CL | DataSchool

Practical Handbook of MACHINE LEARNING

Best suited for Software Professionals/College Students

***Exclusive Offer**

Get access to online **Full Length Video Lectures** by the authors through a **Scratch Code** inside the book

Sujit Bhattacharyya | Subhrajit Bhattacharya

Practical Handbook of Machine Learning

Best suited for Software Professionals/College Students

Sujit Bhattacharyya | Subhrajit Bhattacharya

GKP

Table of Contents

Preface

This book provides an introduction to machine learning written for software engineers or college students in their second or third year as well as for professionals or faculty members who want to learn this exciting field on their own.

The book focuses on explaining concepts of machine learning, which are important in applications. Some theoretical aspects are covered, to give enough intuition on how machine learning works, but without too much detail which can get a first-time student bogged down.

What we believe will set apart the book are the following:

- An unprecedented attention to an intuitive and gentle exposition to ML concepts which should make the book highly accessible to faculty members and professionals as well as serious college students, who want to learn the subject on their own.
- A plethora of practical tips collected from several years of application.
- Step by step hands-on exercises to help illustrate the concepts presented.
- Appendices on Basic Math's and Python Programming required to understand the programming exercises.
- Multiple choice questions with scoring for a quick test of the understanding of concepts. These are available only on the companion website.
- A set of **complementary videos** which we believe will liven up the learning process.
- A **companion website** with all the code and data used for the hands-on exercises.

We have kept out the following key topics:

- Algorithms such as gradient descent which are used to minimize loss functions.
- Artificial Neural Networks which we believe should be part of a second course.

Also, there are many regressors and classifiers available. We have introduced a select few of them instead of a larger number so that the reader is not lost in tracking the many algorithms that exist out there.

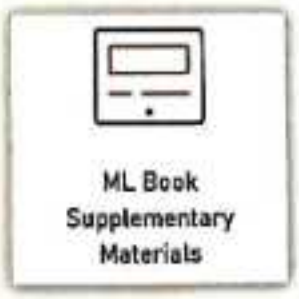
SCRATCH CODE

How to Access the Companion Website

To access the website follow the following steps:

1. Go to **https://www.aspiration.ai/mlbook**.
   You will see the screen below:

2. Register yourself by supplying the **SCRATCH CODE** of the book. This code can be seen on the top right corner of this page.
3. After you login you can access the ML Book Supplementary Materials



   You will see the list of chapters. You may access the videos, coding examples and assessments of each chapter by clicking on the various links.
4. If you need any assistance please write to **gkp@gkpublications.com**.

Sample Page After Login

aspiration.ai 25

- 1. ML/AI Revolution
- 2. Regression
- 3. Feature Engineering
- 4. Sources of Model Error
- 5. Bias and Variance
- 6. Model Validation
- **7. Classification**
- 8. Confusion Matrix
- 9. Distance based ML algorithms
- 10. Feature Scaling
- 11. Decision Trees
- 12. Ensemble Models
- 13. Random Forest Classifier
- 14. Unsupervised Learning
- **Appendix A** Python
- **Appendix B** Numpy
- **Appendix C** MatplotLib pyplot
- **Appendix D** Pandas
- **Appendix E** Statistics and Probability

Introduction to Classification

Classification is one of the most frequently and vastly used supervised learning method.

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y).

For example, spam detection in email service providers can be identified as a classification problem. This is a binary classification since there are only 2 classes as spam and not spam. A classifier utilizes some training data to understand how given input variables relate to the class. In this case, known spam and non-spam emails have to be used as the training data. When the classifier is trained accurately, it can be used to detect an unknown email.

Videos

Coding Examples

Chapter Code

1

The Machine Learning (ML)/ Artificial Intelligence (AI) Revolution

1.1 MACHINE LEARNING (ML): INTRODUCTION

We may not be aware but we are all very advanced users of Machine Learning (ML). From the moment you wake up till the time you go to bed, you would have used and taken advantage of several ML applications. Here are some of the most frequently seen examples of ML in our daily lives. When we tap in a message into WhatsApp and the predictive text suggests words to complete our sentence, the prediction algorithm is using ML. When we look for videos on YouTube, ML predicts which videos we might like and suggests the same to us. If you wish to buy something on Amazon, their ML algorithms will try and suggest the best products to you. If you speak to Alexa or Siri on your Apple iPhone your voice is recognized by Machine Learning algorithms. When you use Facebook, Machine Learning algorithms will decide which posts are most relevant for you and likely to engage you. When you check your email, you will notice that spam mail has been automatically separated from your important mails–this is also Machine Learning at work. When you are browsing a website, the ads which are being shown to you are based on the prediction on how likely you are to click on it–once again ML at work.

1.1.1 Role of ML in Prediction

In today's world, ML is being used by businesses and societies in virtually every sphere of life. There are examples of ML algorithms which have beaten world champions in games of strategy like Chess and Go. ML is being used in health-care to discover drugs and predict the disease and time to cure the disease. In 2012, President Obama's team used ML to predict voter behaviour and they used those predictions to decide where and when to campaign. ML is also being used to predict crime - by observing past patterns of data the algorithms are predicting the location of the future crimes! In the world of finance and stock markets, Machine Learning algorithms are making decisions of which stocks or commodities to buy or sell. In social media platforms, ML is being used to predict likely friends, turning the predictions into friend suggestions. Match-making websites are using ML algorithms to predict who we will like or marry. It is indeed hard to imagine an area in which Machine Learning is not being gainfully used. Some of the most valuable companies in the world-Google, Apple, Amazon, Facebook, Microsoft, Uber, Netflix, Airbnb, etc. are driven by AI and ML at their core.

1.2 MACHINE LEARNING VS CLASSICAL PROGRAMMING

Traditionally, computer programs have been written to provide precise step-by-step directions to computers to input some data and compute results from that data. Machine learning operates in a very different way. Machine learning, in a way, reverse engineers the logic from the past data and past outcomes and is able to solve very complicated problems. Let's take the example of handwriting recognition. Shown below are some digits of the famous MNIST dataset, very popularly used as an example in Machine Learning courses worldwide.

Figure 1.1 Samples from the MNIST Dataset on Handwritten Digits.

Imagine if we were a trying to construct a program to recognize these digits from the images shown above. We will input a single digit and the program will output which digit it is–the correct answer will be a single number from 0 to 9. A quick observation reveals the following challenges for us:

1. The input is non-uniform–every digit is written in a different and unique way
2. There is no formula or law to model the transformation.
3. Some of the digits are written in a way that makes it very hard for even us humans to decipher what digit it is.

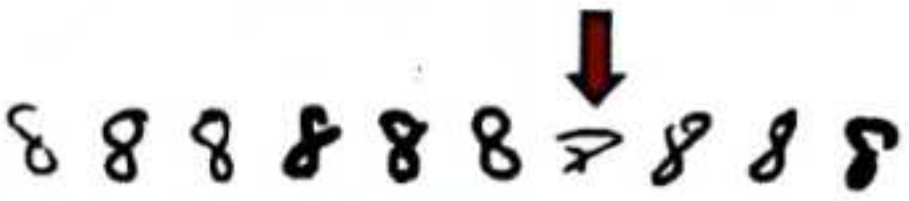


Figure 1.2 Sample 8's from the MNIST Database.

4. Since all the 8s are together, perhaps we can make out the seventh squiggle in the row of 8s is actually an 8. We need to find out similarities between digits to solve these types of problems
5. The correct output is sometimes a probability. In the above example, the poorly written digit is most likely an 8 because its likelihood of being any other digit is even more remote.

Our traditional programming paradigms expect us to provide clear and precise instructions on which a program of logical step-by-step instructions will operate, and reach a specific output. The example discussed earlier amply demonstrates that it is very hard to lay down a program which can solve this problem.

Now let's take the example of small child whom you are teaching and helping her recognize the digits 0 to 9. You show her one digit at a time, allow her to observe the same, and then you tell her what it is. As you show more and more examples repetitively, we know that the child is soon able to start identifying each and every digit. Initially, she might make a few mistakes, but eventually over time, she will achieve a very high accuracy of identifying any digit. She would have done this by inferring a general pattern for each digit by looking at sufficient examples. As more and more examples would be observed, it would have strengthened her pattern recognition ability and she would become more confident in stating the answer.

Machine Learning algorithms work very much like a child, solving problems very differently from traditional programming methods. They are able to do the following things:

a. Work out the general principle by observing the data (learning patterns in the input)
b. Handle the probabilistic nature of the outcome
c. Strengthen the rules of inference by looking at more examples
d. Learn from past errors and improve

The key inputs to Machine Learning algorithms are thus many examples of inputs that the program is expected to take and outcomes corresponding to these inputs. From these examples, ML algorithms are able to 'fit' a logic or a rule to arrive at the outcome for any input, including inputs it has not seen earlier. The central hypothesis of Machine Learning is: "If any learning algorithm is given sufficient and appropriate data, it can approximate any decision function closely."

1.2.1 Diagrammatic Representation of Classical Programming vs ML

In classical programming, the logic of the application is coded by a human programmer based on his/her understanding of the requirement. The results or outputs are produced by the application based on coding logic and the data input provided to the code. For example, an software application for loan processing would receive all details of the loan applicant, and based on the logic, output the amount of loan to be sanctioned. The same application may be built using ML but it would be very differently done. We would take historical data (e.g all loan applications of the previous year) and the past results (in this case the amount of loan sanctioned for each application) , and we would feed the input and past results to a ML algorithm. This ML algorithm would learn the relationship between the input and the output and create a model. This model is similar to a coding logic which can then take a new loan application and output the sanctioned loan amount. It is to be noted that in the second instance a human programmer did not provide the step by step logic of the loan sanction program. The logic was 'created' by the ML algorithm by looking at the mathematical patterns of the historical data and the previous outputs or results. The diagram below will help you understand the difference between classical programming and machine learning.

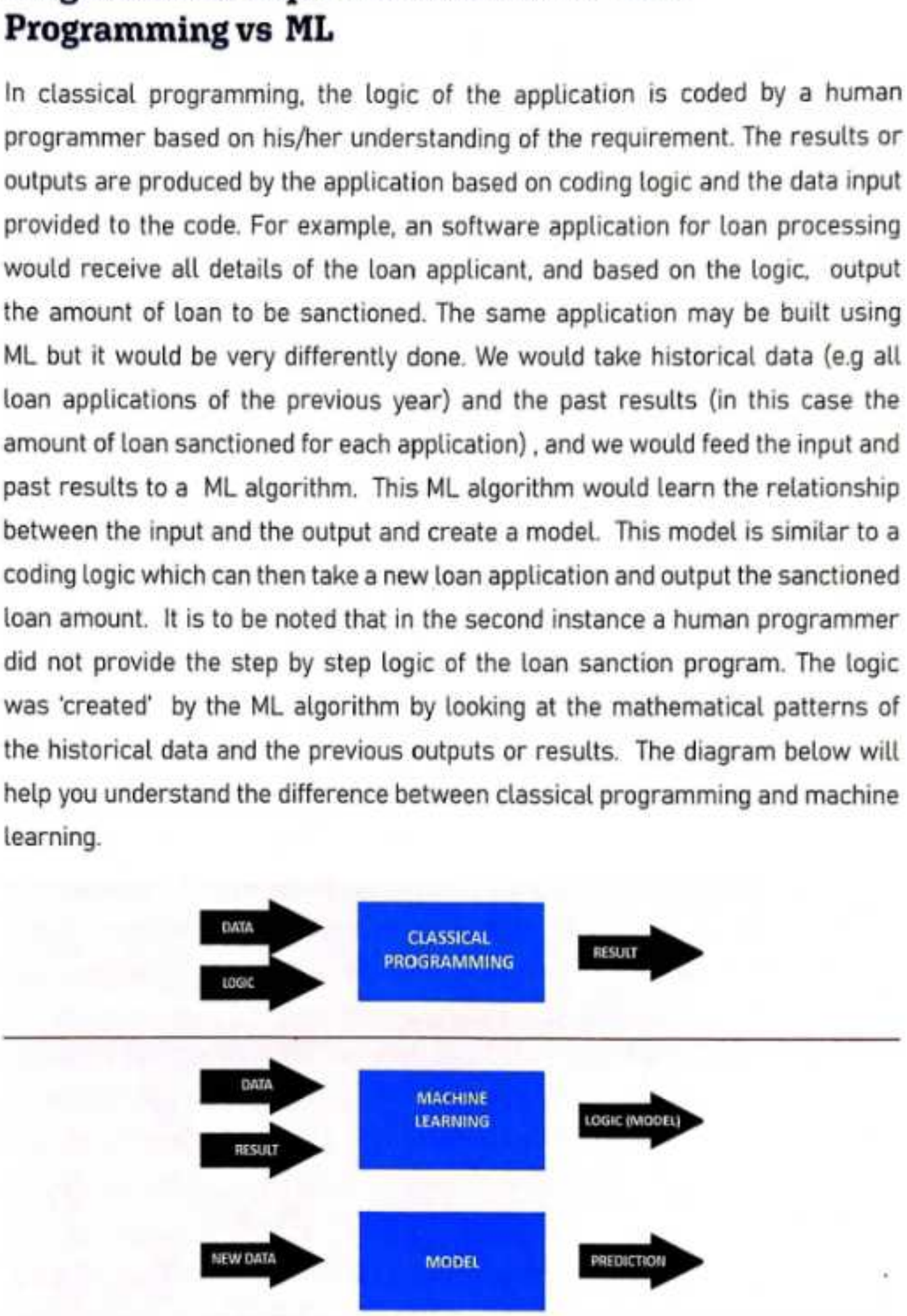


Figure 1.3 Machine Learning vs Classical Programming

1.3 THE ABILITY TO PREDICT

1.3.1 Examples of Prediction in the Real World

Weather Prediction

Suppose we have a model which can predict tomorrow's weather with a high degree of accuracy. This would provide us with many benefits. For example, if the model predicted that the next day would be rainy, we may carry an umbrella with us. A farmer planning to apply pesticide to his crops could plan an alternate date for his activity because rains would otherwise wash his pesticide away.

Object Recognition

Let's consider a second example. We are very good at recognizing objects we see. We can distinguish between different types of flowers, fruits, and animals. It is easy for us to look down the street and recognize if a car is headed our way–which has implications on whether we can cross the street or not. Suppose we want to write a software which can recognize the objects (cars, pedestrians, etc.) in images. This is a very difficult problem to solve. In fact, it is several degrees harder than trying to recognize handwritten digits, which we have discussed earlier in the chapter. But suppose we could build a model which could recognize objects in an image with high accuracy. What could we do with it? Investigative agencies could use the ability to identify wanted people from video feeds. If you wanted to find pictures of cats on the internet, the search software could use the image recognition capability of the aforesaid model to find pictures of interest to you. Such models are also a key component of the future of vehicles, *i.e.*, driverless cars. The object recognition problem is in some sense a prediction problem. When we say we recognize something or someone, we are actually predicting what the object is, usually with 100% confidence. When the object is not very familiar to us or is far away, our recognition becomes more like a guess or prediction. In a similar vein, when we talk about a model which recognizes objects from an image, it is also a prediction problem and we say the model has a predictive ability.

Insurance Expense Estimation

Let's consider yet another example. An insurance company charges premiums to its customers and in exchange, if any of the customers require to be hospitalized, helps them with their hospitalization expenses. Should the insurance company charge the same premium from all its customers? Or should the premium be dependent on the person's potential hospitalization costs? We know that healthcare costs statistically depend upon various factors such as age, lifestyle (smokes or not), area of residence (polluted versus not so), etc. If the insurance company had access to a model which could estimate average hospitalization expenses of a customer depending upon the customer profile, then it could use the estimate to decide upon the premium it charges to customers. This is yet another useful prediction problem, where the estimates of the model are essentially predictions on customer healthcare expenses based on many deciding factors.

