we do not know how to calculate delta and want to use reinforcement learning to calculate the optimal hedging strategy for a short position in an option.

[10] See results in J.Cao, J. Chen, and J. Hull, "A neural network approach to under-standing implied volatility movements," forthcoming, *Quantitative Finance*, available at ssrn 3288067.

Suppose we want to hedge the sale of 10 call options where $S = 100$, $K = 100$, $\mu = 0$, $r = 0$, $q = 0$, $\sigma = 20\%$, and $T = 10$ days or 0.04 years.[11] The hedge is accomplished by owning 0, 1, 2,..., or 10 shares (i.e., there are 11 possible positions). If the hedger knew that the option would be exercised because the final price of the stock was certain to be greater than 100, it would be optimal for her to buy 10 shares. Similarly, if the hedger knew that the option would be not be exercised because the final price of the stock was certain to be less than 100, it would be optimal for the hedger to hold no shares. In practice of course, the outcome is uncertain and, when each hedging decision is made, the hedger must choose a position between 0 and 10 shares.[12]

We assume that the behavior of the stock price is that underlying Black–Scholes–Merton model. As mentioned earlier, this means that the rate of return in the stock price in a short period of time, Δt, is normally distributed with mean $\mu\Delta t$ and standard deviation $\sigma\sqrt{\Delta t}$. The stock price calculated from the model is rounded to the nearest integer.

We assume that the hedge position (i.e., number of shares owned) can be changed once a day so that a total of 10 decisions have to be made. The states for the reinforcement learning algorithm at the time hedging decision is made on a day are defined by

- The number of shares currently held (i.e., the position taken on the previous day)
- The stock price

As mentioned there are 11 possible values for the number of shares currently held. A total of 15 possible price states are considered:

≤93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, ≥107

To simplify matters we assume that at most five shares can be bought and five shares can be sold. For example, if the current holding is 7, possible actions (i.e., changes in the position) are −5, −4, −3, −2, −1, 0, +1, +2, and +3. Similarly, if the current holding is 2, possible actions are −2, −1, 0, +1, +2, +3, +4, and +5.

The objective function is to minimize the variance of the hedging cost. The expected cost in each period (regardless of the decision taken) can be assumed to be zero. The changes in the value of the hedged position

[11] We assume 250 trading days in a year.

[12] In practice, an option contract is to buy or sell 100 shares and the shares themselves are bought or sold in multiples of 100. We assume that only 11 different positions can be taken to simplify the analysis.

on successive days are assumed to be independent. Because the interest rate is zero, this means that the objective is to minimize

$$\sum_i H_i^2$$

where H_i is change in the value of the hedger's position during the ith day and the summation is taken over all periods from the current one onward. If N_i is the number of shares held at the beginning of the ith day, S_i is the stock price at the beginning of the ith day, and c_i is the value of the call option at the beginning of the ith day (obtained from the Black-Scholes–Merton equation),

$$H_i = N_i(S_{i+1} - S_i) - 10(c_{i+1} - c_i)$$

Our Python implementation is at

www-2.rotman.utoronto.ca/~hull

We generated three million paths for the stock price for the training set. (The fact that three million paths are necessary for such a small problem emphasizes the point made in Chapter 7 that reinforcement learning is "data hungry.") We then used a further 100,000 paths for the test set. Initially the probability of exploration was 100%. The probability of exploration decreases as the algorithm proceeds so that the probability of exploration during a trial is 0.999999 times its value at the preceding trial. The minimum value for the probability of exploration was 5%.

We find that the reinforcement learning model tracks delta hedging reasonably well. The mean absolute difference between the position taken by the reinforcement learning algorithm and that taken by delta hedging across all days was 0.32 for the test set. The standard deviation of the cost of writing the option and hedging it was about 9% higher when the algorithm was used than when delta hedging was used.

10.6 Extensions

Two criticisms might be levelled at the analysis just presented. First, the problem does not need to be solved with reinforcement learning because the equation for delta under Black–Scholes–Merton assumptions is well known (see Section 10.2). Second, the problem is not really a multi-period one. The delta each day can be calculated independently of the delta on each other day.

However, there are extensions of the analysis that make it relevant. In practice, a trader faces bid–ask spreads, i.e., the ask price at which she

can buy an instrument is generally higher than the bid price at which she can sell the instrument. If bid–ask spreads are not negligible, the trader wants to use a strategy where the cost of buying and selling is weighed against the reduction in risk. The objective function can then be

$$\sum_i (C_i + \alpha H_i^2)$$

where C_i is the transaction cost (arising from the bid–ask spread) incurred in period i and H_i is as before the change in the value of the hedger's position on day i with the summation being taken over all hedging periods from the current one onward. The parameter α defines the trade-off between expected costs and variance of costs.

Delta hedging is not optimal when transaction costs are considered and the problem is then a genuine multi-period problem that requires reinforcement learning.[13] The code which accompanies the analysis in Section 10.5 can be extended to consider this problem (see Exercise 10.14).

Even when trading the underlying asset entails negligible bid–ask spreads, a trader may want to use reinforcement learning for hedging volatility risk. The exposure to volatility is measured by what is known as vega. Changing vega requires taking a position in another derivative dependent on the same underlying asset. Trading derivatives almost invariably involves non-negligible bid–ask spreads, making the use of reinforcement learning to derive a hedging strategy particularly relevant.

The reinforcement learning algorithm can be used with a number of different stochastic processes (For vega hedging a process where the volatility is stochastic is obviously necessary). In practice, there is some uncertainty about the process that the asset will follow. One approach suggested by Cao et al (2019) is to use a mixture of processes (with each one being equally likely to be chosen for creating a path) and train the model on the mixture.[14] The hedging scheme should then work reasonably well on all the processes.

---

[13] This problem is considered by J. Cao, J. Chen, J. Hull and Z. Poulos, "Deep hedging of derivatives using reinforcement learning," Working paper, 2019, available at ssrn 3514586; H. Buehler, L. Gonon, J. Teichmann, B. Wood, "Deep hedging," 2019, available at ssrn: 3120710; and P.N. Kolm and G. Ritter, "Dynamic replication and hedging: a reinforcement learning approach," *Journal of Financial Data Science*, Winter 2019: 159-171.

[14] See by J. Cao, J. Chen, J. Hull and Z. Poulos, "Deep hedging of derivatives using reinforcement learning," Working paper, 2019, available at ssrn 3514586.

10.7 Other Finance Applications

There are many other applications of machine learning in finance. For example:

- Machine learning is used extensively in investing. As described in Chapter 8, sentiment analysis can be used to determine the influence of opinions in blogs, tweets, news articles, etc. on asset returns. As mentioned, Renaissance Technologies, a secretive hedge fund run by mathematicians, has used machine learning for investing very profitably for many years. Many other hedge funds now use machine learning to make portfolio management decisions, sometimes without any human intervention.[15]
- Machine learning is sometimes used by private equity companies. Data on start ups, can be used to determine the features that predict success (e.g., the age of the company, sales growth, the characteristics of the management team, and so on).
- Natural language processing is sometimes used by human resource professionals for dealing with the large number of resumes that are received.
- Handling loan applications has been a very successful application of machine learning. We have illustrated how this can be done with data from Lending Club earlier in this book. Banks throughout the world are moving routine lending decisions from human loan officers to algorithms such as those we have illustrated. (This is in spite of the fact that humans can do some things algorithms cannot such as meet with the borrower and assess the borrower's character.) New accounting standards such as IFRS 9 require that the valuation of loans on the balance sheet reflect expected losses. This provides an extra incentive for banks to come up with robust methods for estimating probabilities of default and recovery rates.
- Identifying fraudulent transactions, money laundering, and misconduct by employees or agents is an important activity for financial institutions. This is proving to be an area where machines can outperform humans.
- As indicated in Chapter 7, reinforcement learning can be used for order execution. When a large buy or sell order is to be plac-

[15] For a discussion of this, see: M. Lopez de Prado, *Advances in machine learning*, John Wiley and Sons, 2018 and S. Gu, B. Kelly, and D. Xiu, "Empirical asset pricing and machine learning," 2019, available at dachxiu.chicagobooth.edu/download/ML.pdf

ed, a trader has to find a balance between (a) placing a large order all at once and potentially moving the market and (b) spreading the required trade over a period of time and risking adverse market moves.

- Many transactions require collateral and often there are alternative assets that can be used for this purpose (e.g., Treasury bonds, Treasury bills, and corporate bonds). Machine learning can assist in making optimal collateral decisions.
- Some derivatives have to be valued using Monte Carlo simulation or other quite slow numerical procedures. This creates problems when scenarios and other analyses are carried out to investigate the risks in a portfolio. As explained in Chapter 6, one approach is to create a neural network to replicate the value of the derivative obtained from the numerical procedure. Valuing the derivative then involves working forward through the network, which is very fast. There is a lot of computational effort involved in generating the training set, but this can be a good investment.[16]

Summary

There are many applications of machine learning in finance. Many of these applications involve derivatives. In Chapter 6, we saw how a neural network can replicate a method for pricing derivatives. Once it has been developed, the neural network provides very fast pricing. This can be attractive when traditional ways of pricing the derivative are computationally very time consuming.

In this chapter, we have looked at two applications of machine learning in some detail. One involves trying to understand volatility movements. The relationship between movements in an option's implied volatility and the price of the underlying asset is highly non-linear. Neural networks can be used in conjunction with historical data to quantify this relationship.

The other application involves hedging. When there are trading costs, hedging involves complex multi-period decision making and is ideally suited to reinforcement learning. In this case, the only viable approach is to use a model to generate training data because historical data is not

[16] For a fuller description, see R. Ferguson and A. Green, "Deeply learning derivatives," October 2018, available at ssrn: 3244821.

available in the volume required. The application in this chapter illustrates how several million paths for the underlying asset price can be used to determine a viable hedging strategy.

There are many other applications of machine learning in finance involving the algorithms covered in this book. We have used lending decisions to illustrate classification in Chapter 3, 4, and 5. Other applications include those in areas such as fraud detection, order execution, and investment.

SHORT CONCEPT QUESTIONS

10.1 What is the difference between a call option and a put option?
10.2 Why does the price of a derivative depend on volatility?
10.3 What is meant by the moneyness of an option? How is it measured?
10.4 What are the six variables that the price of an option depends on in the Black-Scholes model?
10.5 Explain what delta-neutrality means.
10.6 What is an implied volatility?
10.7 What is a volatility surface?
10.8 How does the volatility surface usually move when the price of an asset changes? Do these movements increase or reduce a derivative trader's exposure to movements in the price of the underlying asset when a call option has been sold?
10.9 Under what circumstances does reinforcement learning lead to an improvement over delta hedging?
10.10 Why is reinforcement learning likely to be particularly useful for hedging against movements in volatility?

EXERCISES

10.11 In the application in Section 10.4, use the Python code available at
www-2.rotman.utoronto.ca/~hull
to test the effect of:
(i) Changing the number of hidden layers from three to (a) one and (b) five.
(ii) Changing the number of nodes per layer from 20 to (a) 10 and (b) 40.
(iii) Using a different activation function such as relu.
(iv) Using min-max scaling (for this test, a useful resource is Sklearn's MinMaxScaler).

10.12 Use the Python code at

www-2.rotman.utoronto.ca/~hull

to repeat the analysis in Section 10.4 adding the value of the VIX index (observed on the first of the two days that the implied volatility and S&P 500 are observed) as a feature. Values for the VIX index can be downloaded from the Yahoo Finance website.

10.13 Use the Python code at

www-2.rotman.utoronto.ca/~hull

to carry out the analysis in Section 10.5 for a 20-day option and compare your results with those for a 10-day option.

10.14 Use the Python code at

www-2.rotman.utoronto.ca/~hull

to extend the application in Section 10.5 along the lines suggested in Section 10.6. Assume that the cost of trading is 1% of the value of what is traded and that the α parameter defining the trade off between the mean cost and variance of cost is 0.15. Compare your results with those from delta hedging.

Chapter 11

Issues for Society

Computers have been used to automate tasks such as record keeping and sending out invoices for many years, and for the most part society has benefited from this. But it is important to emphasize that the innovations we have talked about in this book involve more than just the automation of tasks. They allow machines to learn. Their aim is to allow machines to make decisions and interact with the environment similarly to the way human beings do. Indeed, in many cases the aim is to train machines so that they improve on the way human beings carry out certain tasks.

We have mentioned the success of Google's AlphaGo in beating the world champion Go player, Ke Jie. Go is a very complex game. It has too many moves for the computer to calculate all the possibilities. AlphaGo used a deep learning strategy to approximate the way the best human players think about their moves and then improve on it. The key point here is that AlphaGo's programmers did not teach AlphaGo how to play Go. They taught it to learn how to play Go.

Teaching machines to use data to learn and behave intelligently raises a number of difficult issues for society. Who owns the data used by machine learning algorithms? What biases are present in machine learning algorithms? Can machines be taught to distinguish right from wrong? Should the algorithms underlying machine learning be more transparent? What are the implications of humans no longer being the

most intelligent entities on earth? This chapter considers these questions.

11.1 Data Privacy

Issues associated with data privacy have received a great deal of publicity, partly as a result of the activities of Cambridge Analytica. This company worked for both Donald Trump's 2016 presidential campaign and for an organization campaigning for the U.K. to leave the European Union. It managed to acquire and use personal data on millions of Facebook users without obtaining permission from them. The data was detailed enough for Cambridge Analytica to create profiles and determine what kind of advertisements or other actions would be most effective in promoting the interests of the organizations that had hired it.

Many governments are concerned about issues concerned with data privacy. The European Union has been particularly proactive and passed the General Data Protection Regulation (GDPR) which came into force in May 2018.[1] It recognizes that data is valuable and includes in its requirements the following:

- A person must provide consent to a company before the company can use the person's data for other than the purpose for which it was collected.
- If there is a data breach, notifications to everyone affected are mandatory within 72 hours.
- Citizens have a "right to explanation" about the application of algorithms to their data.
- Data must be safely handled across borders.
- Companies must appoint a data protection officer.

Fines for non-compliance with GDPR can be as high as 20 million euros or 4% of a company's global revenue. It is likely that other governments will pass similar legislation to GDPR in the future. Interestingly, it is not just governments that are voicing concerns about the need to regulate the way data is used by companies. Mark Zuckerberg, Facebook's CEO, considers that rules are needed to govern the internet and has expressed support for GDPR.[2]

[1] See https://gdpr-info.eu/

[2] Zuckerberg's views were outlined in a *Washington Post* article on March 30, 2019.

11.2 Biases

Human beings exhibit biases. Some are risk averse; others are risk takers. Some are naturally caring; others are insensitive. It might be thought that one advantage of machines is that they take logical decisions and are not subject to biases at all. Unfortunately, this is not the case. Machine learning algorithms exhibit many biases.

One bias is concerned with the data that has been collected. It might not be representative. A classic example here (from a time well before the advent of machine learning) is an attempt by *Literary Digest* to predict the result of the United States presidential election in 1936. The magazine polled ten million people (a huge sample) and received 2.4 million responses. It predicted that Landon (a republican) would beat Roosevelt (a democrat) by 57.1% to 42.9%. In fact, Roosevelt won. What went wrong? The answer is that *Literary Digest* used a biased sample consisting of *Literary Digest* readers, telephone users, and those with car registrations. These were predominantly republican supporters.[3] More recently, we can point to examples where facial recognition software was trained largely on images of white people and therefore did not recognize other races well, resulting in misidentifications by police forces using the software.[4]

There is a natural tendency of machine learning to use readily available data and to be biased in favor of existing practices. We encountered this in the data used in Chapters 3, 4, and 5 for classifying loans. The data available for making lending decisions in the future is likely to be the data on loans that were actually made in the past. It would be nice to know how the loans that were not made in the past would have worked out, but this data by its nature is not available. Amazon experienced a similar bias when developing recruiting software.[5] Its existing recruits were predominantly male and this led to the software being

---

https://www.washingtonpost.com/opinions/mark-zuckerberg-the-internet-needs-new-rules-lets-start-in-these-four-areas/2019/03/29/9e6f0504-521a-11e9-a3f7-78b7525a8d5f_story.html?noredirect=on&utm_term=.2365e1f19e4e

[3] See P. Squire, "Why the 1936 *Literary Digest* Poll Failed," *The Public Opinion Quarterly*, 52, 1 (Spring 1988): 125–133.