Machine learning is a discipline which has developed a wide variety of models with powerful predicting capabilities which is helping us do many tasks which require prediction of some sort.

1.3.2 Predictive Models

There are various ways in which predictive models can be built and not all of them fall under the Machine Learning category. Before we dive deeper into Machine Learning, let us understand other types of predictive models briefly.

Numerical Models

Let's reconsider the weather prediction problem. Scientists have a fairly good understanding of how weather phenomena develop. For example, we know from our geography classes that during daytime, land heats up faster than the sea. Hence, air over land gets hotter, expands and becomes less dense, leading to low pressure compared to air over the sea. The pressure difference causes air from the high-pressure area over the sea to flow to the low-pressure area over land, leading to cool sea breezes from sea to land. Given the difference in pressures, the magnitude of the breeze will vary. There are physics equations which govern heating and cooling and the flow of the breeze as a function of the pressure

differences. All aspects of weather are governed by physics equations. Computer scientists and physicists and weather experts have come together to create weather models based on physics equations to predict the weather, from hours in advance to days to months in advance. These models are called numerical models **where the equations used in the model are known to govern the processes being modeled**. These models are far from perfect, the reason being that there are many factors determining weather over our huge earth. Thus, the models cannot include all these factors and lead to inaccuracies. However, these models have improved over the years to give useful predictions.

Other problems where numerical modeling is applied include planning trajectories of spacecrafts, missiles etc. These models are usually highly accurate with a very low margin of error.

But there are many problems where we do not know the equations governing the behavior that we want to model. For example, in image recognition, we do not know of equations by which objects in the image will be recognized from a given image profile. Same is true for the insurance cost prediction problem–we do not know what equations govern the cost. Hence, we need alternative ways of building predictive models for such problems.

Rule-based models

Let's consider the insurance cost prediction problem. As a first cut, we could have a model which predicts the insurance cost by age category. Customers between ages 10–20 may be charged a lower premium than those in the category 20–30, and so on. But we know that age is not the only determinant of health-care costs. For example, whether the person is a smoker or not will have an effect on the person's health-care costs. Again, the model could have a separate premium based on whether the person is a smoker or not. We can build rules such that the model would have separate estimates for people based on their age category, and whether they are smokers or not. If there are 10 age categories and the variable 'Smoker' can have the value 'Yes' and 'No', the model would generate 10 × 2 = 20 different estimates based on the user's profile. This kind of model is a rule-based model, since it applies rules based on some given parameters.

An example of a rule-based system for our insurance cost problem could be:

If (age between [10-20] and non-smoker) premium = p1
If (age between [10-20] and smoker) premium = p2
If (age between [20-30] and non-smoker) premium = p3
If (age between [20-30] and smoker) premium = p4

Figure 1.4 Rules for deciding insurance premiums

Now consider the situation where the number of categories increase, *e.g.*, the insurance company wants to decide on the premiums by age instead of range of ages. Now we have maybe 100 age categories instead of the initial few. Also let's say the insurance company wants to consider various other factors of the customer, including the customer's alcohol habits, the person's gender (assuming gender has an effect on costs), smoking habits, location, height, weight, family size, etc. Quickly, the number of possibilities and rules increases to hundreds, maybe even thousands. One can imagine that maintaining a system with thousands of rules will become error-prone and difficult. Therefore, a predictive system which can learn the rules from data might be easier to maintain and is often more accurate than rules-based systems.

1.3.3 ML Based Predictive Models

Machine learning models are in some sense a generalization of numerical and rule-based models with the added ability to learn the rules or equation parameters that govern the behavior of the predictive system that has to be created. It has been demonstrated that the algorithms learn the rules or parameters from past data better than humans can do–especially when the data (*e.g.*, images) is highly complex. Since the algorithms learn from data, the rules or parameters need not be generated or maintained by humans, making such systems inherently easier to develop and maintain.

ML models have come in to fill the gap in numerical and rule-based models and are being used where numerical or rule-based systems are not effective.

1.4 BOOK OUTLINE

To understand Machine Learning, one needs to have a thorough grasp of several math topics. We will cover these topics in the Appendix chapters. This includes basics probability and statistics. A knowledge of linear algebra and calculus is very important for understanding advanced algorithms used in Machine Learning, for example, gradient descent, but they are beyond the scope of this book.

To build Machine Learning systems, one needs to know programming. Standard programming languages such as C++, Java, etc. are not very popular for implementing ML solutions. Python and R have emerged as the two most popular languages for ML. We will use Python as the programming language of choice for this book. Several libraries of python have become indispensable in implementing ML solutions. We will discuss the most common of these–NumPy, Pandas, Matplotlib, and sklearn–in addition to the python language itself. There are books dedicated to some of these libraries such as pandas, let alone tomes on Python. What we will cover in this book is a small fraction of the language and its libraries and is aimed at making the code in this book easy to understand. However, some prior programming knowledge will help. Finally, it is expected of the reader that for pieces of code which have not been explained in the book, some effort will be expended in researching the topic on the internet.

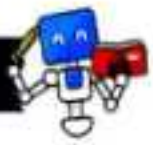
The reader is strongly advised to cover the Math and Programming topics in the Appendix before starting on the core chapters.

ML algorithms can be broadly divided into two categories called **supervised** and **unsupervised**. Supervised algorithms are further divided into two categories–**regression** and **classification**. Each category has many algorithms and it is not feasible to cover all of them. We have thus attempted to develop the intuition in each area with a few algorithms.

Data is the central ingredient for Machine Learning algorithms to work. Significant amount of time is usually spent in analyzing the data when embarking on a Machine Learning project. Some effort is spent in the book for discussing data analysis but the reader will have to refer to other sources for a broader exposure to data analysis techniques. Transforming data is critical for ML algorithms to

work well. Topics under the transformation category such as feature engineering and scaling are discussed.

Most of the concepts and algorithms are discussed with hands-on exercises including code and code output. We hope this will help in cementing the concepts discussed.

CHAPTER 1  EXERCISES

Review Exercises

1. Name some real-life applications of Machine Learning. What is Machine Learning doing within these applications?
2. What is the main difference between classical software development and a software which uses Machine Learning? Draw a diagram to illustrate this difference.
3. Why does Machine Learning work well on some problems for which traditional software built on rules (if-then-else) don't? Give an example to illustrate.
4. Predictive models can be of various types. What are the three main types of predictive models?
5. How can predictive models be useful? Explain with two different examples.
6. When are predictive models based on Machine Learning more suitable than other types of predictive models?
7. It is said that 'data' is the most important part of a Machine Learning application. Explain.

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#ml-al-revolution

2

Prediction Problems Using Regression

2.1 INTRODUCTION TO REGRESSION

As we have alluded to in the introduction chapter, machine learning algorithms can be broadly divided into two categories, supervised and unsupervised. In supervised learning, the dataset given for machine learning purposes has observed values of the variable for which a predictive machine learning model has to be built. Under supervised learning category, there are two sub-categories of problems called Regression and Classification. We will discuss regression problems in this chapter. Classification problems, and unsupervised learning problems will be discussed in later chapters.

2.1.1 Regression Problems

In this section, we present some representative data for a regression problem. We want to predict the cost incurred for a customer of a health insurance company. The customer's age and smoking habit is known-whether the customer is a smoker is indicated by 1 (is a smoker) or 0 (not a smoker). Several examples of

existing customers are available. In the table below, we have shown four such examples. For the fifth example, where the customer's age is 25 and the customer is a smoker, we want to estimate or predict the likely cost he/she will incur.

In this case, the outcome variable 'incurred cost' can have any value-arguably a positive one-from possibly 0 to some high number. Such variables which can have any value (possibly within a range) is called a **continuous variable.** Prediction problems where the variable to be predicted is continuous is called a **regression** problem.

Age	Smoker	Incurred Cost
23	1	82
27	0	75
30	0	85
35	1	111
25	1	?

Table 2.1.1 Customer health-care costs as a function of age and smoking habits

In the object recognition problem, we need to recognize objects in an image, for example we may need to identify if the object in the image is a car, a pedestrian, or a tree. In this case, the possible values of the outcome are discrete and not continuous. Problems where the outcome variable is discrete is called a **classification** problem. We will discuss classification problems in a separate chapter. In this chapter, we will discuss the regression problems in detail.

It should be mentioned that there are many types of problems which are regression problems. For example predicting company sales, predicting real estate prices, predicting crop yields, etc are all regression problems.

2.1.2 Elements of a Regression Problem

Target Variable

The variable to be predicted is called the target variable. For the insurance cost prediction problem, the *Incurred Cost* is the target variable. In case of the object recognition problem, the object in the image (possible values being pedestrian,

car, or tree) is the target variable. The target variable is also called the **dependent** or **outcome** variable. It is often denoted by the variable symbol *y*.

Feature Variable

The variables which are used to predict the target variable are called the feature variables. For the insurance cost problem, *age* and *smoker* are used to predict the target and are feature variables. For the object recognition problem, if the image is represented by a million pixels, then each of the million pixels is a feature. A feature variable is also referred to as the **independent** variable. It is often denoted by the variable symbol *x*. When there are many features, symbols $x_1, x_2, ..., x_n$ are typically used for them.

m (Example)

A collection of data which contains the values of the feature variables and the corresponding value of the outcome variable is called an example. In Table 2.1.1, we have 4 examples. We use the notation *m* to indicate the number of examples provided. In Table 2.1.1, we have $m = 4$.

2.2 REGRESSION MODELS

There are many models which can be used to model regression problems. The simplest one is called the linear regression model. There are non-linear models, generalized additive models (GAMs), and even tree-based models which can be used for regression problems. In this chapter, we will study linear models applied to regression problems in detail and will provide a limited introduction to non-linear models. In later chapters, we will study two more models which can be used for regression problems called K-Nearest Neighbors (KNN) and Decision Trees.

2.2.1 Linear Regression Model

A linear regression model for a dataset with one feature/independent variable is given below:

$$\hat{y} = \beta_0 + \beta_1 x_1 \qquad \textit{Equation } \mathbf{2.2.1}$$

A linear regression model with multiple features (n of them) is given by the equation below:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n \qquad \textit{Equation } \mathbf{2.2.2}$$

Why is this model a predictive model?

This is because given values of the features or independent variables (x's) of the model generate a value of the target variable ($\hat{y}$). This generated value is an estimate, and may not match the actual or observed value of y for the given value of x. Hence, we say it is an estimate or a prediction and is indicated by a ($\hat{y}$) instead of y which represents the actual value, when known. For example, for the 5th row in Table 2.1.1, if the x value is fed into the corresponding linear model, it will generate an estimate or prediction of the y variable.

Notations and parameters used in Linear regression models

Model Parameters

The constants (β_0, β_1, β_2) which determine the predicted value as a function of the independent variables are the model coefficients or parameters.

NOTE Alternate notations like (θ_0, θ_1, θ_2) are also used instead of (β_0, β_1, β_2)

Bias Parameter:

The model parameter (β_0) which is not a multiplicative factor for any of the independent variables is called the bias parameter.

Predicted or Estimated value:

A model provides a prediction or an estimate of the outcome variable. We will use the $\hat{y}$ (y-hat) notation to indicate that a variable is the predicted value. We will use the y notation to represent the actual or observed value of the outcome variable when it is present.

Why is this model a regression model?

The target variable $\hat{y}$ can take on a continuous range of values and is a continuous variable. Hence, it is a regression model.

Why is this model called a linear regression model?

This is because each independent variable appears with a degree of 1 (*i.e.*, we have terms such as x_1 and not x_1^2) and we add the combinations of each independent variable, *i.e.*, we have $\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$ instead of having products of the independent variables as in $\hat{y} = \beta_0 + \beta_1 x_1 * x_2$.

2.2 Building a Predictive Linear Regression Model

Let us consider the data given in Table 2.1.1. Let us assume we have decided to use a linear regression model to predict the outcome variable 'incurred cost'. Since we have two independent variables, what this means is that we have decided that we will use the linear equation below to build the predictive model where 'x_1' could represent 'age' and 'x_2' could represent 'smoker' while $\hat{y}$ represents the cost estimate for the outcome variable y.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 \qquad \textit{Equation } \mathbf{2.2.3}$$

By using some mathematical processes (which we will discuss later), let us assume that we determine the model parameter values which gives predictions that are close to the observed values of the available data, and the model then looks like:

$$\hat{y} = -10 + 3 * x_1 + 20 * x_2 \qquad \textit{Equation } \mathbf{2.2.4}$$

Since x_1 is the 'age' and the feature x_2 indicates whether the person is a smoker or not, we can thus also write the equation as:

$$\text{predicted cost} = -10 + 3 * \text{age} + 20 * \text{smoker} \qquad \textit{Equation } \mathbf{2.2.5}$$

The above model will predict costs as given in the table below for various values of the 'age' and 'smoker' variable. All these combinations of 'age/smoker' already exist in our given dataset of Table 2.1.1 along with the observed value of incurred

cost. The model of equation 2.2.5 has predicted a value for the age/smoker combinations which is close to the observed value but not necessarily exactly equal to the observed value.

age	Smoker	incurred cost (y)	predicted cost ($\hat{y}$) as per Eq 2.2.5
23	1	82	79
27	0	75	71
30	0	95	80
35	1	110	115

Table 2.2.1 Predicted [illegible] based on linear model for existing [illegible] examples).

We will discuss this difference between [illegible] and the predicted value, also called the error, in more detail later.

NOTE

The i^{th} value of a variable The i^{th} value of a variable is its value in the i^{th} example. We start the numbering from 0 usually. For our dataset given in Table 2.1.1 the 0th value of the age variable, *i.e.*, age_0 is 23, $smoker_0$ is 1, y_0 is 82, and $\hat{y}_0$ is 79.

Now that we have the model, we can use it to predict the outcome for combinations of the features variables we have not seen before, for example, as shown below.

age	Smoker	predicted cost ($\hat{y}$)
25	1	85
32	0	56
50	1	160
60	1	190

Table 2.2.2 Predicted healthcare costs for new data (value of outcome variable does not exist).

2.3 LINEAR REGRESSION MODEL AND MACHINE LEARNING

So where is the machine learning in what we did so far? We were given data about the outcome variable we want to build a predictive model for, along with data of related variables which determine the value of the outcome variable. *Determining of the model parameters from observed data is the learning part of machine learning.* After the model parameters are determined from past data, we can now use the model to predict the incurred cost for values of 'age' and 'smoker' which do not exist in the database.

NOTE

The machine learning problem thus, is to determine (or learn) the values of the model parameters from the given observed data, such that the model predictions ($\hat{y}$) are close to the observed values (y) of the outcome variable. In case of linear regression, the machine learning problem is to find the optimal values of the model parameters $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ of the linear model.

2.3.1 A Visual Representation of Linear Regression

We will explore visually what we are doing when we are using linear regression to build a predictive model.

x	y
0.300873	2.253513
0.186946	1.878124
0.323183	2.214512
0.665750	2.940856
0.566971	2.910902
0.398254	2.646285

Table 2.3.1 An independent variable x, and the dependent variable y.

For simplicity, we will assume that we have a single feature variable from which we need to build our predictive model. So, we are given several examples with x's and y's as given in the Table.

A plot of the variable x versus the variable y is provided below.

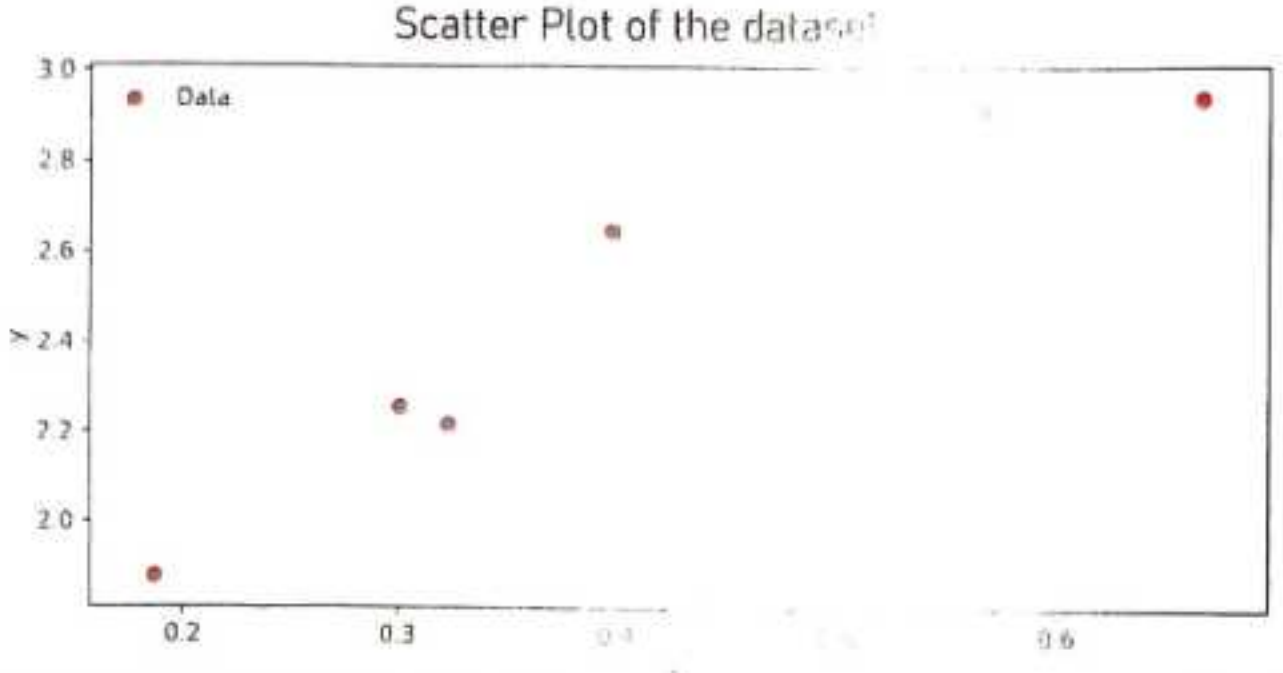


Figure 2.3.1 A plot of the data in Table 2.3.1.

When we decided that we will approximate the relationship between the x and y variables with a linear model, we essentially decided that we will use a straight line as the model. The two figures below shows three different straight lines which have been used to model the relationship between the x and y variable.

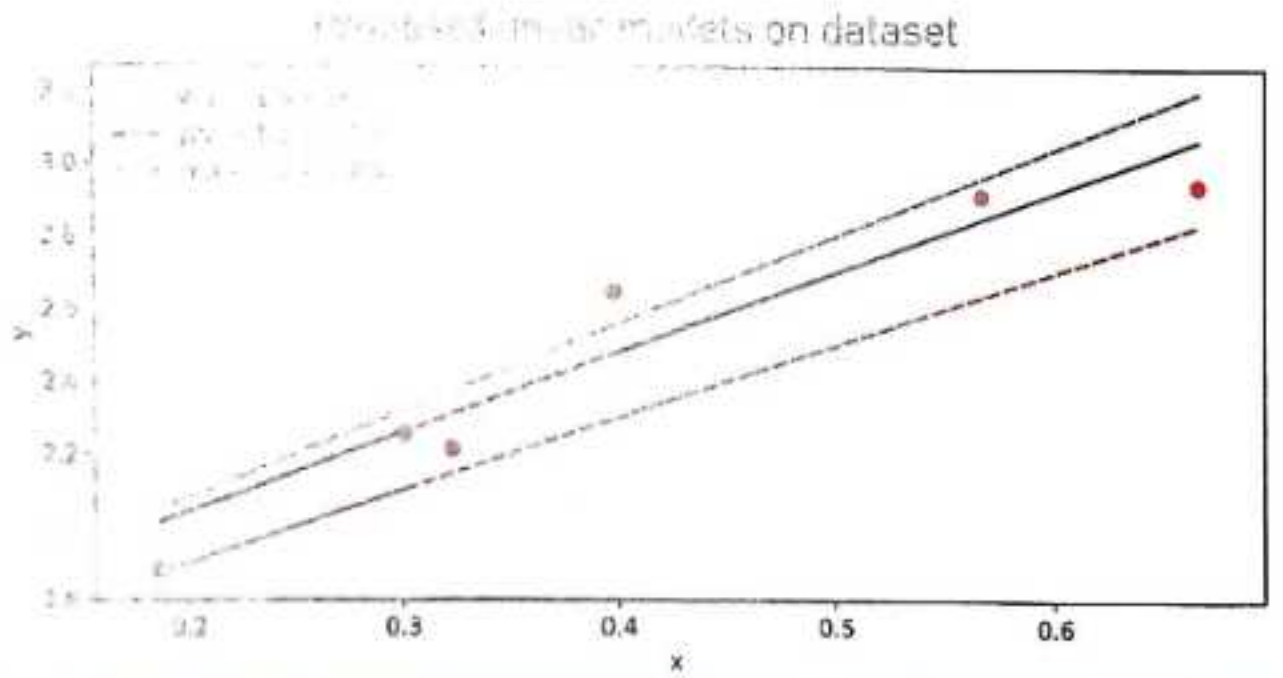


Figure 2.3.2 Three linear regression models for the data in Table 2.3.1.

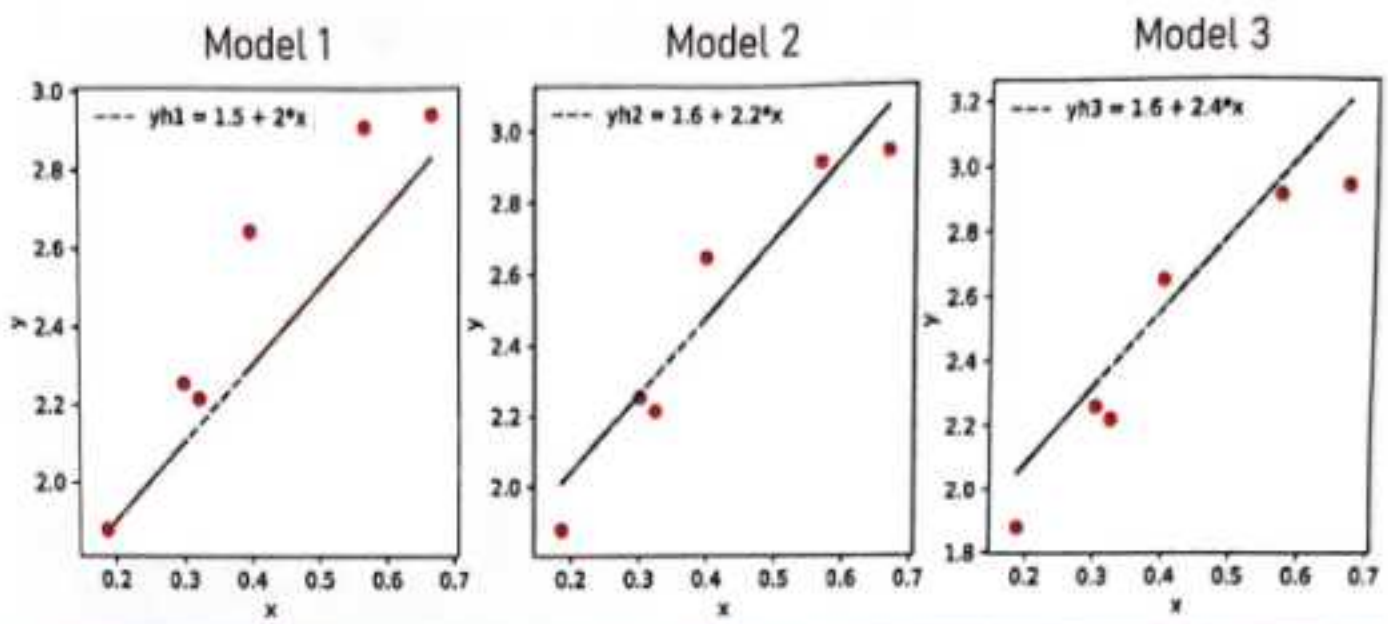


Figure 2.3.3 Three linear regression models shown separately for the data in Table 2.3.1.

By visually inspecting these models, the line corresponding to the linear regression model 2 which is $yh2 = 1.6 + 2.2x$ seems to be the better model for the given data. Model 1 seems to under-predicting the values (the actual points lie above the line mostly) and Model 3 seems to over predicting the values (the actual points mostly lie below the line). Model 2 seems have fewer errors of prediction and lies central to the data points. Hence, we may claim that Model2 is the 'line of best fit'.

The process of determining the model parameters which minimizes the differences or errors between the outcome variable and its estimate is the linear regression machine learning problem.

2.3.2 Model Error in Predictive Model Predictions

To understand how the model parameters are determined, we will need to understand the concept of model error. As we have seen before (refer to Table 2.3.1), in general, a predictive model will not be able to predict exact values of the outcome variable. So, there will be errors in the model predictions. We need to be able to measure the model error objectively if we are to be able to determine how good the model is. If we can provide an objective measurement, then we can use the same error measure to determine which model is optimal.

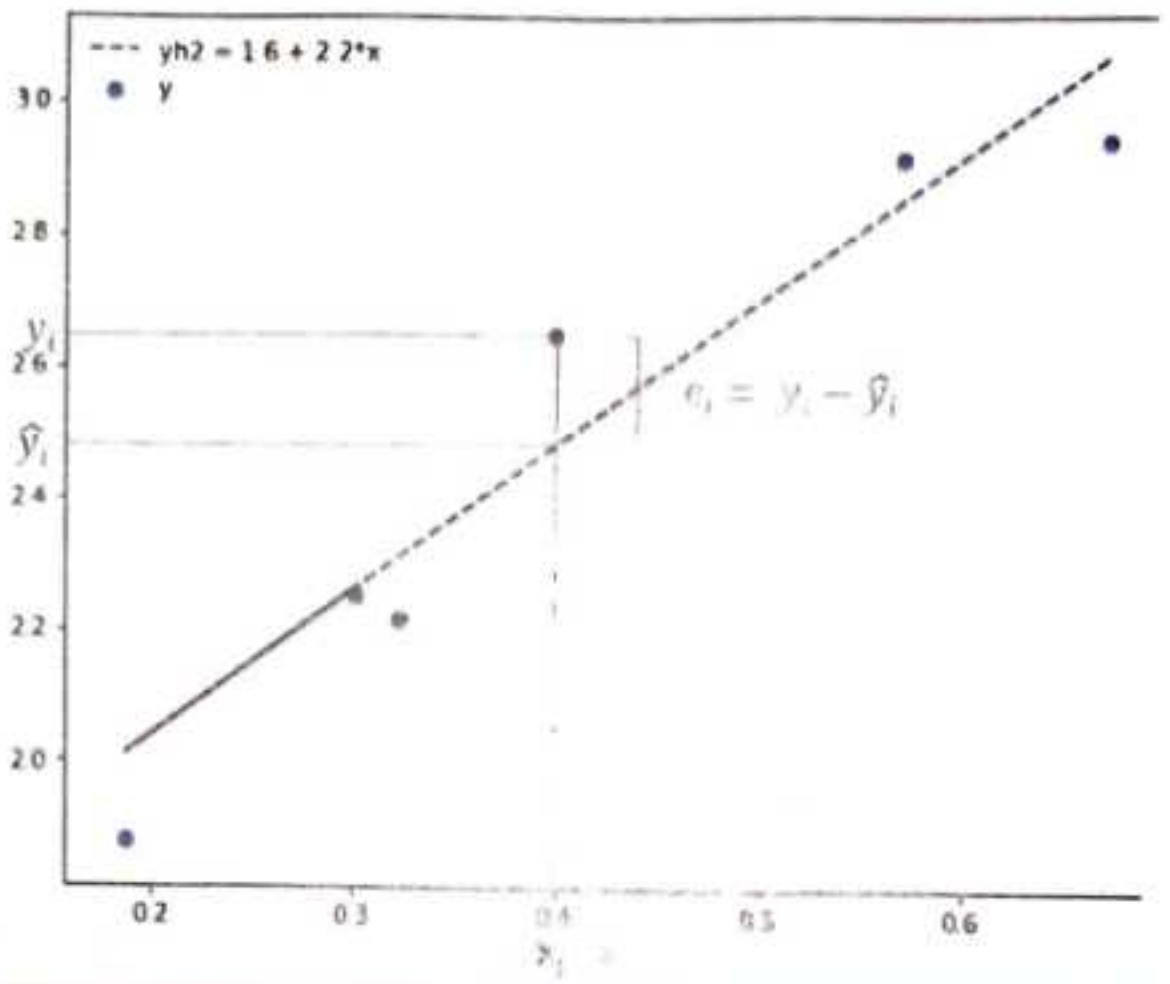


Figure 2.3.4 Illustrating model error

We reproduce the same data in Figure 2.3.2 along with one of the models $yh2 = 1.6 + 2.2x$ in Figure 2.3.4. Note that for x_i, the model predicts $\hat{y}_i$ while the observed value is y_i. The difference in this predicted value and the actual value is the model error e_i for the i^{th} example.

Instead of measuring the error as the difference, one can also use the square of the difference or the absolute value of the difference. The most common metric used to measure model error in case of regression problems is the Residual Sum of Squares (RSS).

$$RSS = \sum error_i^2$$

$$= \sum (y_i - \hat{y})^2 \qquad \textit{Equation } \textbf{2.3.1}$$

i.e. the error for all of the predicted values is summed up, where the error is represented by the square of the difference between the estimate and the observed value of the outcome variable.

NOTE

If we were to use the error metric $(y_i - \hat{y}_i)$ instead of the squared value, then the value can be negative for some examples and it can be positive for others. Then, if we were to sum the error $(y_i - \hat{y}_i)$ of all examples, then the positives and negatives could cancel out, giving an impression that the model error is less than it actually is. Hence, RSS uses the square of the model error $(y_i - \hat{y}_i)^2$ on all the examples.

Another error measure is the absolute value of the difference or $|\hat{y}_i - y_i|$. For technical reasons (the discussion is beyond the scope of this book), the RSS is preferred over the sum of the absolute errors.

2.3.3 Determining Optimal Model Parameters (aka Training the Model)

Recall that in Section 2.3, we mentioned that "the machine learning problem is to determine the values of the model parameters from the given observed data such that the model predictions ($\hat{y}$) are close to the observed values (y) of the outcome variable". The optimal model parameter values are those values which minimize the model error over all the examples provided. In case of regression, as we mentioned, the preferred metric for measuring model error is the RSS metric. **Hence, the regression problem is to determine the regression model parameters that minimize the RSS metric over all the examples.**

How does one go about determining the optimal values of the parameters so as to minimize the error metric? The algorithms vary depending upon the model. For linear regression, a technique called Stochastic Gradient Descent (SGD) is frequently used to determine the optimal model parameters values which minimize the RSS metric. A method called Singular Value Decomposition (SVD) is another method of choice. SVD uses techniques from linear algebra. Discussion of these techniques is beyond the scope of this book.

The process of determining model parameters from given data is also called **Building** or **Training** or **Fitting** the model to the data. It should also be remarked that the 'learning' of machine learning refers to learning or determining the model parameters from the data, hence **learning** in ML is the same as training or fitting the model.

2.4 EVALUATING MODEL QUALITY USING DIFFERENT METRICS

In prior sections, we discussed model error and the RSS as a metric for measuring model error. Now suppose that we, as humans, want to get a feel for how good our trained model is and whether the model errors are large or small. Can we use RSS for the same? The answer is no. There are other metrics available, of which we will discuss the **RMSE** and **R-squared** metric which are often used by humans to evaluate model accuracy. There is a third metric called the TSS metric which is required to understand the R-squared metric. Hence, we will discuss the TSS metric also in this section in addition to the RMSE and R-squared metric.