[4] See R. McCullom, 2017, "Facial Recognition Software is Both Biased and Understudied," at
https://undark.org/article/facial-recognition-technology-biased-understudied/

[5] For an account of this, see: https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G.

biased against women. As a result, its use was discontinued.

Choosing the features that will be considered in a machine learning exercise is a key task. In most cases, it is clearly unacceptable to use features such as race, gender, or religious affiliation. But data scientists also have to be careful not to include other features that are highly correlated with these sensitive features. For example, if a particular neighborhood has a high proportion of black residents, using "neighborhood of residence" as a feature when developing an algorithm for loan decisions may lead to racial biases.

There are many other ways in which an analyst can (consciously or unconsciously) exhibit biases when developing a machine learning algorithm. For example, the way in which data is cleaned, the choice of models, and the way the results from an algorithm are interpreted and used can be subject to biases.

11.3 Ethics

Machine learning raises many ethical considerations. Some people feel that China has gone too far with its Social Credit System, which is intended to standardize the way citizens are assessed.

Should machine learning be used in warfare? It is perhaps inevitable that it will be. Google canceled Project Maven, which was a collaboration with the U.S. Department of Defense to improve drone strike targeting, after thousands of Google employees signed an open letter condemning the project. However, the U.S. and other nations continue to research how AI can be used for military purposes.

Can machine learning algorithms be programmed to behave in a morally responsible and ethical way? One idea here is to create a new machine learning algorithm and provide it with a large amount of data labeled as "ethical" or "non-ethical" so that it learns to identify non-ethical data. When new data arrives for a particular project, the algorithm is used to decide whether or not it is ethically appropriate to use the data. The thinking here is that if a human being can learn ethical behavior so can a machine. (Indeed, some have argued that machines can learn to be more ethical than humans.)

An interesting ethical dilemma arises in connection with driverless cars. If an accident is unavoidable, what decision should be taken? How should an algorithm choose between killing a senior citizen and younger person? How should it choose between killing a jaywalker and someone who is obeying the rules for crossing roads? How should it choose between hitting a cyclist wearing a helmet and one who is not? Dilem-

mas such as these, which involve choosing who lives and who dies, are sometimes referred to as the "trolley problem."[6]

The interaction of human beings with machine learning technologies can sometimes lead to unexpected results with inappropriate and unethical behavior being learned. In March 2016, Microsoft released Tay (short for "thinking about you"), which was designed to learn by interacting with human beings on Twitter so that it would mimic the language patterns of a 19-year-old American girl. Some Twitter users began tweeting politically incorrect phrases. Tay learned from these and as a result sent racist and sexually charged messages to other Twitter users. Microsoft shut down the service just 16 hours after it was released.

11.4 Transparency

In recent years a lot of progress has been made in making machine learning algorithms more transparent. We discussed this in Chapter 9. When making predictions, it is important to develop ways of making the results of machine learning algorithms accessible to those who are affected by the results. It is also important for companies to understand the algorithms they use so that they can be confident that decisions are being made in a sensible way. There is always a risk that algorithms appear to be making intelligent decisions when they are actually taking advantage of obscure correlations. We gave two examples of this in Chapter 9. The German horse Hans appeared intelligent because there was a correlation between the correct answer and the expressions on the questioner's face as the horse stomped its foot. Software might distinguish polar bears and dogs because of the background (ice or grass/trees), not to the images of the animals themselves.

11.5 Adversarial Machine Learning

Adversarial machine learning refers to the possibility of a machine learning algorithm being attacked with data designed to fool it. Arguably it is easier to fool a machine than a human being! A simple example of this is an individual who understands how a spam filter works and designs an email to get past it. Spoofing in algorithmic trading is a form of adversarial machine learning. A spoofer attempts to (illegally) ma-

[6] The original trolley problem was a thought experiment in ethics concerning a runaway trolley which will either kill five people or, if a lever is pulled, kill one person.

nipulate the market by feeding it with buy or sell orders and canceling before execution. Another example of adversarial machine learning could be a malevolent individual who targets driverless cars, placing a sign beside a road that will confuse the car's algorithm and lead to accidents.

One approach to limiting adversarial machine learning is to generate examples of it and then train the machine not to be fooled by them. However, it seems likely that humans will have to monitor machine learning algorithms for some time to come to ensure that the algorithms are not being manipulated. The dangers of adversarial machine learning reinforce the points we have already made that machine learning algorithms should not be black boxes without any interpretation. Transparency and interpretability of the output is important.

11.6 Legal Issues

We can expect machine learning algorithms to give rise to many issues that have not previously been considered by the legal system. We have already mentioned issues concerned with the ownership and use of data. As it becomes more evident that data is a valuable commodity, it is likely that class action lawsuits concerned with the misuse of data will become common. We have explained how machine learning algorithms can exhibit biases. We are likely to see more class action lawsuits from groups that, rightly or wrongly, feel that an algorithm is biased against them.

Driverless cars are likely to become an important mode of transportation in the near future. If a driverless car hits a pedestrian, who is the guilty party? It could be any of

- the person who programmed the car's algorithm
- the manufacturer of the car
- the owner of the car

Contract law may have to be modified because in the future many contracts are likely to be from one machine to another (with both machines using learning algorithms). What if there is a dispute? Could there be a challenge in a court of law concerning whether a machine has the authority to execute a contract?

It is not inconceivable that in the future machines will be given rights (just as companies have rights today). Consider the situation where a great deal of experience and intelligence (far exceeding that of

any human being) is stored in a certain machine. Should a human being be allowed to shut down the machine so that the experience and intelligence is lost?

11.7 Man vs. Machine

Human progress has been marked by a number of industrial revolutions:

1. Steam and water power (1760–1840)
2. Electricity and mass production (1840–1920)
3. Computers and digital technology (1950–2000)
4. Artificial intelligence (2000–present)

There can be no doubt that the first three industrial revolutions have brought huge benefits to society. The benefits were not always realized immediately but they have eventually produced big improvements in our quality of life. At various times there were concerns that jobs traditionally carried out by humans would be moved to machines and that unemployment would result. There were upheavals in society, but they did not lead to permanent unemployment. Some jobs were lost during the first three industrial revolutions, but others were created. For example, the first industrial revolution led to people leaving rural lifestyles to work in factories. The second industrial revolution changed the nature of the work done in factories with the introduction of assembly lines. The third industrial revolution led to more jobs involving the use of computers. The impact of the fourth industrial revolution is not yet clear.

It is worth noting that the third industrial revolution did not require all employees to become computer programmers. It did require people in many jobs to learn how to use computers and work with software. We can expect the fourth industrial revolution to be similar to the third in that many individuals will have to learn new skills related to the use of artificial intelligence.

We are now reaching the stage where machine learning algorithms can make many routine decisions as well as, if not better than, human beings. But the key word here is "routine." The nature of the decision and the environment must be similar to that in the past. If the decision is non-standard or the environment has changed so that past data is not relevant, we cannot expect a machine learning algorithm to make good decisions. Driverless cars provide an example here. If we changed the

rules of the road (perhaps on how cars can make right or left turns), it would be very dangerous to rely on a driverless car that had been trained using the old rules.

A key task for human beings is likely to be managing large data sets and monitoring machine learning algorithms to ensure that decisions are not made on the basis of inappropriate data. Just as the third industrial revolution did not require everyone to become a computer programmer, the fourth industrial revolution will not require everyone to become a data scientist. However, for many jobs it will be important to understand the language of data science and what data scientists do. Today, many jobs involve using programs developed by others for carrying out various tasks. In the future, they may involve monitoring the operation of machine learning algorithms that have been developed by others.

A human plus a trained machine is likely to be more effective than a human or a machine on its own for some time to come. However, we should not underestimate future advances in machine learning. Eventually machines will be smarter than human beings in almost every respect. A continuing challenge for the human race is likely to be how to partner with machines in a way that benefits rather than destroys society.

Answers to End of Chapter Questions

Chapter 1

1.1 Machine learning is a branch of artificial intelligence where intelligence is created by learning from large data sets.

1.2 One type of prediction is concerned with estimating the value of a continuous variable. The other is concerned with classification.

1.3 Unsupervised learning is concerned with identifying patterns (clusters) in data.

1.4 Reinforcement learning is concerned with situations where a sequence of decisions has to be made in a changing environment.

1.5 Semi-supervised learning is concerned with making predictions where some of the available data have values for the target and some do not.

1.6 A validation set is used. If the answers given by the validation set start to get worse as model complexity is increased there is over-fitting.

1.7 The validation set is used to compare models so that one that has good accuracy and generalizes well can be chosen. The test set is held back to provide a final test of the accuracy of the chosen model.

1.8 A categorical feature is a non-numerical feature where data is assigned to one of a number of categories.

1.9 The bias-variance trade-off is the trade-off between (a) under-fitting and missing key aspects of the relationship, and (b) over-fitting so that idiosyncrasies in the training data are picked up. The

linear model is under-fitting and therefore gives a bias error. The fifth order-polynomial model is over-fitting and therefore gives a variance error.

1.10 Data cleaning can involve (a) correcting for inconsistent recording, (b) removing observations that are not relevant, (c) removing duplicate observations, (d) dealing with outliers, and (e) dealing with missing data.

1.11 Bayes' theorem deals with the situation where we know the probability of X conditional on Y and we want the probability of Y conditional on X.

1.12 For a polynomial of degree three, the standard deviation of the error for the training set and the validation set are 31,989 and 35,588, respectively. For a polynomial of degree 4, the standard deviation of the error for the training set and the validation set are 21,824 and 37,427, respectively. As the degree of the polynomial increases, we obtain more accuracy for the training set but there is a bigger difference between the performance of the model for the training set and the validation set.

1.13 Using Bayes' theorem

$$P(\text{Spam}|\text{Word}) = \frac{P(\text{Word}|\text{Spam})P(\text{Spam})}{P(\text{Word})}$$

$$= \frac{0.4 \times 0.25}{0.125} = 0.8$$

There is an 80% chance that an email containing the word is spam.

Chapter 2

2.1 Feature scaling is necessary in unsupervised learning to ensure that features are treated as being equally important. In the Z-score method, each feature is scaled so that it has a mean of zero and a standard deviation of one. In min-max scaling each feature is scaled so that the lowest value is zero and the highest is one. Min-max scaling does not work well when there are outliers because the rest of the scaled values are then close together. But it may work better than the Z-score method when features have been measured on different scales with lower and upper bounds.

2.2 The distance is $\sqrt{(6-2)^2 + (8-3)^2 + (7-4)^2} = 7.07$.

2.3 The center is obtained by averaging feature values. It is the point that has values 4, 5.5, and 5.5 for the three features.

2.4 We choose k points as cluster centers, assign observations to the nearest cluster center, re-compute cluster centers, re-assign observations to cluster centers, and so on.

2.5 In the elbow method we look for the point at which the marginal improvement in inertia (i.e., within cluster sum of squares) when an additional cluster is introduced is small. In the silhouette method, we calculate for each value of k and for each observation i

$a(i)$: the average distance from other observations in its own cluster, and

$b(i)$: the average distance from observations in the nearest cluster.

The observation's silhouette is

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

and the best value of k is the one for which the average silhouette across all observations is greatest.

2.6 As the number of features increases, the sum of the squared differences between feature values has more terms and therefore tends to increase. When the ten additional features are created by mistake, the distance between two observations increases by $\sqrt{2}$ because every squared difference is calculated twice.

2.7 In hierarchical clustering we start by putting every observation in its own cluster. At each step, we find the two closest clusters and join them to create a new cluster. The disadvantage is that it is slow. The advantage is that it identifies clusters within clusters.

2.8 Distribution-based clustering involves assuming that observations are created by a mixture of distributions and using statistical methods to separate them. Density-based clustering involves adding new points to a cluster that are close to several points already in the cluster. It can lead to non-standard shapes for the clusters.

2.9 Principal components analysis (PCA) is useful when there are a number of highly correlated features. It has the potential to explain most of the variability in the data with a small number of new factors (which can be considered as manufactured features) that are uncorrelated with each other.

2.10 A factor loading is the amount of each original feature in the factor. Each observation can be expressed as a linear combination of the

factors. A factor score for an observation is the amount of the factor in the observation.

2.11 We can verify the numbers in Table 2.5 as follows:

	Peace index	*Legal risk index*	*GDP*
Average for 14 high-risk countries	2.63	4.05	−3.44
Average for all countries	2.00	5.60	2.37
SD for all countries	0.45	1.49	3.24
Normalized average for 14 countries	1.39	−1.04	−1.79

For example, for the peace index, (2.63−2.00)/0.45 = 1.39.

2.12 Define X_1, X_2, X_3, and X_4 as the corruption index, peace index, legal index, and GDP growth rate. If these are normalized Table 2.11 shows that the factors are:

$$0.594X_1-0.530X_2+0.585X_3+0.152X_4$$

and

$$0.154X_1+0.041X_2+0.136X_3-0.978X_4$$

The first factor assigns roughly equal weight to the three indices and little weight to GDP growth rate. The second factor assigns nearly all the weight to GDP growth rate. This shows that GDP growth rate is providing a different type of information from the other three measures.

If we want to create factors using non-normalized data, we can divide each coefficient by the standard deviation of the corresponding feature value. The first factor becomes:

$$0.031X_1-1.185X_2+0.394X_3+0.047X_4$$

and the second one becomes:

$$0.008X_1+0.092X_2+0.091X_3-0.302X_4$$

Chapter 3

3.1 The objective in "plain vanilla" linear regression is to minimize the mean squared error of the forecasted target values.

3.2 In the case of Ridge regression we add a constant times the sum of the squares of the coefficients to the mean squared error. In the case of Lasso regression we add a constant times the sum of the absolute values of the coefficients. In the case of Elastic Net regression we add a constant times the sum of the squares of the coefficients and a different constant times the sum of the absolute values of the coefficients.

3.3 Ridge regression reduces the magnitude of the coefficients when the correlation between features is high. Lasso regression sets to zero the values of the coefficients of variables that have little effect on prediction results.

3.4 A single dummy variable which equals one if the house has air conditioning and zero otherwise could be used.

3.5 We could use a single dummy variable which equals 0 for no slope, 1 for gentle slope, 2 for moderate slope, and 3 for steep slope.

3.6 We would create a dummy variable for each neighborhood. The dummy variable equals one if the house is in the neighborhood and zero otherwise.

3.7 Regularization is designed to avoid over-fitting by reducing the weights (i.e. coefficients) in a regression. L1 regularization is Lasso where a constant times the sum of the absolute values of the coefficients is added to the objective function. It makes some of the weights zero. L2 regularization is Ridge where a constant times the sum of the squares of the coefficients is added to the objective function. It reduces the absolute magnitude of the weights.

3.8 The sigmoid function is:

$$f(y) = \frac{1}{1 + e^{-y}}$$

3.9 The objective in logistic regression is to maximize

$$\sum_{\substack{\text{Positive}\\ \text{Outcomes}}} \ln(Q) + \sum_{\substack{\text{Negative}\\ \text{Outcomes}}} \ln(1 - Q)$$

where Q is the estimated probability of a positive outcome.

3.10 The true positive rate is the proportion of positive outcomes that are predicted correctly. The false positive rate is the proportion of

negative outcomes that are predicted incorrectly. The precision is the proportion of positive predictions that are correct.