Why is the RSS metric not suitable to evaluate model accuracy? RSS is a cumulative error metric. If there are 1000 examples used to train the model versus 2000 examples, assuming similar errors for each example, the RSS in the second case can be expected to be double of that in the first case for the same model. However, it does not mean that the model is any worse in the second case compared to the first. Another reason why RSS is not suitable for general use is that RSS is a squared quantity. So, if we are predicting weights, say in grams, then RSS will be in grams squared which may be difficult to interpret. What we need is an average measure of the model error such that it is independent of the number of examples used to evaluate the model.

2.4.1 Root Mean Square Error (RMSE) Metric

The root mean square error or RMSE metric addresses both the problems of the RSS metric. The metric is given by the formula below:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \qquad \textit{Equation } \mathbf{2.4.1}$$

$$RMSE = \sqrt{MSE} \qquad \textit{Equation } \mathbf{2.4.2}$$

$$= \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}$$

So, to compute RMSE, we first take an average of the RSS by dividing the RSS by the total number of examples m. This quantity, called the Mean Square Error

or MSE is given by Equation 2.4.1. Once we get the average, we then take the square root of the average to remove the squaring of the errors in the RSS. The square root of the MSE is called the root mean square error or RMSE as given by Equation 2.4.2.

Hence, RMSE no longer grows with the number of examples since it is an average. Also, the squaring effect of the errors in the RSS are taken out by the taking the square root of the MSE.

Let's return again to the insurance cost prediction problem. Let us say that our model gives an RMSE of 1500 INR. Is that good or bad? To answer that question, we first need to find what are the typical values of the cost incurred by a patient. If a typical (mean or median maybe) cost is 20000 INR, then the RMSE is $\frac{1500}{20000} \times 100 = 7.5\%$ of the typical cost. If the typical cost is 30000 INR, then the RMSE is 5% of the typical cost. So, a more effective way of evaluating the error is using the **Normalized RMSE (NRMSE)** where:

$$NRMSE = \frac{RMSE}{\text{Average of Taet}} \times 100 \qquad \textit{Equation } \mathbf{2.4.3}$$

The R-squared metric is another very popular metric used for evaluating a regression model. Before discussing the R-square metric, let us first discuss a metric known as the Total Sum of squares or TSS.

2.4.2 TSS Metric

The TSS is defined as below:

$$TSS = \sum(y_i - \overline{y}_i)^2 \qquad \textit{Equation } \mathbf{2.4.4}$$

Where $(\overline{y}_i)$ corresponds to the mean value of the target variable.

Compare the formula for TSS with the formula for RSS in equation 2.3.1. Note that the $\hat{y}_i$ of the RSS has been replaced by $(\overline{y}_i)$ in the TSS. Thus, the TSS can be interpreted as the RSS for a model which always predicts the average value of the target variable (*i.e.*, $\hat{y}_i$ is always $(\overline{y}_i)$ for all i). A model which predicts the average value always is probably the simplest model one can imagine but not a very useful model.

2.4.3 R-squared (R2) Metric

Now let us look at the formula for the R-squared or R^2 metric. The metric is given by the formula below:

$$R2 = 1 - \frac{RSS}{TSS} \qquad \textit{Equation } \mathbf{2.4.5}$$

If the model RSS is close to the TSS value, then we can say that the model is as good as the model which predicts the average always (which is not a great model). Since RSS is close to TSS, RSS/TSS will be close to 1 and the R^2 in this case will be close to 0. So, if the R^2 value is small and close to 0, our model is comparable to a model which predicts the average value always and is arguably not very good.

On the other hand, if the model errors are very small, then RSS will be close to 0 and RSS/TSS will be close to 0 and R^2 in this case will be close to 1. A model with small errors is obviously a good and desirable model. So, if the R^2 value is close to 1.0, then our model is arguably a good one.

Finally, if the RSS value is larger than TSS, it means that the model performs worse than the average model, and in this case, our R^2 value can be negative.

R^2 is a very popular metric used for evaluating models and we look for high R^2 values close to 1.0 in a good model.

Recap of Steps in Linear Regression

Note that these steps are applicable whether we use a linear model for a regression dataset or some other possibly non-linear model. We will discuss some non-linear models briefly later on.

1. We are provided with data about the variable we need to build a predictive model for (target/dependent variable) as well as the variables the target variable depends upon (the features/independent variables).

2. We choose a general predictive model to represent the relationship between the independent and dependent variables.
3. We determine the parameters of the predictive model (also called training the model) using the given data.
4. We use error metrics such as RMSE and R^2 to determine the goodness of the model.
5. If we are happy with the accuracy of our trained model, we use the model to predict the outcome variable for unseen values of the feature variables.

2.5 HANDS-ON EXERCISE : INSURANCE COST PREDICTION PROBLEM

We revisit the data of an insurance company which contain customer profiles including the costs incurred by the customers, presumably over a one-year period [*Source:* This dataset has been used in various coding examples on Kaggle and is available on https://github.com/stedy/Machine-Learning-with-R-datasets/blob/master/insurance.csv. It was possibly sourced from the book - *Machine Learning with R by Brett Lantz, Packt Publishing.*] We want to develop a model for predicting the healthcare costs for new customers for whom the insurance company has the profile but not the costs incurred. **We decide to use a linear model. In real life, we will most probably try other models too.** We provide the complete Python code below to illustrate the likely sequence of steps that would be used to train, evaluate, and use the model.

Step 1

Load the Data.

In [1]:
```
# Load the dataset into pandas dataframe and print the sample of first 5 rows

import pandas as pd
df = pd.read_csv('../Data/insurance_data.csv')
df.head()
```

higher healthcare costs. It is important that when doing exploratory data analysis, intuitions regarding the data are confirmed. If the data does not seem to agree with intuition, then an investigation for the disagreement needs to be conducted.

(Optional)

Let us assume that the relationship between a given set of features and the target is indeed linear. However, when we plot the target versus each of the features, the plot may not show a clean linear relationship. The reason is as follows. Let the linear function be $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$. To see the linear relationship $y = \beta_0 + \beta_1 x_1$, x_2 must be kept constant. However, in the dataset that we are given, there is no guarantee that x_2 is constant while x_1 is being varied. Hence, y does not appear as a clean linear function of x_1 when the correlation plots are made.

Step 4.

Separate the Data into the Features and Target.

This will be required for training our linear regression model using available functions in a Python library. Since the first 6 columns contain the features, the X DataFrame is created from the first 6 columns of the dataset. Y is created from the last column and is technically a pandas Series object.

In [4]: ▸ *# Create the pandas dataframe with the features and the pandas series with the target.*

```
x = df.iloc[:,:6]
y = df.iloc[:,6]
```

Step 5.

Training the Linear Regression Model.

The Python Sklearn library has functions for many machine learning models. We simply need to know the particular name used for the model we are interested in. In our case, we are interested in the Linear Regression model. We first need to instantiate a copy of the model as in the code provided below. After instantiating it, the fit function can be invoked with the data available for training the model. Under the covers, the fit function finds the values of the linear model parameters or coefficients which will minimize the RSS value for the given data. In a third step which is not essential, the coefficients for the optimal model are printed out for each feature.

It seems that smoking and obesity has the most effect on the cost compared to other features. The negative coefficient for sex implies that as the value of sex increases (*i.e.*, goes from 0—male to 1—female), the costs actually come down.

In [5]:

```
from sklearn import linear_model

# Let's create an instance for the LinerRegression model.
lr = linear_model.LinearRegression()

# Train the model on our train dataset
lr.fit(x, y)

# Print the coefficients of the trained model.
coeffs = pd.DataFrame(lr.coef_, x.columns, columns=['Coefficient'])
coeffs
```

Out [5]:

	Coefficient
age	257.795444
bmi	125.493807
children	473.142002
smoker	23841.932222
obese	3007.932754
sex	-151.591317

(Optional)

Even though the coefficients indicate the nature of dependency of the target variable on the features, one should treat these coefficients with a bit of caution. If the scale of the parameters is changed, for example, if age is given in months and not years, the coefficient corresponding to age will be scaled down by a factor of 12. It does not imply that age has become less of a factor simply because it is specified in months. Other factors which influence the coefficient values are correlations between the features. If the features themselves are correlated, then the interpretation of the relationship between the features and the target cannot be directly given by the coefficients. Detailed explanation of this effect is beyond the scope of this book.

Step 6.

Predict on the dataset.

We use the trained model to see how it predicts on the dataset it was trained with. For that, we simply call the predict function with the features DataFrame X. We then create a DataFrame of the actual values and the predicted values. Creating such a DataFrame is not essential, but is used so as to display the comparison between the observed and predicted values succinctly.

```
In [6]: 1  # Getting predictions from the model for the given examples.
        2  predictions = lr.predict(X)
        3
        4  # Compare with the actual charges.
        5  Scores = pd.DataFrame({'Actual': y, 'Predictions': predictions})
        6  Scores.head()
```

Out [6]:

	Actual	Predictions
0	16884.92400	3647.497463
1	1725.55230	4513.901889
2	4449.46200	3641.510104
3	21984.47061	3611.169823
4	3866.85520	4128.298637

The predicted values do not seem very accurate. Only the 5th one corresponding to row 4 of the DataFrame looks reasonably accurate. We will improve the accuracy of the model in a later exercise.

Step 7.

Compute model accuracy metrics.

```
In [7]: # Let's evaluate the model for its accuracy using various metrics such as RMSE,
        # and especially R-squared.
        from sklearn import metrics
        import numpy as np

        print('RMSE:', np.sqrt(metrics.mean_squared_error(y, predictions)))
        print('Average Cost:', y.mean())
        print('R-squared', metrics.r2_score(y, predictions))
```

```
RMSE: 5988.526691650194
Average Cost: 13270.422265141257
R-squared 0.7552765462621057
```

The R^2 value is 0.75, which means that our model is far better than a model which always spits out the average cost as the predicted value. However, the RMSE is about 45% of the average cost, which is not that great.

Overall, the model does not seem to be good enough for the insurance company's purpose. We will see how we can improve the model very soon. However, the above exercise should give you an idea of how to build a simple predictive model using machine learning.

EXERCISES

Review Exercises

1. What makes a problem a regression problem? Give an example and explain what makes it a regression problem.
2. Explain the difference between a target variable and a feature.
3. A linear regression model is often used for regression problems. What are the characteristics of a linear regression model?
4. Differentiate between an observed value and a predicted value. Which kind of value does a regression model generate?
5. How do the features and targets of a dataset correspond to the components of a linear regression model?
6. When building a linear regression problem, which part of the process can be termed as learning?
7. Which error metric is used when training a linear regression model? Provide the formula and an explanation of what the metric measures.
8. Explain the RMSE and R^2 error metrics with their respective equations.

9. How can the R-squared metric be explained in terms of the model performance relative to a model which always predicts the average value of the target variable?
10. Which Python library provides machine learning models?

Investigative Exercises

1. Both regression and classification are supervised learning problems. Find out what supervised learning means.
2. The equation below relates distance covered (d) to initial velocity (u), acceleration (a), and time (t). Is the relationship linear? If yes, why? If not, why not?

$$d = u \times t + \frac{1}{2} \times a \times t^2 \qquad \textit{Equation 1}$$

3. Load the insurance dataset. Then multiply the values in the age column by 12. Rebuild the linear regression model and examine the value of the coefficients. Explain the changes you see in the coefficients compared to the first model which was built with the age values as it is. How do the error metrics for the new model compare with that of the first one?
4. Build another linear regression model, but without the *bmi* feature. What is the effect on the model accuracy?
5. In the correlation graphs shown as part of the hands-on exercise, the scatter plot of *charges* versus *smoker* appears in two vertical lines, one for the value 0 for the 'smoker' variable and the other for the value of 1 for the variable. There are charges for smokers which are less than that for non-smokers. What could be the reason for this?
6. The RSS metric uses a square term to measure the error in the model predictions. Suppose the errors on a set of four examples with model 1 is 1, 2, 3, and 4. The errors with model 2 is 1, 3, 3, and 3. The errors with model 3 is 1, 1, 3, and 5. If we were to use the sum of the errors without squaring the error terms, which model would be preferred? If we were to use square terms (*i.e.*, RSS) to evaluate the models, which one would we then prefer? What does this say about the use of RSS as an error metric?

7. For the insurance cost prediction problem, the dataset had six features. What other features could be considered to improve the model accuracy? Could some of the existing features be refined to improve model accuracy?

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#regression

3

Non-Linear Models and Feature Engineering

3.1 NON-LINEAR MODELS

You may recall from your high school days the equation below:

$$s = \frac{1}{2} * a * t^2 \qquad \textit{Equation } \mathbf{3.1.1}$$

Where a is the acceleration of an object (of your car maybe), t is the time for which the object has been accelerating, and s then is the distance covered by the object in the time t. In this case, a and t are independent variables (features in machine learning terminology), while s is the dependent or target variable. A linear model for features a and t would have been of the form:

$$s = \beta_0 + \beta_1 \times a + \beta_2 \times t \qquad \textit{Equation } \mathbf{3.1.2}$$

Clearly, equation 3.1.1 is not linear. It is not linear because it has a term which is the square of one of the independent variables (t^2) and further, the square of the variable is multiplied by the second independent variable a.

3.2 FEATURE ENGINEERING

Feature engineering is the task of deriving new features from initial features such that the derived features can be used to build a model with higher accuracy. If we were to use the example above of distance covered as a function of acceleration and time, the initial features would be *a* and *t*, while the derived feature would be a^*t^2. Depending upon the application, we may want to use the initial features as well as the derived feature in building our model.

How do we know that we have to use a linear model or a non-linear model to model the relationship between the features and the target variable? Or how do we know if we need to do feature engineering? If we knew the nature of the relationship between the features and the target, then we would know what to do. But in most cases, we do not know the relationship between the variables for which we want to build a model. There are a few ways in which we can try to figure out the relationship. One way is to plot the (*x*, *y*) values and visually detect whether the relationship is linear (straight line) or non-linear plot would not resemble a straight line).

A second, possibly easier approach is to use models which are flexible enough to model linear as well as non-linear relationships. Tree-based models, which we will discuss in a forthcoming chapter, have this flexibility and they tend to automatically pick up the non-linearity if it exists. Artificial Neural Networks (ANN) are yet another category of models with enormous flexibility and they are at the basis of the most cutting-edge ML applications.

None of the above methods are foolproof for reasons whose discussion is beyond the scope of this book. In general, (a) a combination of visual inspection to detect non-linearity, or (b) trial and error feature engineering, and (c) models capable of automatically modeling non-linear relationships may have to be used finally.

We will, through a manual analysis, show how non-linear relationships can be detected and hence modeled in the hands-on exercise below.

3.3 HANDS-ON EXERCISE: INSURANCE COST MODELING PROBLEM

We will go back to our insurance cost modeling problem. When we had plotted the correlation between the different features and the target, we had noticed a very interesting pattern with the 'age' feature. We reproduce that scatter plot below.