3.11 In the ROC curve the true positive rate is plotted against the false positive rate. It shows the trade-off between correctly predicting positive outcomes and failing to predict negative outcomes.

3.12 The dummy variable trap is the problem that when a categorical variable is hot encoded and there is a bias (constant term), there are many sets of parameters that give equally good best fits to the training set. The problem is solved with regularization.

3.13 The plain vanilla linear regression result for salary ('000s), Y, is:

$$Y = 178.6 - 20{,}198.5X_1 + 89{,}222.3X_2 \\ -151{,}267.2X_3 + 116{,}798.2X_4 - 34{,}494.8X_5$$

With an mse of 604.9. For Ridge we have:

λ	A	b_1	b_2	b_3	b_4	b_5	mse
0.02	178.6	102.5	56.2	10.0	−33.4	−72.9	889.5
0.05	178.6	78.2	43.4	9.7	−21.1	−48.2	1,193.9
0.10	178.6	57.3	33.3	10.2	−10.7	−28.9	1,574.0

For Lasso we have

λ	A	b_1	b_2	b_3	b_4	b_5	mse
0.02	178.6	0.0	175.6	0.0	264.0	−380.3	711.8
0.05	178.6	0.0	250.8	0.0	0.0	−190.2	724.4
0.10	178.6	0.0	249.7	0.0	0.0	−189.1	724.6

3.14 (a) The objective function in equation (3.7) has the nice property that

$$1 - Q = 1 - \frac{1}{1 + \exp\left(-a - \sum_{j=1}^{m} b_j X_j\right)}$$

$$= \frac{\exp\left(-a - \sum_{j=1}^{m} b_j X_j\right)}{1 + \exp\left(-a - \sum_{j=1}^{m} b_j X_j\right)} = \frac{1}{1 + \exp\left(a + \sum_{j=1}^{m} b_j X_j\right)}$$

When default is made the positive outcome, the maximum likelihood objective function leads to the sign of the bias and the sign of each of the weights changing. However, estimates of the probability of default and no-default are unchanged.

(b) When Z=0.25, the confusion matrix is

	Predict positive (default)	*Predict negative (no default)*
Outcome positive (default)	1.62%	16.26%
Outcome negative (no default)	4.53%	77.59%

When Z=0.20, the confusion matrix is

	Predict positive (default)	*Predict negative (no default)*
Outcome positive (default)	8.13%	9.75%
Outcome negative (no default)	26.77%	55.34%

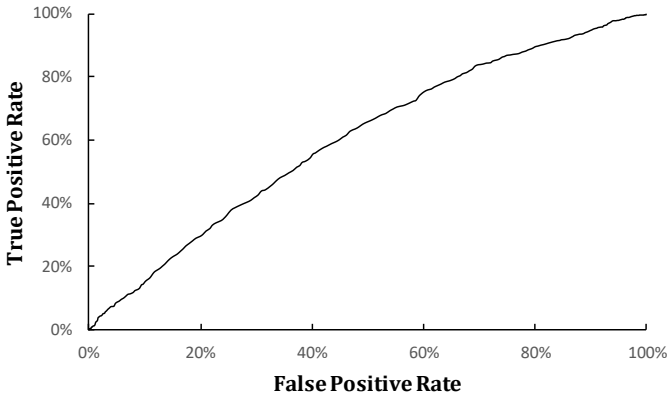
When Z=0.15, the confusion matrix is

	Predict positive (default)	*Predict negative (no default)*
Outcome positive (default)	14.15%	3.74%
Outcome negative (no default)	53.47%	28.65%

The ratios become:

	Z = 0.25	*Z = 0.20*	*Z = 0.15*
Accuracy	79.21%	63.47%	42.80%
True Positive Rate	9.07%	45.46%	79.11%
True Negative Rate	94.48%	67.39%	34.89%
False Positive Rate	5.52%	32.61%	65.11%
Precision	26.37%	23.29%	20.93%
F-score	13.50%	30.80%	33.10%

(c) The ROC curve can be calculated by changing the data so that a default is labeled as 1 and no-default is labeled as 2. The ROC curve is



Define TPR(new), FPR(new), and Z(new) as the true positive rate, false positive rate, and Z-value when default is the positive outcome and TPR(old), FPR(old), Z(old) as the true positive rate, false positive rate, and Z-value when no-default is the positive outcome. When the Z(new) equals one minus the Z(old)), we have

$$\text{TPR(new)} = 1 - \text{FPR(old)}$$

$$\text{FPR(new)} = 1 - \text{TPR(old)}$$

This symmetry leads to AUC being unchanged.

Chapter 4

4.1 In the decision tree approach the features are considered one-by-one in order of importance whereas in the regression approach they are considered all at once. The decision tree approach does not assume linearity and is more intuitive. It is also less sensitive to outlying observations than linear regression.

4.2 When there are n alternative outcomes entropy is

$$-\sum_{i=1}^{n} p_i \ln(p_i)$$

where p_i is the probability of the ith outcome.

4.3 When there are n alternative outcomes the Gini measure is defined as

$$1 - \sum_{i=1}^{n} p_i^2$$

where p_i is the probability of the ith outcome.

4.4 Information gain is measured as the reduction in either entropy or the Gini measure.

4.5 The threshold is the value that maximizes the information gain.

4.6 The naïve Bayesian classifier assumes that, for observations in a class, feature values are independent.

4.7 The ensemble method is a way of combining multiple algorithms to make a single prediction.

4.8 A random forest is an ensemble of decision trees. The different decision trees are created by using a subset of features or a subset of observations or by changing threshold values.

4.9 Bagging involves sampling from observations or features so that the same algorithm is used on different training sets. Boosting involves creating models sequentially with each model attempting to correct the error made by the previous model.

4.10 The decision tree algorithm is transparent in that it is easy to see why a particular decision was made.

4.11 Predict that loans are good only if FICO > 717.5 and Income > $48,750. The confusion matrix is:

	Predict no default	Predict default
Outcome positive (no default)	12.94%	69.19%
Outcome negative (default)	1.42%	16.45%

4.12 Conditional on a good loan, the probability density for a FICO score of 660 is

$$\frac{1}{\sqrt{2\pi} \times 31.29} \exp\left(-\frac{(660 - 696.19)^2}{2 \times 31.29^2}\right) = 0.0065$$

Conditional on a good loan, the probability density function for an income of 40 is

$$\frac{1}{\sqrt{2\pi} \times 59.24} \exp\left(-\frac{(40 - 79.83)^2}{2 \times 59.24^2}\right) = 0.0054$$

Conditional on a default the probability density for a FICO score of 660 is

$$\frac{1}{\sqrt{2\pi} \times 24.18} \exp\left(-\frac{(660 - 686.65)^2}{2 \times 24.18^2}\right) = 0.0090$$

Similarly, conditional on a default the probability density for an income of 40 is

$$\frac{1}{\sqrt{2\pi} \times 48.81} \exp\left(-\frac{(40 - 68.47)^2}{2 \times 48.81^2}\right) = 0.0069$$

The probability of the loan being good is

$$\frac{0.0065 \times 0.0054 \times 0.8276}{Q} = \frac{2.90 \times 10^{-5}}{Q}$$

where Q is the joint probability density of FICO = 660 and income =40. The probability of the loan defaulting is

$$\frac{0.0069 \times 0.0090 \times 0.1724}{Q} = \frac{1.07 \times 10^{-5}}{Q}$$

The probability of the loan defaulting is therefore

$$\frac{1.07}{1.07 + 2.90}$$

or about 27%.

Chapter 5

5.1 The objective in SVM classification is to find a pathway that minimizes an objective function which is a function of the cost of violations and the width of the pathway. The center of the pathway is used to classify observations.

5.2 In a hard margin classification, the objective is to find the widest path that has no misclassified observations (assuming such a pathway exists). In a soft margin classification, an objective function incorporates a trade-off between the width of the pathway and the extent of the misclassification.