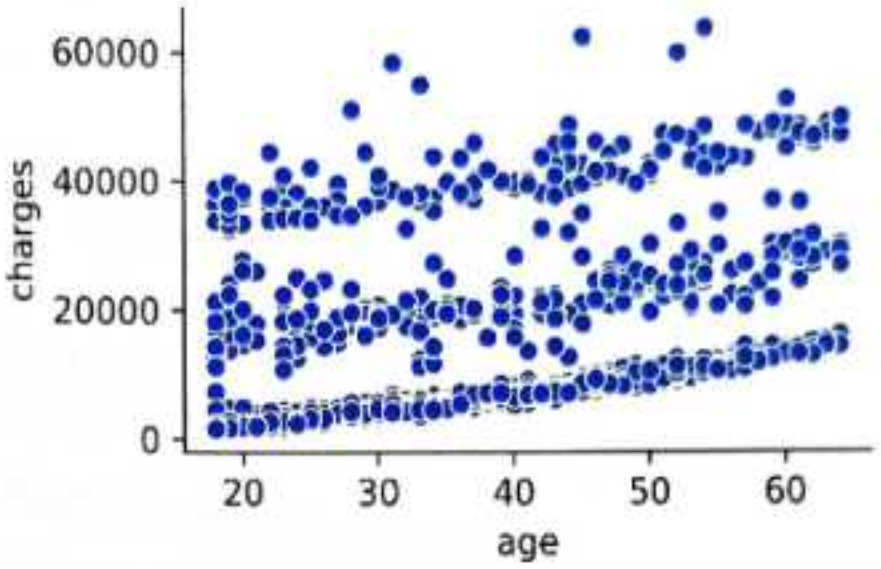


Figure 3.3.1 Age versus charges plot for the insurance dataset

From examining the figure, it seems that there are three separate linear relationships in the data. What could be going on? We notice that for the lowermost set of points, the costs seem to be roughly upto 15000, for the group of points in the middle, costs seem to be between 15000 and 30000, and for the uppermost set, the costs seem to be more than 30000. In an attempt to analyze the patterns, we decide to look at the values of the features other than 'age' for each of these three ranges of charges. The code below explores the impact of smoking and obesity along with age and gives us more insights into the dataset.

Step 1.

Read the data.

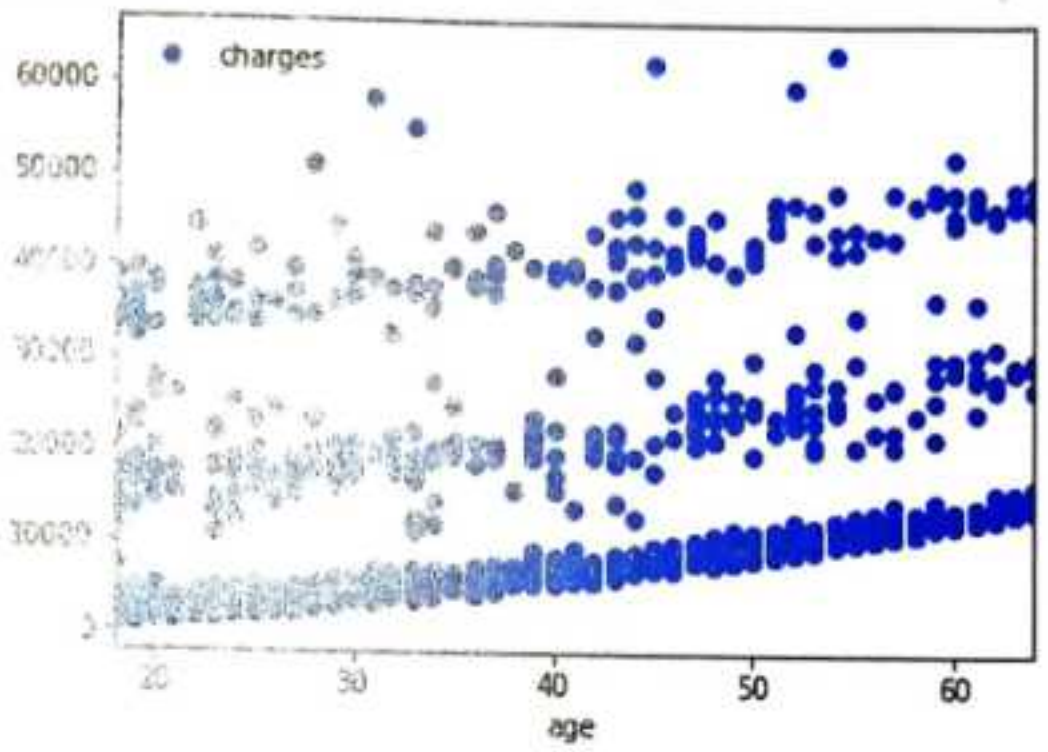
In [1]:
```
# Load the dataset into pandas dataframe and print the sample of first 5 rows

import pandas as pd
import numpy as np

df = pd.read_csv('../Data/insurance_data.csv')
```

Step 2.

Plot the Charges versus Age to examine the aforementioned pattern.

```
In [2]: # Lets see how the charges vary with the age.
        %matplotlib inline
        df.plot(x='age', y='charges', style='o')
```

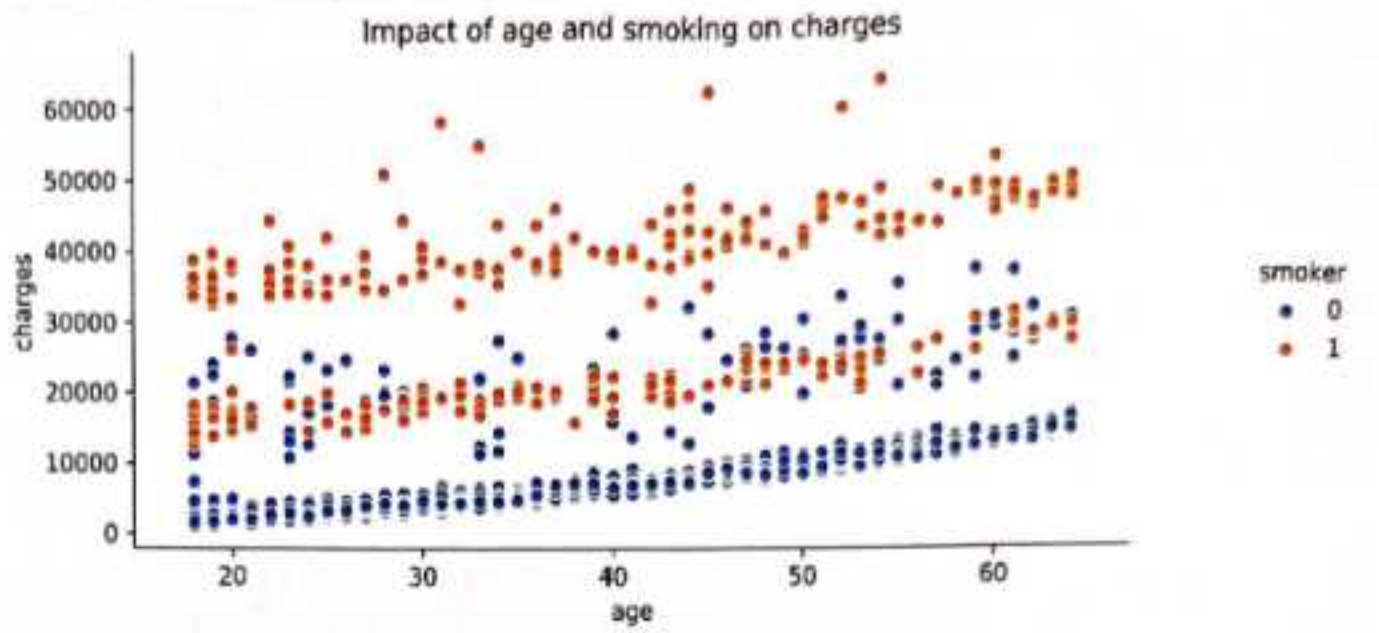
Out [2]: <matplotlib.axes._subplots.AxesSubplot at 0x18db5820e10>



Step 3.

Examine the impact of Smoking and Age together.

```
In [4]: # Explore impact of Age and Smoking together

        sns.pairplot(data = df[['age', 'bmi', 'children', 'smoker', 'obese', 'sex', 'charges']],
                     x_vars=['age'], y_vars='charges', aspect=1.5, hue='smoker')
        g.fig.set_size_inches(9,4)

        plt.title('Impact of age and smoking on charges')
        plt.show()
        g.savefig('fig_3_1_1.jpg', dpi=1200)
```

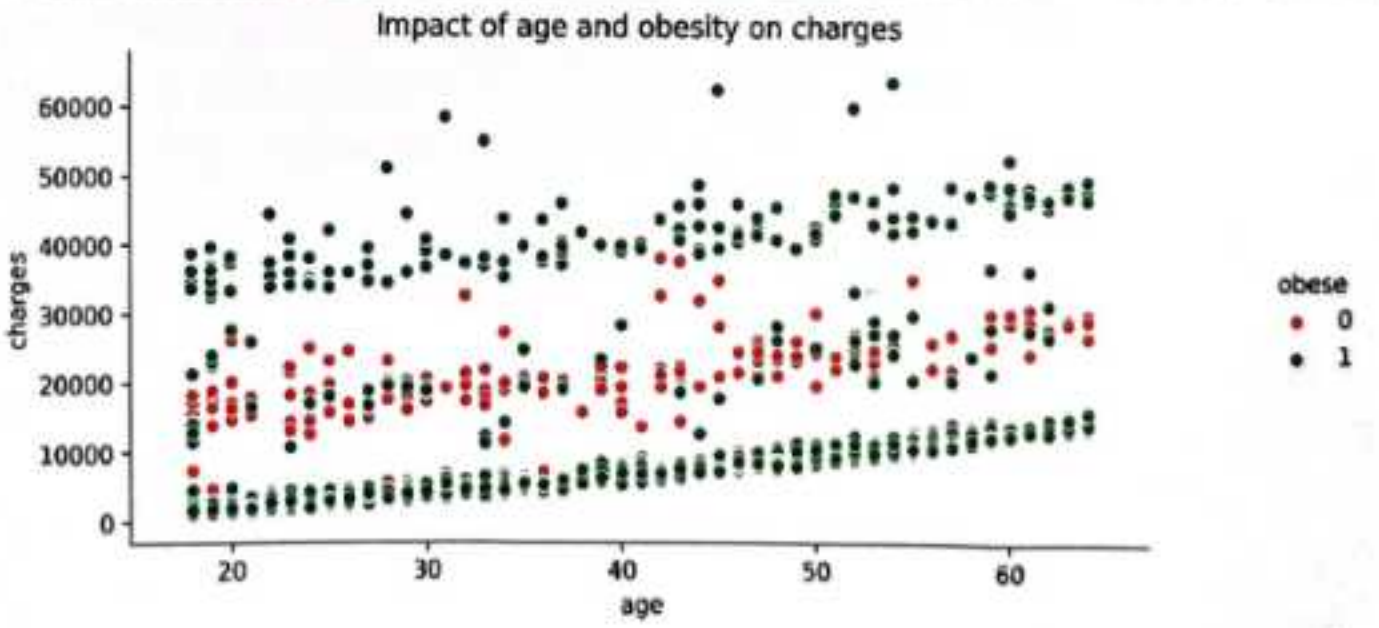


We see some very interesting patterns when we look at the 'smoker' and 'age' features. In the pairplot above, we have given a different color to the smokers (orange) and a different color to the non-smokers (blue). It becomes quite apparent that the persons with the highest charges (topmost band) are almost all smokers.

Step **4.**

Examine the impact of Obesity and Age together.

```
In [5]: 1 g=sns.pairplot(data = df[['age', 'bmi', 'children', 'smoker', 'obese', 'sex', 'charges']],
        2                 x_vars=['age'], y_vars='charges', aspect=1.5, hue='obese', palette='hus1')
        3 g.fig.set_size_inches(9,4)
        4 plt.title('Impact of age and obesity on charges')
        5 plt.show()
        6 g.savefig('fig_3_1_2.jpg', dpi=1200)
```



In this pairplot, a different color has been given to persons who are not obese (red) and a different color for those who are obese (green). We are able to see the role of obesity impacting the topmost layer of data which have the highest charges. In fact it is quite apparent that both smoking and obesity is impacting the charges the most.

How do we capture the additional costs of the group of people who both smoke and are obese? Mathematically, a term which is a product of the two terms can be used. Hence, to model the additional costs of those who are both smokers and obese, we may want to add a new feature (smoker * obese), much like the at^2 used in equation 3.1.1. Thus, our model would look as shown below:

$$charges = \beta_0 + (\beta_1 * age) + (\beta_2 * bmi) + (\beta_3 * children) + (\beta_4 * sex) + (\beta_5 * obese) + (\beta_6 * smoker) + (\beta_7 * obese * smoker)$$ *Equation* **3.3.1**

Step 5.

Add the extra feature to our dataset.

In [5]:
```
# We create a new feature which is the product of the smoker and obesity features.
df['smokOb'] = df['smoker'] * df['obese']
print("Number of customers who are both obese and smoke:", df[df.smokOb == 1].shape[0])
print("Total number of customers", df.shape[0])
```

```
Number of customers who are both obese and smoke: 144
Total number of customers 1338
```

We note that there are 144 people or roughly 10% of the customers who both smoke and are obese.

Step 6.

Create the features **DataFrame X and the series for the target variable.**

In [6]:
```
# Create the pandas dataframe with the features and the pandas series with the target.

X = df[['age', 'bmi', 'children', 'smoker', 'obese', 'sex', 'smokOb']]
y = df['charges']
```

Step 7.

Train the model with the additional feature.

In [7]:
```
from sklearn import linear_model

# Let's create an instance for the LinerRegression model
lr = linear_model.LinearRegression()

# Train the model on our train dataset
lr.fit(X, y)

# Print the coefficients of the trained model.
coeffs = pd.DataFrame(lr.coef_, X.columns, columns=['Coefficient'])
coeffs
```

Out[7]:

	Coefficient
age	264.217528
bmi	91.801444
children	505.007978
smoker	13434.330529
obese	-734.023995
sex	-467.296184
smokOb	19860.775096

Very interestingly, the coefficient for 'obese' has now turned negative. Does it mean that higher obesity means lower insurance cost? Not so. So why is this happening? Note that the new feature we have added, 'smokOb' has a huge coefficient. This feature is modeling the effect of obesity to a large extent, so much so that the obesity feature by itself has to be given a negative weight to balance the huge coefficient that the combined feature is being given. We have seen before that the feature 'smoker' had a very big influence on insurance costs (refer to the coefficients for the original model). So, in spite of the added feature 'smokOb' having a huge coefficient, the 'smoker' feature still requires a positive coefficient, albeit a smaller one than the one required for the first model.

Step 8.

Predict on the dataset.

In [8]:
```
# Getting predictions from the model for the given examples.
predictions = lr.predict(X)

# Compare with the actual charges.
Scores = pd.DataFrame({'Actual': y, 'Predictions': predictions})

Scores.sample(5)
```

Out[8]:

	Actual	Predictions
1332	11411.68500	13901.434399
468	23288.92840	5413.308344
1198	6393.60345	8578.842764
433	12638.19500	13196.570184
134	2457.21115	3204.453590

We see that the new model seems to be doing much better than the first model.

Step 9.

Compute model accuracy metrics.

In [9]:
```
# Let's evaluate the model for its accuracy using various metrics such as RMSE,
# and especially R-squared.
from sklearn import metrics
import numpy as np

print('MAE:', metrics.mean_absolute_error(y, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y, predictions)))
print('Average Cost:', y.mean())
print('R - squared:' , metrics.r2_score(y, predictions))
```

```
MAE: 2452.4664569912807
RMSE: 4460.511762340599
Average Cost: 13270.422265141257
R - squared 0.864229633041254
```

We see that all the metrics have improved significantly.

We talked about adding non-linear features (*smoker * obese*) to our model. However, we ended up using the Linear Regression function from the Sklearn library to build our model. Why did we do that?

Note that our new model is:

$$charges = \beta_0 + (\beta_1 * age) + (\beta_2 * bmi) + (\beta_3 * children) + (\beta_4 * sex) + (\beta_5 * obese) + (\beta_6 * smoker) + (\beta_7 * obese * smoker)$$

This model is not linear since it has a non-linear term *obese*smoker*.