5.3 The initial equations are

$$\sum_{j=1}^{m} w_j x_j = b_u$$

$$\sum_{j=1}^{m} w_j x_j = b_d$$

Without loss of generality parameters can be scaled so that the equations become

$$\sum_{j=1}^{m} w_j x_j = b + 1$$

and

$$\sum_{j=1}^{m} w_j x_j = b - 1$$

5.4 The width of the pathway decreases as we give more weight to violations.

5.5 For a positive outcome observation, it is

$$\max\left(b + 1 - \sum\nolimits_{j=1}^{m} w_j\, x_{ij}, 0\right)$$

for negative outcome observations it is

$$\max\left(\sum\nolimits_{j=1}^{m} w_j\, x_{ij} - b + 1, 0\right)$$

5.6 We transform the features and create new features so that linear classification can be used.

5.7 A landmark is a point in feature space, which may or may not correspond to an observation, and the Gaussian radial bias function is a synthetic feature whose value for an observation declines as the observation becomes further away from landmark.

5.8 In SVM regression the objective is to find a pathway with a prespecified width through the space defined by the target and the features. The pathway is designed to include as many observations as possible. Observations outside the pathway give rise to violations. The objective function minimizes the extent of the violations and incorporates some regularization. Similarly to Ridge regression, the regularization is designed to avoid weights that are large

in magnitude.

5.9 The differences are as follows:

- The relationship between the target and the features is represented by a pathway rather than a single line.
- The prediction error is counted as zero when an observation lies within the pathway
- Errors for observations outside the pathway are calculated as the difference between the target value and the closest point in the pathway that is consistent with the feature values.
- There is some regularization built into the objective function.

5.10 The table produced by Sklearn's SVM package is as follows. The numbers produced by Excel may be slightly different.

C	w_1	w_2	B	*Loans misclassified*	*Width of pathway*
0.01	0.042	0.015	3.65	20%	44.8
0.001	0.038	0.013	3.40	20%	49.8
0.0005	0.019	0.008	1.90	20%	96.8
0.0003	0.018	0.004	1.71	30%	105.6
0.0002	0.018	0.002	1.63	40%	212.7

Chapter 6

6.1 A hidden layer is a set of intermediate values used when the outputs are calculated from the inputs in a neural network. The set of inputs form the input layer. The set of outputs form the output layer. A neuron is one element within a hidden layer for which a value is calculated. An activation function is the function used for the calculation of the values at neurons in one layer from values at neurons in the previous layer.

6.2 The sigmoid function for calculating the value at a neuron is

$$f(y) = \frac{1}{1 + e^{-y}}$$

where y is a constant (the bias) plus a linear combination of the values at the neurons in the previous layer.

6.3 The universal approximation theorem states that any continuous non-linear function can be approximated to arbitrary accuracy using a neural network with one layer.

6.4 When the target is numerical the suggested final activation function is linear. When observations are being classified, the suggested activation function is the sigmoid function.

6.5 The learning rate is the size of the step taken down the valley once the line of steepest descent has been identified.

6.6 If the learning rate is too low the steepest descent algorithm will be too slow. When it is too high there are liable to be oscillations with the minimum not being found.

6.7 The results for the validation set are produced at the same time as the results for the training set. The algorithm is stopped when the results for the validation set start to worsen. This is to avoid overfitting.

6.8 When a derivative is normally valued using a computationally slow numerical procedure such as Monte Carlo simulation, an ANN can be created for valuation. Data relating the derivative's value to the inputs is created using the numerical procedure. The ANN is then trained on the data and used for all future valuations.

6.9 In a regular ANN the value at a node in one layer is related to values at all nodes in the previous layer. In a CNN it is related to a small subset of the nodes in the previous layer.

6.10 In an RNN there is a time sequence to the data. The nodes in one layer are related to values calculated for the same nodes at the previous time as well as to the nodes in the previous layer.

6.11 The number of parameters is $6\times10+10\times11\times1+11\times1= 181$

6.12 When the starting point is 1.0000 and the learning rate is 0.0002 we obtain:

Iteration	Value of b	Gradient	Change in b
0	1.0000	−11,999	2.3998
1	3.3998	−4342.2	0.8684
2	4.2682	−1571.4	0.3143
3	4.5825	−568.6	0.1137
4	4.6962	−205.8	0.0412
5	4.7374	−74.5	0.0149
6	4.7523	−26.9	0.0054
7	4.7577	−9.8	0.0020
8	4.7596	−3.5	0.0007
9	4.7603	−1.3	0.0003
10	4.7606	−0.5	0.0001
11	4.7607	−0.2	0.0000
12	4.7607	−0.1	0.0000

6.13 The sigmoid function has an argument of zero so that $V_1 = V_2 = V_3 = 0.5$. The house value is calculated as $3 \times 0.5 \times 100 = 150$.

Chapter 7

7.1 In reinforcement learning the objective is to calculate the best strategy for taking a sequence of decisions through time in a changing environment. Supervised learning involves one or more estimates being made from features at a single point in time.

7.2 Exploitation involves taking the best action identified so far. Exploration involves randomly selecting a different action. If an algorithm just involved exploitation it might never find the best action. If it just involved exploration, it would not benefit from what it has learned.

7.3 Dynamic programming involves working from the horizon date back to the beginning, working out the best action for each of the states that can arise.

7.4 The optimal strategy is to leave your opponent with $4n+1$ matches where n is an integer. When there are 8 matches the optimal strategy is to pick up 3 matches. After 1000 games this has been identified as the best strategy, but not convincingly so. After 5,000 and 25,000 games, the best decisions become more clearly differentiated.

7.5 In the Monte Carlo approach each trial involves actions being taken with exploration and exploitation. The new observation on the value of taking a particular action in a particular state is the total future reward (possibly discounted) from the time of the action until the horizon date

7.6 In temporal difference learning each trial involves actions being taken with exploration and exploitation. The new observation on the value of taking a particular action in a particular state is determined by looking one period ahead and using the most recent estimate of the value of being in the state that will be reached.

7.7 When there are many actions or many states (or both) the action/state matrix does not fill up quickly and values can be estimated using an ANN.

7.8 Deep Q-learning is when an ANN is used in conjunction with temporal difference learning.

7.9 In the case of the Monte Carlo approach we update as follows:

$$Q(8,1) = 0.272+0.05(1.000-0.272) = 0.308$$

$Q(6,1) = 0.155+0.05(1.000-0.155) = 0.197$

$Q(4,1) = 0.484+0.05(1.000-0.484) = 0.510$

$Q(2,1) = 0.999+0.05(1.000-0.999) = 0.999$

In the case of the temporal difference approach we update as follows:

$Q(8,1) = 0.272+0.05(0.155-0.272) = 0.266$

$Q(6,1) = 0.155+0.05(1.000-0.155) = 0.197$

$Q(4,1) = 0.484+0.05(0.999-0.484) = 0.510$

$Q(2,1) = 0.999+0.05(1.000-0.999) = 0.999$

Chapter 8

8.1 A sentiment analysis involves processing textual data from such sources as social media and surveys to determine whether it is positive, negative, or neutral about a particular product, company, person, event, etc.

8.2 There are some publicly available data sets where opinions have been labeled. These are sometimes used for training. Otherwise, it is necessary to manually label the opinions used for training and testing.

8.3 The text must be split into words. Punctuation must be removed. Very common words, such as "the", "a" and "and" (referred to as stop words) can be removed. Stemming can be applied to replace a word by its stem (e.g., "sleeping" by "sleep"). Lemmatization can be used to reduce a word to its root (e.g., "better" to "good"). Spelling mistakes can be corrected. Abbreviations can be replaced by the full word. Rare words can be removed.

8.4 Stemming involves removing suffices such as "ing" and "s". Lemmatization involves searching for the root word.

8.5 A bag-of-words model represents text by the frequency with which words occur.

8.6 Negatives such as "not" mean that wrong conclusions are liable to be reached if a bag-of-words model merely looks at the frequency with which a single word appears. An improvement is to look at word pairs (bigrams).

8.7 The naïve Bayes classifier assumes that the occurrence of one word is independent of the occurrence of another word.

8.8 A trigram is a group of three words.

8.9 (a) Logistic regression provides a probability of a positive and negative sentiment while SVM does not.
(b) Logistic regression does not require the assumption that the occurrence of one word is independent of the occurrence of another word.