But we added a new feature to our dataset 'smokOb' by multiplying the 'smoker' column with the 'obese' column to create a new feature '*smokOb*'. Using this new feature, we could thus rewrite our model as below, which is a linear model:

$$charges = \beta_0 + (\beta_1 * age) + (\beta_2 * bmi) + (\beta_3 * children) + (\beta_4 * sex) + (\beta_5 * obese) + (\beta_6 * smoker) + (\beta_7 * smokOb)$$

So, using feature engineering, we generated a new feature. When the new feature was added to our model, it again became linear. Hence, we could use linear regression model for our new model.

Feature engineering is usually considered as an art and as mentioned earlier, visualization may be used to determine the new features to be created.

CHAPTER 3 EXERCISES

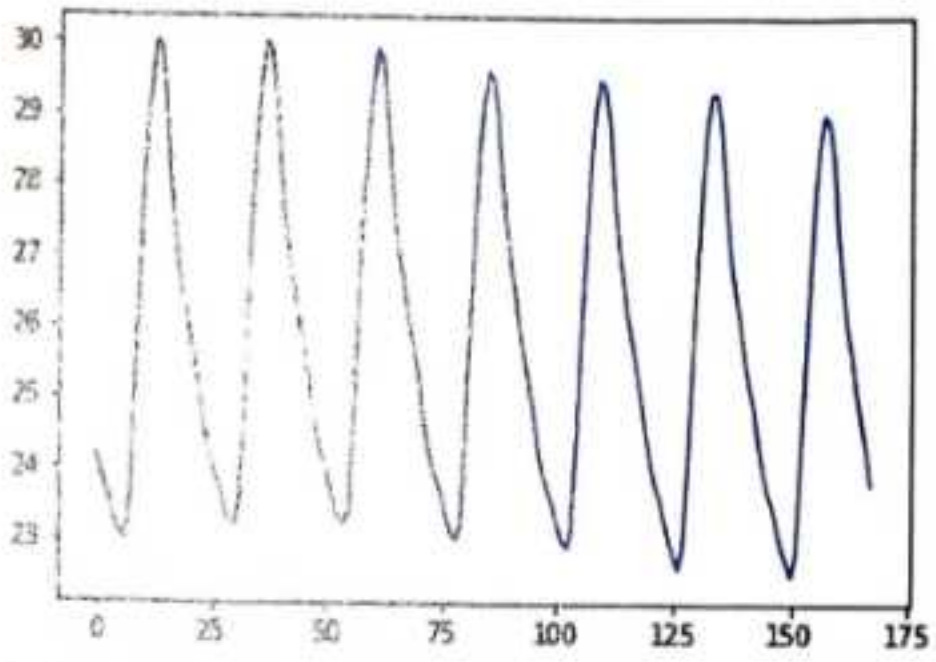
Review Exercises

1. When is a model non-linear? Give an example.
2. What is feature engineering?
3. In a parametric regression model (such as linear or non-linear regression), the coefficient for a feature is a measure of the multiplicative effect the feature has on the target variable. Consider a case where a feature has a positive multiplicative effect on the target, but the coefficient for the feature is negative. How is this possible?

4. How can use linear regression for non-linear models?

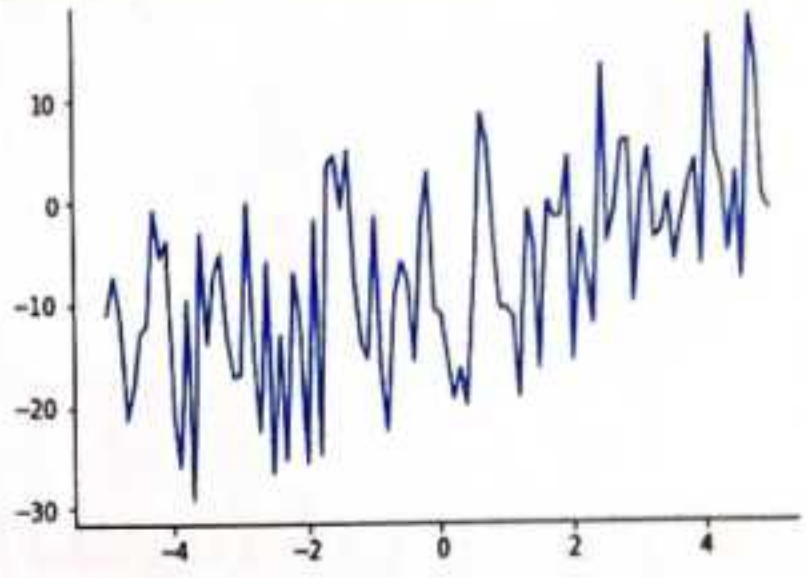
Investigative Exercises

1. The graph below shows the mean hourly temperature in the city of Chicago between 1st May, 2010 and 7th May 2010. The data is available from NOAA (www.noaa.gov). What kind of function can be used to model the relationship between the temperature and the day of the year? Different approximations are possible, hence there is no one correct answer for this question.



2. Study the code below and the corresponding graph.

```
In [14]: import numpy as np
         import matplotlib.pyplot as plt

         # y = 2x1 - 3x2 + 5
         x1 = np.arange(-5, 5, 0.1)

         np.random.seed(50)
         x2 = np.random.rand(100,) * 10

         y = 2*x1 -3*x2 + 5
         plt.plot(x1, y)
         plt.show()
```



It is clear that y is a linear function of x1 and x2. However, the plot between x1 and y is anything but linear (though a linear trend is discernible). Why is the plot not showing y as a linear function of x1?

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#feature-engineering

4

Sources of Model Error

4.1 REASONS FOR MODEL ERRORS

We have already been exposed to the fact that a model will have errors. The next two chapters will explore model errors in more detail and will provide the intuition behind more realistic processes for estimating the model error.

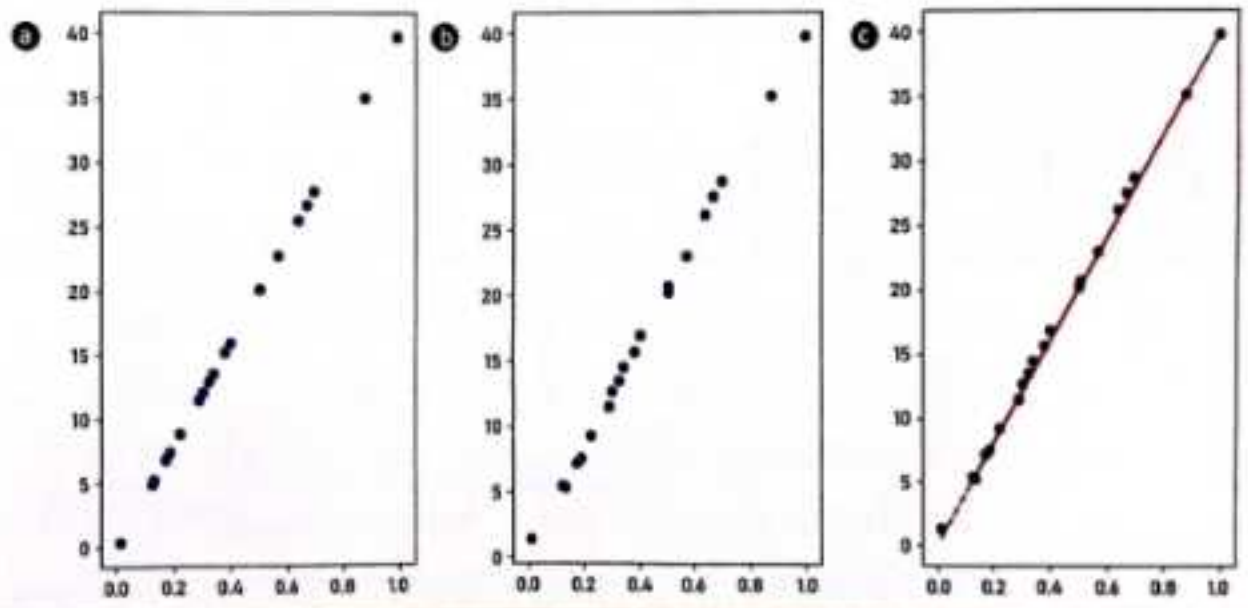


Figure 4.1 (*a*) A plot of time versus distance covered for a fixed speed assuming exact measurements (*b*) Same plot in (a) but assuming errors in measurement which is more realistic (*c*) A best fit linear model for the observations in (b).

In this chapter, let's look deeper into the sources of model error. We provide four potential reasons behind model errors (not necessarily in any order of importance).

4.1.1 Measurement errors

The first reason we will explore is measurement errors. The datasets that are used to build models consist of measured data. If the data is from sensors (*e.g.*, pressure, temperature, etc.), even if the sensors are accurate and well calibrated, they will still have measurement error. If the data is collected manually (*e.g.*, age, cost, etc.), there will almost always be a certain amount of rounding that will be used or there could be manual errors. For example, if the age is 59 years and 7 months, it may be reported as 59 years. In a good system, the errors are usually small, but will be there. Consider the relationship between time and distance covered. If speed is constant, then the relationship is linear as in:

$$\text{distance} = \beta \times \text{time}, \ \beta \text{ is average speed} \qquad \textit{Equation } \mathbf{4.1.1}$$

If the measurements of time and corresponding distance covered were exact, we would get the graph in Figure 4.1.1 (a). However, if there were small errors in the measured quantities, we would get the graph in Figure 4.1.1 (b). The second graph is more realistic and reflects imperfect measurements in the real world. If we were to express the relationship in Figure 4.1.1 (b) with a linear model, we would get the straight line shown in Figure 4.1.1 (c). The model has errors, but the errors are due to measurement errors rather than any intrinsic problem with the model itself.

4.1.2 Sub-optimal Model Parameters

Recall that in ML, we first assume a certain model, for example a linear regression model. Then, the model parameters are computed from the dataset so as to minimize the model error. We did not provide details of how the optimal model parameters are computed, except for mentioning that the model parameter computation process assumes an error metric such as RSS for minimizeation of the error. The computation processes are essentially optimization techniques to minimize the error metric and they work by changing the model parameter values in efficient ways to arrive at the optimal values. Depending upon the error metric we choose (*e.g.*, RSS), and the efficacy of our method which minimizes the error metric (*e.g.*, efficacy of SVD or Gradient Descent). The model parameters determined may be closer to optimal values or may not be close enough. Usually, for simple models like linear regression, error optimization techniques produce

parameter values which are very close to the optimal value. However, for more complex models which we will explore later on, we may have to put in some extra work to determine parameter values which are close to optimal. But it should be understood that some part of the model error may be due to the fact that the model parameter values that have been determined by the ML algorithm are not actually optimal due to sub-optimality of the model determination algorithm.

4.1.3 Sub-optimal Model

Consider the scatter plot in Figure 4.1.2 (a) between an independent variable *x* and a dependent variable *y*. The actual relationship is:

$$y = 2.5 + X + 0.5 * x^2 \qquad \textit{Equation } 4.1.2$$

To simulate measurement errors, we have added some noise to the *y* values as in:

$$y = 2.5 + X + 0.5 * x^2 + noise \qquad \textit{Equation } 4.1.3$$

The noisy but realistic scatter plot is shown in Figure 4.1.2 (a). In figure 4.1.2 (b), we fit the data using the model in Equation 4.1.2. In Figure 4.1.2 (c), we use a linear model to fit the data. We can see that our linear model does not fit the data well, and hence will have much higher errors than our polynomial model of Figure 4.1.2 (b).

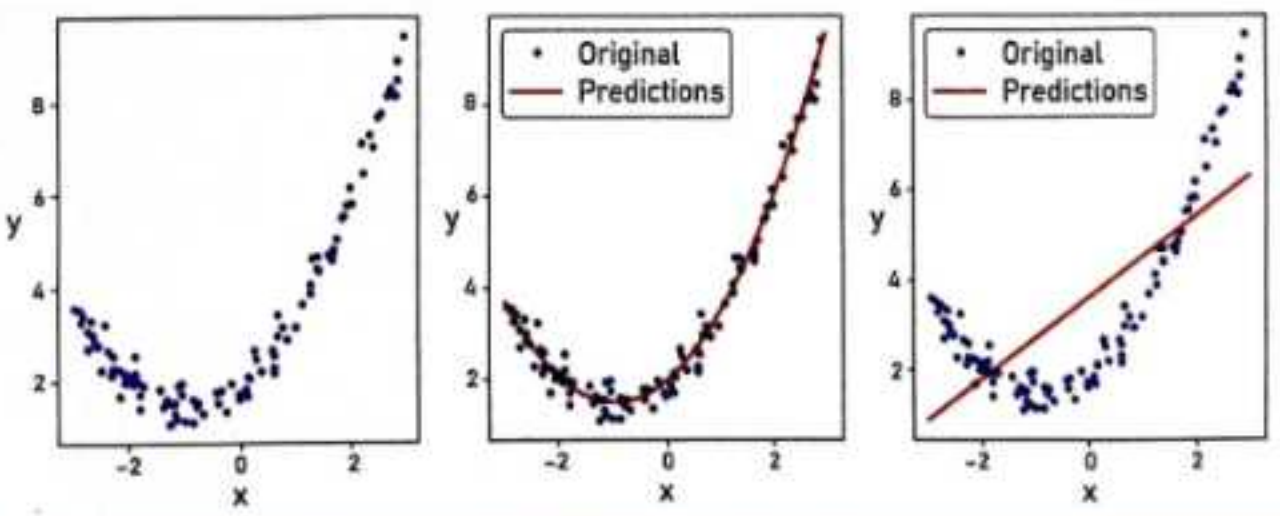


Figure 4.1.2 (*a*) A plot of the equation. (*b*) A second-degree polynomial fit of the data in (a). (*c*) A linear model fit for the data in (a).

As mentioned earlier, figuring out the best model (linear, non-linear, and the variations within) is not straightforward. If the model we select is not the most appropriate one to represent the inherent relationships in the data, then no matter how well we determine the model parameters, the model will have errors.

4.1.4 Missing Features

With the insurance cost modeling exercise, we have seen that the insurance costs have a very strong dependency on whether the person is a smoker and is obese, and lesser so on the other features such as age, bmi, etc. What if the customer profile did not even include whether the customer was a smoker or whether the customer was obese? It should be obvious that missing features which influence the outcome variable can reduce model accuracy. The more important the feature is, the more significant its absence will be on model accuracy. In most modeling problems, we assume that all the relevant features are available. However, in many real-life scenarios, we may not have the data on some features which we know influence the outcome, or sometimes, we may have missed features of importance completely. Usually, a domain expert has a good understanding of the features which influences the outcome variable. It is crucial that such domain experts are consulted with to decide if all relevant features are accounted for in the data.

4.2 RECTIFICATION OF MODEL ERRORS

What can you do about model errors?

1. Since one of the sources of the model error is a sub-optimal model (*e.g.*, linear selected while relationship is non-linear), one should try the many machine learning models that have been developed by scientists. We will explore several more models in this book. However, there are many more out there than we cover in this book and ML engineer should get their hands dirty with as many of the different models as possible.
2. If the source of model error is sub-optimal model parameters, then one can try tweaking the optimization technique. Discussing how to do so is beyond the scope of this book.
3. If the source of the error is missing features, then obviously, data for the missing feature needs to be collected, else this component of the error will not go away.
4. If the model error is due to error in measurements, then what should you do? The quick answer is to improve accuracy of measurements since it is not a modeling problem. But the problem is, we often do not know if the model

error is due to inherent measurement errors or due to the other sources. So, if we are not careful, we may end up trying to modify our model to remove measurement errors which can, in turn, lead to the model performing poorly on new data. We will discuss this issue in detail in chapter 5.

CHAPTER 4  EXERCISES

Review Exercises

1. A predictive model has errors. Explain why.
2. What is the source of sub-optimal model parameters? What are some ways of addressing the issue of sub-optimal model parameters?
3. The model can have errors because the model itself is not appropriate. Explain.
4. What are missing features and how can they contribute to prediction errors?

Investigative Exercises

1. In a sensor-based application, what source of error would be very common? How can the effect of the error on the predicted target be estimated?
2. For a credit scoring application, the data consists of the person's age, number of dependents, income, and credit history. Consider how this data may be collected. Where could the errors creep in? How could the magnitude of the errors be quantified?

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#sources-of-model-error

5

Overfitting and Underfitting (Bias and Variance)

5.1 INTRODUCTION TO OVERFITTING AND UNDERFITTING

In the previous chapter, we have seen that the training dataset might have errors, for example, errors due to measurement. When we try to identify a function which models the relationship between the features and the outcome, and we find that the function starts to model the errors in the dataset, we have a condition called 'Overfitting'. Overfitting causes the model to perform poorly on unseen data points. Another condition occurs when the function chosen is not powerful enough to model the relationship between the features and the outcome variable. This situation is called 'Underfitting'. Underfitting also causes the function to perform poorly on new datasets. We will discuss these two conditions in this chapter.

Consider the graphs plotted below in Figure 5.1.1 (a). It is the graph of the function in Equation 5.1.1:

$$y = 2.5 + x - 0.5 * X^2 + noise \qquad \textit{Equation } \mathbf{5.1.1}$$

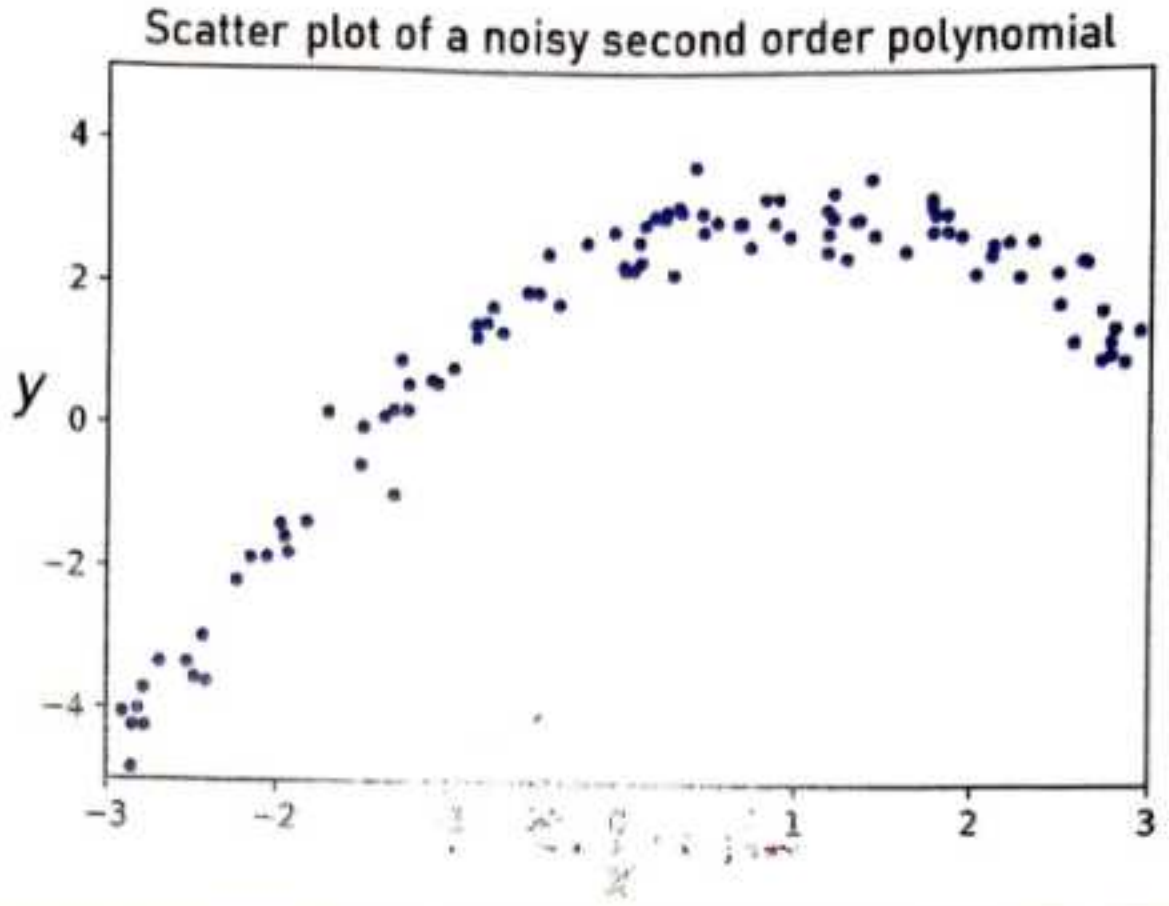


Figure 5.1.1 (a) A scatter plot of a noisy second-degree polynomial

The data has been fitted with three polynomials:

(a) A polynomial of degree 1 in Figure 5.1.1 (b), *i.e.*,

$$y = \beta_0 + \beta_1 \times X \qquad \textit{Equation } \mathbf{5.1.2}$$

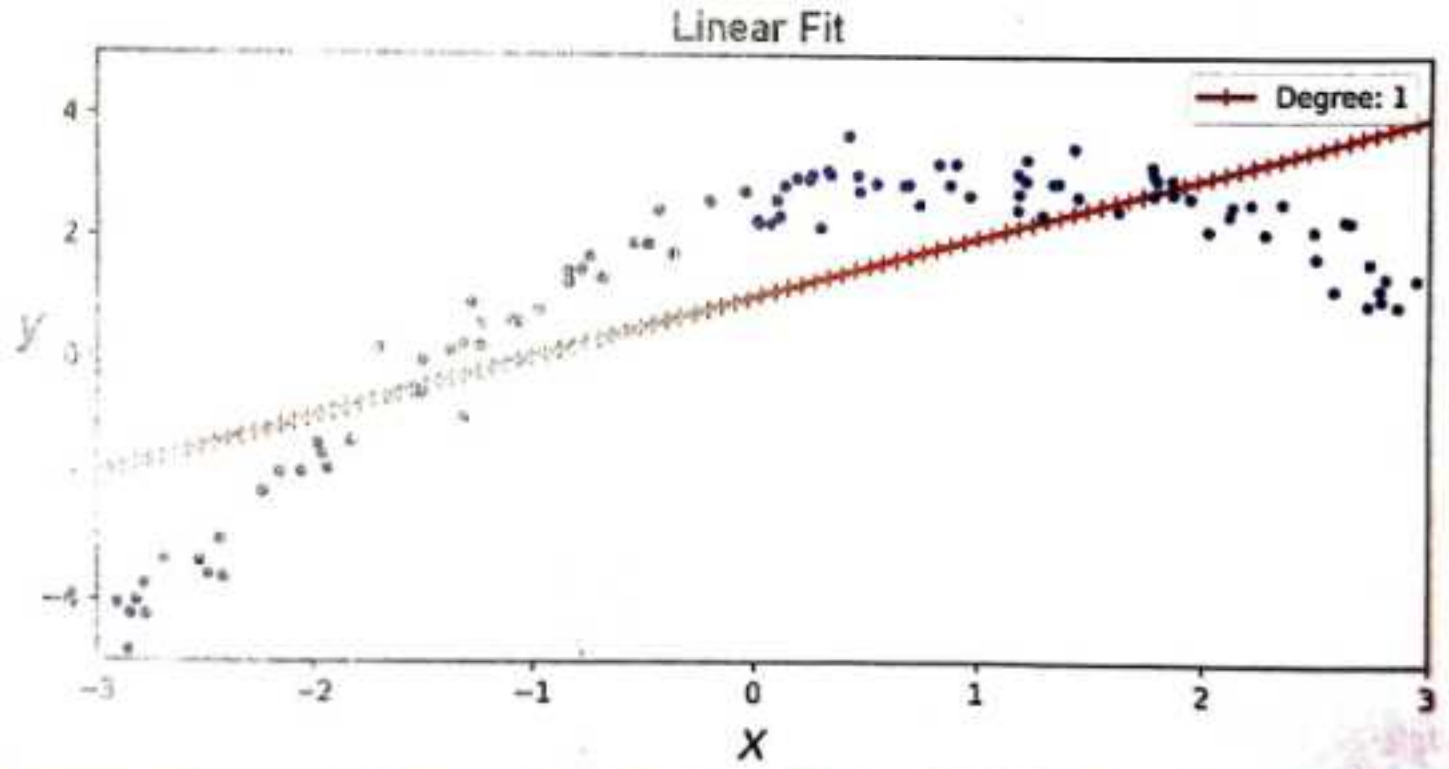


Figure 5.1.1 (b) A linear fit over the data from Equation 5.1.1

(b) and a polynomial of degree 100 in Figure 5.1.1 (c), *i.e.*,

$$y = \beta_0 + \beta_1 \times X + \beta_2 \times X^2 + \ldots + \beta_{100} \times X^{100} \qquad \textit{Equation } \mathbf{5.1.3}$$

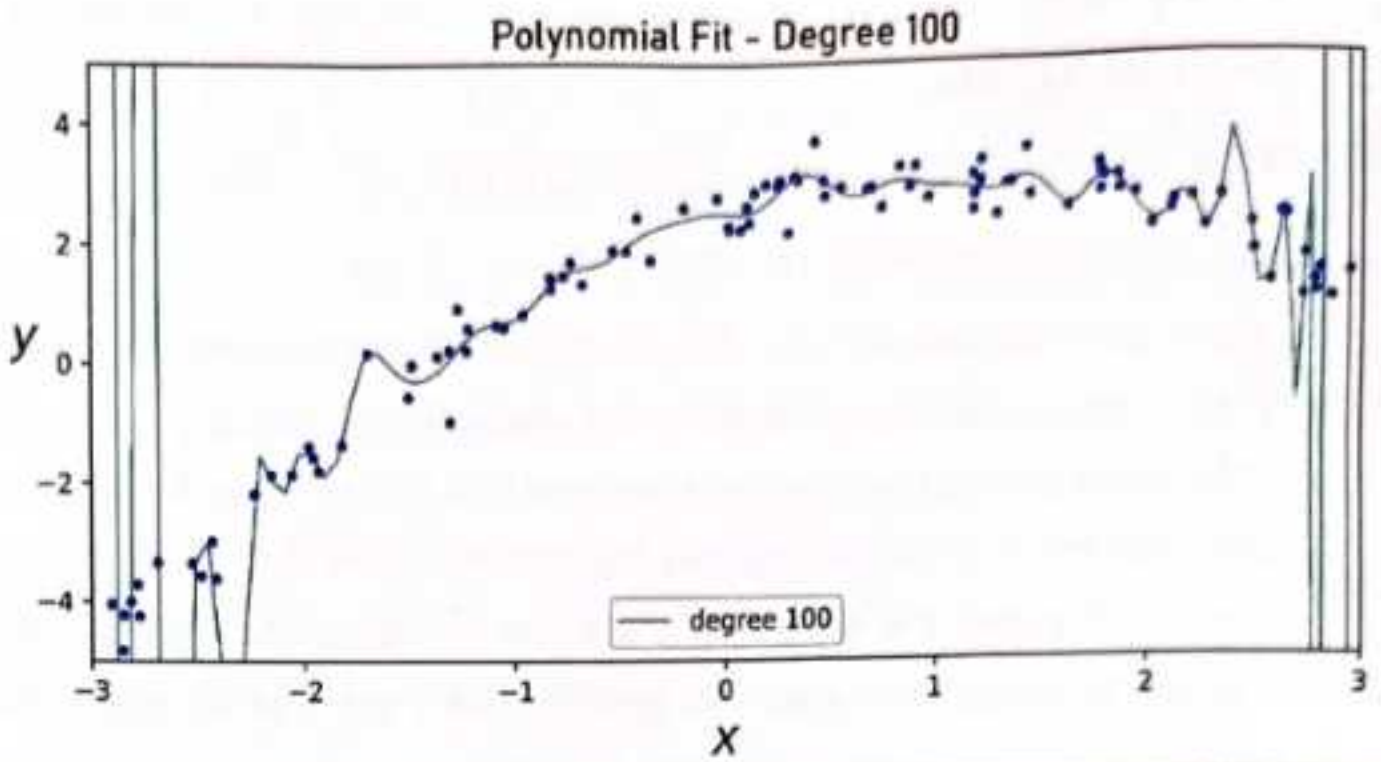


Figure 5.1.1 (c) A polynomial fit of degree 100, over the data from Equation 5.1.1

And

(c) a polynomial of degree 2, shown in Figure 5.1.1 (d), *i.e.*,

$$y = \beta_0 + \beta_1 \times X + \beta_2 \times X^2 \qquad \textit{Equation } \mathbf{5.1.4}$$

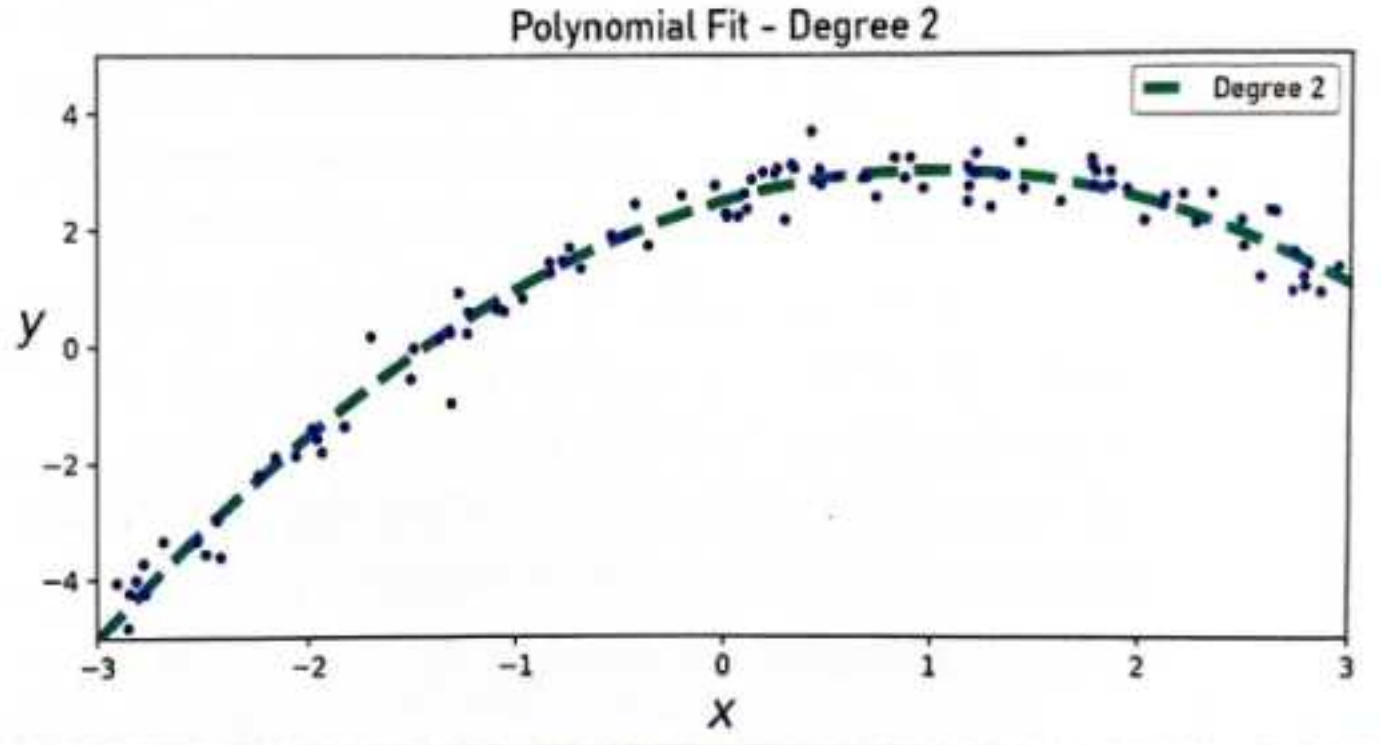


Figure 5.1.1 (d) A polynomial fit of degree 2, over the data from Equation 5.1.1

5.1.1 Evaluation Of The Model: Underfitting, Over-fitting, Or Fitting Right

When we closely observe the graph of the function given earlier with noise added, we can perceive whether our model is fitting right or not.