8.10 Laplace smoothing is designed to deal with the problem that a word appears in an opinion but not in one class of the training set observations. It changes the probability of that class from zero to a small number.

8.11 TF of a word in a document is the number of times the word appears in the document divided by the number of words in the document. IDF of a word is $\log(N/n)$ where N is the total number of documents and n is the number of documents containing the word. In information retrieval, TF is multiplied by IDF to provide, for each document that might be retrieved, a score for each word in a search request.

8.12 A word vector is a set of numbers describing the meaning of a word. It does this by quantifying the extent to which the word tends to appear in close proximity to other words.

8.13 In this case, $p_1 = 0.667$, $p_2 = 0.5$, $p_3 = 0.667$, $q_1 = 0.5$, $q_2 = 0.25$, and $q_3 = 0.75$. Also Prob(Pos) = 0.6 and Prob(Neg) = 0.4. The probability of a positive sentiment is:

$$\frac{0.667 \times 0.5 \times 0.667 \times 0.6}{0.667 \times 0.5 \times 0.667 \times 0.6 + 0.5 \times 0.25 \times 0.75 \times 0.4} = 0.78$$

Chapter 9

9.1 Neural networks, SVM models, and ensemble models are difficult to interpret.

9.2 The weights of a linear model have a simple interpretation. They show the effect of changing the value of one feature while keeping the others the same.

9.3 From Table 9.1, each extra square foot of lot size is worth \$0.3795. it follows that an extra 5,000 square feet is worth 5,000 × \$0.3795 or \$1,897.50.

9.4 It is the minus the sensitivity of a positive outcome or

$$-\frac{\exp[-(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)]}{\{1 + \exp[-(a + b_1X_1 + b_2X_2 + \cdots + b_mX_m)]\}^2} b_j u$$

9.5 "Odds against" shows the profit of $1 bet that an event will happen when the $1 is forfeited if it does not happen. "Odds on" shows the amount that must be staked to provide a $1 profit if an event happens and a total loss if it does not happen. The natural logarithms of odds on and odds against are linear in the features in a logistic regression.
9.6 The contributions calculated for a feature usually assume that the feature changes with all other features remaining fixed. If there are interactions between features, it may be unrealistic to assume that one feature can change without other features changing.
9.7 A partial dependence plot shows what happens on average when one feature changes. Results are averaged over random changes in the other features.
9.8 Shapley values are designed so that when there is a change the sum of the contributions to the change of each feature equals the total change.
9.9 LIME finds a simple model that fits a complicated model for values of the features that are close to the currently observed values.
9.10 4! = 24.

Chapter 10

10.1 A call option is an option to buy at a certain price. A put option is an option to sell at a certain price.
10.2 The payoffs for a derivative are not symmetrical. The impact of a certain increase in the price of the underlying asset is not the same as the impact of the same decrease in the price of the underlying asset.
10.3 The moneyness of an option is a measure of the extent to which it seems likely that the option will be exercised. It can be measured by comparing S to K. Traders tend to measure moneyness of an option in terms of its delta.
10.4 The six variables are the stock price, strike price, risk-free rate, dividend yield, volatility, and time to maturity.
10.5 A portfolio is delta neutral if its value is virtually unchanged when a small change is made to the price of the underlying asset with all other variables being kept the same.
10.6 An implied volatility is the volatility which, when substituted into the Black–Scholes–Merton formula (or its extensions) gives the market price of an option.

10.7 A volatility surface is a three-dimensional chart showing the relationship between (a) the implied volatility of an option and (b) its strike price and time to maturity.

10.8 Volatilities tends to increase (decrease) when the price of the underlying asset decreases (increases). An increase in the price of the underlying asset causes a call option price to increase. However, the increase tends to be accompanied by a decrease in volatility which reduces the call option price. A decrease in the price of the underlying asset causes a call option price to decrease. However, the decrease tends to be accompanied by an increase in volatility which increases the call option price. The relationship between the volatility surface movements and the asset price movements therefore tends to reduce a trader's exposure to movements in the asset price.

10.9 There are improvements when the cost of trading is not negligible.

10.10 When securities to hedge volatility are traded, bid–offer spreads tend to be high.

Glossary of Terms

Accuracy ratio: The percentage of observations that are classified correctly

Activation function: Function used to relate values at neurons in one layer to values at neurons in previous layer

AdaBoost: Boosting where weights for misclassified observations are increased

Adam: Short for adaptive moment estimation. It is a popular way of choosing learning rates in neural networks

Adaptive learning rate: Learning rate in a neural network that is designed to adapt to circumstances

Adversarial machine learning: Developing data to fool a machine learning algorithm

Agglomerative clustering: See hierarchical clustering

AlphaGo: Program developed by Google to play the board game Go

ANN: See neural network

Artificial intelligence: The development of intelligence artificially. Machine learning is one branch of artificial intelligence.

Artificial neural network: See neural network

AUC: Area under the ROC curve

Autoencoder: The use of a neural network to reduce the number

of features in data

Backpropagation: A fast way of calculating partial derivatives in a neural network by working from the end to the beginning

Bag-of-words: A natural language processing model where the words in text are considered without regard to their order

Bagging: Training the same algorithm on different random subsets of data

Bayes' theorem: A theorem for inverting conditionality, i.e., deriving $\text{Prob}(Y|X)$ from $\text{Prob}(X|Y)$

Bayesian learning: Using Bayes' theorem to update probabilities

Bias: Constant term

Bias-variance trade-off: The trade-off between using a simple model that under-fits the training set and a too complicated model that over-fits

Bigram: A sequence of two words

Black-box model: Model that is difficult to interpret

Boosting: Iterative training procedure where one algorithm attempts to correct errors in a previous algorithm

Categorical feature: Non-numerical feature that falls into one of a number of categories

Centroid: The center of a cluster

Class: One of two or more groups into which data is divided

Class imbalance: Situation where the number of observations in one class is quite different from the number in another class

Classification model: Model for dividing observations into classes

Classification threshold: Threshold for deciding which class an observation belongs to

CNN: See convolutional neural network

Clustering: Process of dividing data into clusters so as to understand it better

Confusion matrix: A table for classification models showing which prediction for the test set are correct and which are incorrect

Convolutional neural network: A type of neural network particularly suitable for image recognition where there are one or more rectangular arrays of numbers at each layer

Cost function: The objective function to be minimized

Decision Tree: A way of considering features one by one

Decoding: The second part of an autoencoder

Deep Q-learning: Combining reinforcement learning with a neural

network to determine Q-values

Deep reinforcement learning: See deep Q-learning

Density-based clustering: A way of forming non-standard cluster patterns

Distribution-based clustering: A way of clustering by fitting observations to a mixture of distributions

Dropouts: Neurons removed in a gradient descent algorithm to make training faster

Dummy variable: A variable that has a value of 0 or 1 to handle a categorical feature

Dummy variable trap: A situation where the introduction of dummy variables leads to no unique best fit set of parameters in a linear regression.

Dynamic programming: Calculating optimal decisions by working back from the horizon date

Elastic Net regression: Combination of Lasso and Ridge regression

Elbow method: Procedure for choosing the optimal number of clusters by observing the effect on inertia

Encoding: The first part of an autoencoder

Ensemble learning: Learning by combining the results from several algorithms

Entropy: A measure of uncertainty

Episode: A trial in reinforcement learning

Epoch: A set of iterations that make one complete use of the training set

Exploitation: Pursuing the best action identified so far

Exploration: Choosing an action at random

Exponentially weighted moving average: An average from past observations where the weight given to the observation n periods ago is λ times the weight given to the observation $n-1$ periods ago ($\lambda < 1$)

F1-score: See F-score

Factor loading: The amount of a feature in a factor when a principal components analysis is carried out

Factor score: The amount of a factor in an observation when a principal components analysis is carried out

False negative rate: Percentage of positive outcomes predicted as negative in classification

False positive rate: Percentage of negative outcomes predicted as

positive in classification

Feature: A variable used in a machine learning algorithm

Feature map: A component of a layer in a convolutional neural network

Feature scaling: A procedure to ensure that the features are measured on similar scales

FICO score: Credit score in the United States

F-score: A measure calculated from the precision and true positive rate in a classification

Gap statistic: A way of choosing the number of clusters by comparing the clustered data with randomly distributed data

GDPR: See General Data Protection Regulation

General Data Protection Regulation: Regulation introduced by the European Union

Generalize: A model developed using the training set generalizes well if its results are almost as good for the validation set

Gini measure: A measure of uncertainty

Gradient boosting: Boosting where a new predictor is fitted to the errors of previous predictors.

Gradient descent algorithm: Calculates minimum by taking steps down a multi-dimensional valley

Greedy action: See exploitation

Hans: German horse that seemed to be intelligent

Hard margin classification: SVM classification when there are no violations

Hidden layer: A layer of neurons between the input layer and the output layer

Hierarchical clustering: A way of building up clusters one observation at a time

Hyperbolic tangent function: An activation function sometimes used in neural networks

Hyperparameter: Parameter used to train a model, but not to make predictions

Hyperplane: A boundary separating an n-dimensional space into two regions. The boundary has n-1 dimensions

Imbalanced data set: See class imbalance

Inertia: The within-cluster sum of squares when data is clustered

Information gain: Reduction in uncertainty

Input layer: The set of feature values that are input to a neural network

Instance: An observation

Iteration: A single update of weights during training

Kernel trick: Short cut when new features are created from existing features

***k*-means algorithm:** An algorithm for finding clusters

L1 regularization: See Lasso regression

L2 regularization: See Ridge regression

Label: Value of a target

Landmark: A point in feature space used to create a new feature

Lasso regression: Regularization by adding the sum of the absolute values of the weights to the objective function in a linear regression

Layer: Term used to describe the inputs or the outputs or a set of intermediate neurons in a neural network

Learning rate: Step size in a gradient descent algorithm

Learning rate decay: Reduction in learning rate as the gradient descent algorithm proceeds

Lemmatization: Mapping a word to its root word. For example, "better" could be mapped to "good"

LIME: A way of interpreting a black-box model for values of features that are close to their current value

Linear regression: Regression where relationship between target and features is assumed to be linear

Logistic regression: Version of regression used for classification

Long short-term memory: A way of testing the importance of values from previous observations and using them for updating as appropriate

LSTM: See long short-term memory

Machine learning: Creating intelligence by learning from large volumes of data

MAE: See mean absolute error

Maximum likelihood method: Determination of parameters by maximizing the probability of observations occurring

Mean squared error: The arithmetic average of the squares of the errors

Mini-batch stochastic gradient descent: Gradient descent where a subset of data is used on each iteration

Min-max scaling: Feature scaling by subtracting the minimum feature value and dividing by the difference between the maximum and the minimum feature values

Momentum: The use of exponentially weighted moving averages for gradients in gradient descent algorithms

Monte Carlo method: Updating Q-values in reinforcement learning by looking at total future rewards (possibly with discounting)

MSE: See mean squared error

Multi-armed bandit problem: The problem of choosing which is best of a number of levers that give rewards

Naïve Bayes classifier: A way of calculating conditional probabilities (or conditional values for a numerical variable) if features in a category are independent

Natural language processing: The application of machine learning to written or spoken words

Negative class: One of two groups into which data is divided

Neural network: Network of functions used to create a non-linear model relating targets to features

Neuron: Node containing an intermediate value in a neural network

Nim: Game to illustrate reinforcement learning

NLP: See natural language processing

Normalization: Scaling feature values to make them comparable

***n*-step bootstrapping:** Updating Q-values in reinforcement learning by observing value after n time steps

Objective function: A function that is to be maximized or minimized by an algorithm

One-hot encoding: Conversion of a categorical feature to numerical variables

Outliers: Observations that are distant from most other observations

Output layer: The set of targets that are output from a neural network

Over-fitting: A model that fits noise in the training set and does not generalize well to the validation set

Partial dependence plot: Plot of average value of target against a feature when other features are allowed to take random values

Positive class: One of two groups into which data is divided

Precision: Percentage of positive predictions that turn out to be positive in a classification

Principal components analysis: A way of replacing data on correlated features with a smaller number of uncorrelated factors

Project Maven: Project between Google and U.S. Department of

Defense which was canceled by Google

***P*-value:** Probability in a linear regression of obtaining a *t*-statistic as large as the one observed if the corresponding feature has no explanatory power

***Q*-learning:** Learning the optimal actions for different states in reinforcement learning

***Q*-value:** The best value identified so far for a particular state and action in reinforcement learning

Radial bias function: Function of the distance of an observation from a landmark. Used to create a new feature to determine a non-linear pathway in SVM

Random forest: Ensemble of decision trees

RBF: See radial bias function

Recall: See true positive rate

Receiver Operating Curve: Plot of true positive rate against false positive rate in a classification

Receptive field: A rectangular array of nodes in the layer of a convolutional neural network

Recurrent neural network: Neural network where weights are given to values calculated from the immediately preceding observation

Regularization: Simplification of a model that can avoid overfitting and reduce the magnitude of weights

Reinforcement learning: Developing a multistage decision strategy in a changing environment

Relu: An activation function used in neural networks

Reward: Payoff in reinforcement learning

Ridge regression: Regularization by adding the sum of the squared weights to objective function in a linear regression

RMSE: See root mean squared error

RNN: See recurrent neural network

ROC: See receiver operating curve

Root mean squared error: Square root of the mean squared error

***R*-squared statistic:** Proportion of variance in target explained by the features in a linear regression

Semi-supervised learning: Predicting the value of the target when only part of the available training data includes values for the target

Sensitivity: See True positive rate

Sentiment analysis: Determining a group's attitude (positive or negative) to a service, product, organization, or topic

Shapley values: Values that determine the contributions of features to a change in such a way that the total of the contributions equals the total change

Sigmoid function: S-shaped function with values between zero and one used in logistic regression and neural networks

Silhouette method: A way of calculating the number of clusters based on the distances between observations

SMOTE: Synthetic Minority Over-sampling Technique. A way of overcoming class imbalance.

Soft margin classification: SVM classification when there are violations

Specificity: See true negative rate

Spoofing: Illegal attempt to manipulate the market

Standardization: See normalization

Stemming: Removing suffices such as "ing" from a word

Step size: See learning rate

Stop word: A word such as "the" or "a" that does not add much to the meaning of text

Stopping rule: A rule for stopping learning in a neural network when validation results worsen

Supervised learning: Predicting the value of one or more targets

Support vector: Observation at the edge of the pathway

SVM classification: Construction of a pathway to classify observations

SVM regression: Using a pathway to predict the value of a continuous target variable

Target: A variable about which predictions are made

Tay: Program introduced by Microsoft for interacting with female teenagers

Temporal difference learning: Updating Q-values in reinforcement learning by observing the value at the next time step

Test set: Data set used to determine the accuracy of the model that is finally chosen

Tokenization: Splitting text into its constituent words

Training set: Data set used to estimate parameters for models that are tested

True negative rate: Percentage of negative outcomes correctly predicted in a classification

True positive rate: Percentage of positive outcomes correctly predicted in a classification

***t*-statistic:** Value of a parameter divided by its standard deviation in a linear regression

Under-fitting: Using a model that does not capture all key relationships in the training set

Universal approximation theorem: A theorem showing that a neural network with a single hidden layer can reproduce any continuous function to arbitrary accuracy if it has enough neurons

Unlabeled data: Data without target values

Unsupervised learning: Finding patterns in data, often by using clustering

Validation set: Data set used to determine how well a model derived from the training set works on different data

Weight: Coefficient of a feature value in a linear or logistic regression or coefficient of value at one layer to determine value at the next layer in a neural network

White-box model: Model that is easy to interpret

***Z*-score scaling:** Feature scaling by subtracting the mean and dividing by the standard deviation

Index

References to items in Glossary of Terms are **bolded**