What we see is that in Figure 5.1.1 (b), the linear model is not fitting the curved nature of the plot. It should be obvious that the model error on the dataset will be high. If **unseen** data (*X* values not used in building the model) is given to the model, and the data follows the pattern of the data used for building the model, **we expect the errors on the unseen data to be similar to that on the training data, *i.e.*, we expect the errors to be high**. This is called **underfitting**.

If we look at Figure 5.1.1 (c), we are trying to use a polynomial of degree 100 to fit the data. We see that the model is trying hard to thread its way through every point of data, and in doing so, the model error will be very low on the **given** data. But consider the 100 degree polynomial where the value of *x* is between –3.0 and –2 and also between 2 and 3. The curve swings wildly between many of the succeeding data points that it threads its way through. If a new *X* falls between two of the existing *X*'s, the value of *y* returned will be wildly outside the range of *y* for the given data points. This will lead to huge errors for the new data points or *X*'s. So, we expect the errors of the **model on the given dataset to be very small, but errors on new data points to be huge.** This is called **overfitting.**

If we examine Figure 5.1.1 (d), we are using a polynomial of degree two to approximate the pattern in the data. Model error on the given data is seen to be low. If the new data follows the same pattern as the training data, we expect the model error to be low on the new data. So, **in this case model error is low both on the given data as well as on new data.** In this case, the model is fitting the data right and neither underfitting or overfitting it.

The model performing well on the training dataset is not important. We want our model to perform well on data it has not seen before. Hence, we want a model which will neither underfit, nor overfit. It is not possible to plot a function which has more than three dimensions–so we cannot always plot the model function to visualize whether it is overfitting or underfitting the data.

NOTE When a model is overfitting the data, it produces a small error on the dataset used for building the model and large errors when used on unseen data. Hence, we also say that in such cases, the model does not generalize well.

To evaluate whether our model is overfitting, underfitting, or fitting right, we need to evaluate it on a dataset that it has not seen before mathematically, without resorting to visual aids. How do we generate a dataset that the model has not seen before on which we can evaluate it? We will discuss this in the next chapter.

Remedies for overfitting

There are many ways of addressing overfitting. We briefly describe each of the documented methods below.

1. **Remove irrelevant features.** Training with irrelevant features may cause the model to fit the data in the irrelevant features leading to overfitting. Identifying irrelevant features may not be straightforward. Check if the feature is expected to influence the outcome. If not, then try removing it and rebuilding the model.
2. **Regularization.** Regularization is a technique used for reducing the flexibility of a model. Lasso is a regularization technique used to limit the size of the coefficients of a parametric regression model such as Linear Regression leading to a modified algorithm called Lasso Regression. In decision trees, regularization techniques such as pruning the size of the tree are used. Decision trees are discussed later in the book.
3. **Ensembling.** Ensemble means a collection. An ensemble of models is sometimes used to reduce overfitting. Ensembling is also discussed later in the book.
4. **More data.** At times, adding more data to the training data set can reduce overfitting by forcing the model to generalize on unseen data points.

CHAPTER 5 EXERCISES

Review Exercises

1. When do we say a model is overfitting the data? Give an illustrative example.
2. When do we say that a model is underfitting the data? Give an illustrative example.
3. Suppose, we are using a polynomial of degree 200 to fit a given dataset. Are we likely to overfit or underfit the data and why?
4. When can we say that our model is neither overfitting nor underfitting, *i.e.*, it is fitting right?

Investigative Exercises

1. What are ways of reducing overfitting of a model?
2. Concepts of model **Bias** and **Variance** are related to overfitting and underfitting of data. The interested reader is asked to consider alternate sources for an understanding of the bias-variance trade off in model selection.

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#bias-and-variance

6

Model Validation (Train-Test Split)

6.1 DERIVING DATA TO TRAIN THE MODEL

In the previous chapter, we have seen that a model can underfit or overfit the data from which it has been built. To evaluate whether the model is doing so or not, we need data that the model has not been built with and data which it has not seen before. It can also be shown (and you may be able to figure this out on your own) that it is possible to build a model which provides perfect predictions for the dataset that has been provided by

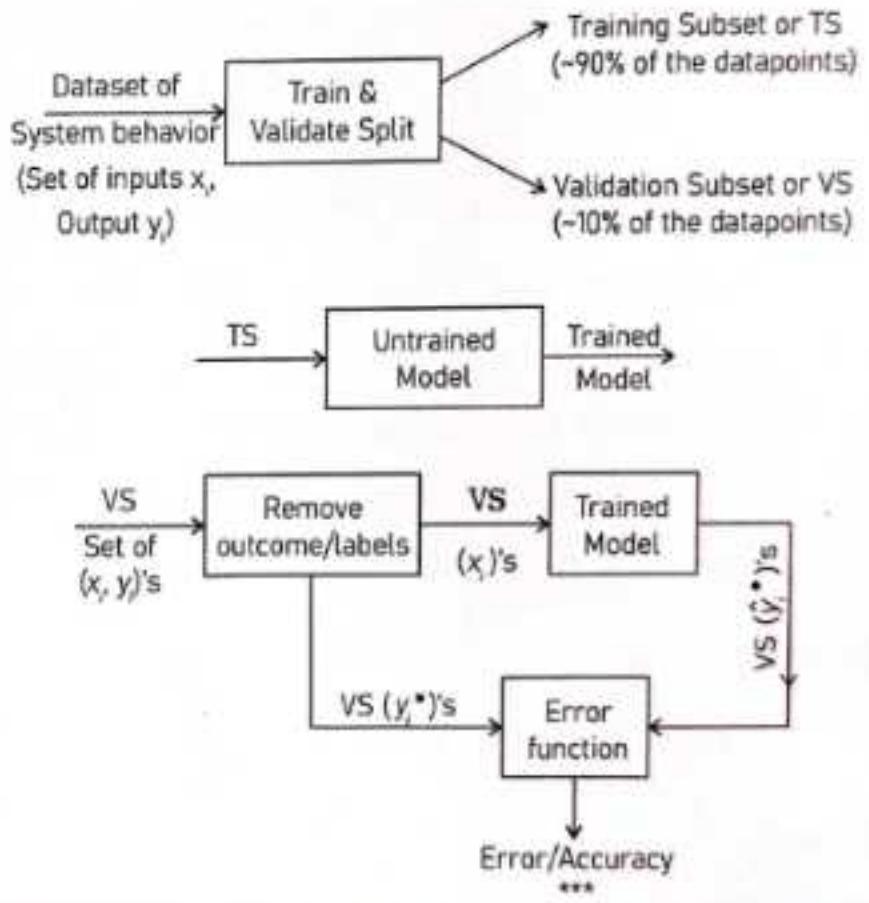


Figure 6.1.1 Illustration of splitting up a dataset into a training set, for training a model and a validation or test set for an unbiased evaluation of the accuracy of the trained model

memorizing the dataset. But this 'perfect' model may not do very well on a dataset it has not seen before. Hence, for an unbiased estimate of the accuracy of a model, it is important that it be tested to on a dataset which is different from the one it was built on. How do we get data that has not been used to train the model but can be used to evaluate the accuracy of the model predictions?

When we do not have two such separate sets of data, we have to create the two sets from the single dataset that we have for building the model by splitting it into two parts. One part is used to train the model, and the other part is used only for testing purposes. We use the terms 'test' and validate' interchangeably in literature, though these terms are used differently in ML. This step is called creating the train/test split. The test set is often called the validation set also. The process is illustrated in the coming section.

6.1.1 Creating The Train/Test Split

As shown in the Figure 6.1.1 earlier, the process of train/test split and then model building and accuracy checking proceeds as follows:

Step 1.

Split the data into a training set and a test set.

The percentage of the data that should be set aside for testing depends upon the characteristics of the dataset. The size of the test set can vary from 30% to even 1%. For the time being, an 80-20% split can be assumed. When splitting the data, typically, random sampling is used to create the two sets. The importance of random sampling is explained in the Appendix E Chapter on *Statistics* and *Probability*.

Step 2.

Train the model using the training dataset.

Step 3.

Perform predictions on the training dataset.

When performing predictions, the features are given only to the trained model. Calculate accuracy of predictions on the training dataset.

Step 4.

Perform predictions on the test dataset.

When performing predictions, the features are given only to the trained model. Calculate accuracy of predictions on the test dataset.

Step 5.

Determine if the model is underfitting, overfitting, or fitting right by comparing the accuracies obtained in steps 3 and 4.

We will illustrate this process on a synthetic example we discussed in Chapter 5 and then on the insurance cost example.

6.2 HANDS-ON EXERCISE: TRAIN/TEST SPLIT FOR EVALUATING MODEL FIT (OVERFITTING AND UNDERFITTING)

We are going to demonstrate overfitting and underfitting on the quadratic polynomial we had used earlier for the same purpose, *i.e.*,:

$$y = 2.5 + X + 0.5 \times X^2 + noise \qquad \textit{Equation } \mathbf{6.2.1}$$

Step 1.

Generate the data with the noise.

In [1]:

```
# Some useful Imports

import numpy as np
import numpy.random as rnd
import matplotlib.pyplot as plt
%matplotlib inline
np.random.seed(42)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

```
In [2]: ▸  1  # Generate the dataset on which we will test various models.
           2
           3  ndata = 100
           4  factor = 100
           5  X = 6 * np.random.rand(ndata, 1) - 3
           6  y = -0.5 * X**2 + X + 2.5 + np.random.randn(ndata, 1)*factor
```

Step 2.

Generate the Features for a Model of Degree 1.

The function PolynomialFeatures automatically generates all powers of the variable upto the given degree. Since degree is given as 1, *X*_poly will be same as *X*. So, we will essentially be using a polynomial of degree 1 to fit the data, *i.e.*, a model given by

$$y = \beta_0 + \beta_1 * X \qquad \textit{Equation } \mathbf{6.2.2}$$

```
In [3]: ▸  # generate the polynomial features. In this case, since degree is 1, X_poly will be same as X.

           degree = 1
           poly_features = PolynomialFeatures(degree=degree, include_bias=True)
           X_poly = poly_features.fit_transform(X)
```

Step 3.

Do the train/test split.

The Python library Sklearn has a function for doing the split by means of random sampling. The percentage to be used for the test set can be specified as shown in the code below.

```
In [4]: ▸  # Do the train/test split.

           from sklearn.model_selection import train_test_split
           X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20, random_
           state=17)
```

Step 4.

Build the model using the training set only.

Note that the target variable (y_train) needs to be provided in addition to the X_train (containing features) for building the model.

```
In [5]: ▸ # Train the model with the training set.

          lin_reg = LinearRegression()
          lin_reg.fit(X_train, y_train)

Out [5]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                          normalize=False)
```

Step 5.

Find the error metrics for the training set.

Note that for the model to predict, the outcome variable should not and need not be specified.

```
In [6]: ▸ 1 # Compute the error metrics on the training set.
        2 y_train_hat = lin_reg.predict(X_train)
        3
        4 # Regression Evaluation Metrics
        5 from sklearn import metrics
        6
        7 print('MAE:', metrics.mean_absolute_error(y_train, y_train_hat))
        8 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_hat)))
        9 print('R_squared:', metrics.r2_score(y_train, y_train_hat))

          MAE: 1.1963278531204378
          RMSE: 1.4777215148981244
          R-squared: 0.49313576737813525
```

Step 6.

Find the error metrics for the test set.

```
In [7]: ▸ 1 # Compute error metrics on the test set.
        2 y_test_hat = lin_reg.predict(X_test)
        3
```

```
4  print('MAE:', metrics.mean_absolute_error(y_test, y_test_hat))
5  print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_hat)))
6  print('R-squared:', metrics.r2_score(y_test, y_test_hat))
```

```
MAE: 1.231480417558453
RMSE: 1.658394138192764
R-squared: 0.1118707819537007
```

Remedy for underfitting

We see in the previous section that the error metrics are mediocre, both with the training and the test data. We are definitely **underfitting**.

Repeat for model with polynomial degree of upto 20, *i.e.*, a model given by:

$$y = \beta_0 + \beta_1 \times X + \beta_2 \times X^2 + \ldots + \beta_{20} \times X^{20} \qquad \textit{Equation } \mathbf{6.2.3}$$

Step 1.

Generate the features, do the train/test split, and train the model with the training set.

In [8]:
```
degree = 20
poly_features = PolynomialFeatures(degree=degree, include_bias=True)
X_poly = poly_features.fit_transform(X)
```

In [9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20, random_
state=42)
```

In [10]:
```
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
```

Out [10]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

Step 2.

Find the error metrics for the training set.

In [11]:
```
1  y_train_hat = lin_reg.predict(X_train)
2
3  print('MAE:', metrics.mean_absolute_error(y_train, y_train_hat))
```

```
4  print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_hat))
5  print('R_squared:', metrics.r2_score(y_train, y_train_hat))
```

```
MAE: 0.22081907112598867
RMSE: 0.42941188651773804
R-squared: 0.9571989348726645
```

Step 3.

Find the error metrics for the test set.

In [12]:
```
1  y_test_hat = lin_reg.predict(X_test)
2
3  print('MAE:', metrics.mean_absolute_error(y_test, y_test_hat))
4  print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_hat)))
5  print('R_squared:', metrics.r2_score(y_test, y_test_hat))
```

```
MAE: 10.898178460413243
RMSE: 16.502729543118O2
R-squared: -86.94521487489763
```

The R^2 metric is close to 0.96 for the training data set, and the MAE (Mean Absolute Error) and RMSE are very low, indicating a very good fit. But we see for the test set that the R^2 has an abnormal value and the other metrics are incredibly large. This is clearly an example of **overfitting**. Our model has exceptionally high RMSE on the test dataset and a ridiculously large (negative) value of the R^2 metric.

Remedy for overfitting

Let's repeat with a model which is a polynomial of degree 2. This should allow an almost perfect fit, barring the noise which is there in the dataset. The polynomial will be of the form below:

$$y = \beta_0 + \beta_1 \times X + \beta_2 \times X^2 \qquad \textit{Equation } \mathbf{6.2.4}$$

Step 1:

Generate the features, do the train/test split and train the model with the training set.

In [13]:
```
1  degree = 2
2  poly_features = PolynomialFeatures(degree=degree, include_bias=True)
3  X_poly = poly_features.fit_transform(X)
```

```
In [14]: 1 from sklearn.model_selection import train_test_split
         2 X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20,
         3             random_state=42)
```

```
In [15]: 1 from sklearn.model_selection import train_test_split
         2 X_train, X_test, y_train, y_test = train_test_split(X_poly, y, test_size=0.20,
         3             random_state=42)

Out [15]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
                  normalize=False)
```

Step 2.

Find the error metrics for the training set.

```
In [16]: 1 y_train_hat = lin_reg.predict(X_train)
         2
         3 print('MAE:', metrics.mean_absolute_error(y_train, y_train_hat))
         4 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_hat)))
         5 print('R-squared:', metrics.r2_score(y_train, y_train_hat))
```

```
MAE: 0.6305346130956531
RMSE: 0.7766921439783101
R-squared: 0.859975420305426
```

Step 3.

Find the error metrics for the test set.

```
In [17]: 1 y_test_hat = lin_reg.predict(X_test)
         2
         3 print('MAE:', metrics.mean_absolute_error(y_test, y_test_hat))
         4 print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_hat))
         5 print('R-squared:', metrics.r2_score(y_test, y_test_hat))
```

```
MAE: 0.7686317266892623
RMSE: 0.9731373041500693
R-squared: 0.6941919651035398
```

The figure given here shows the fitted curve to the data.

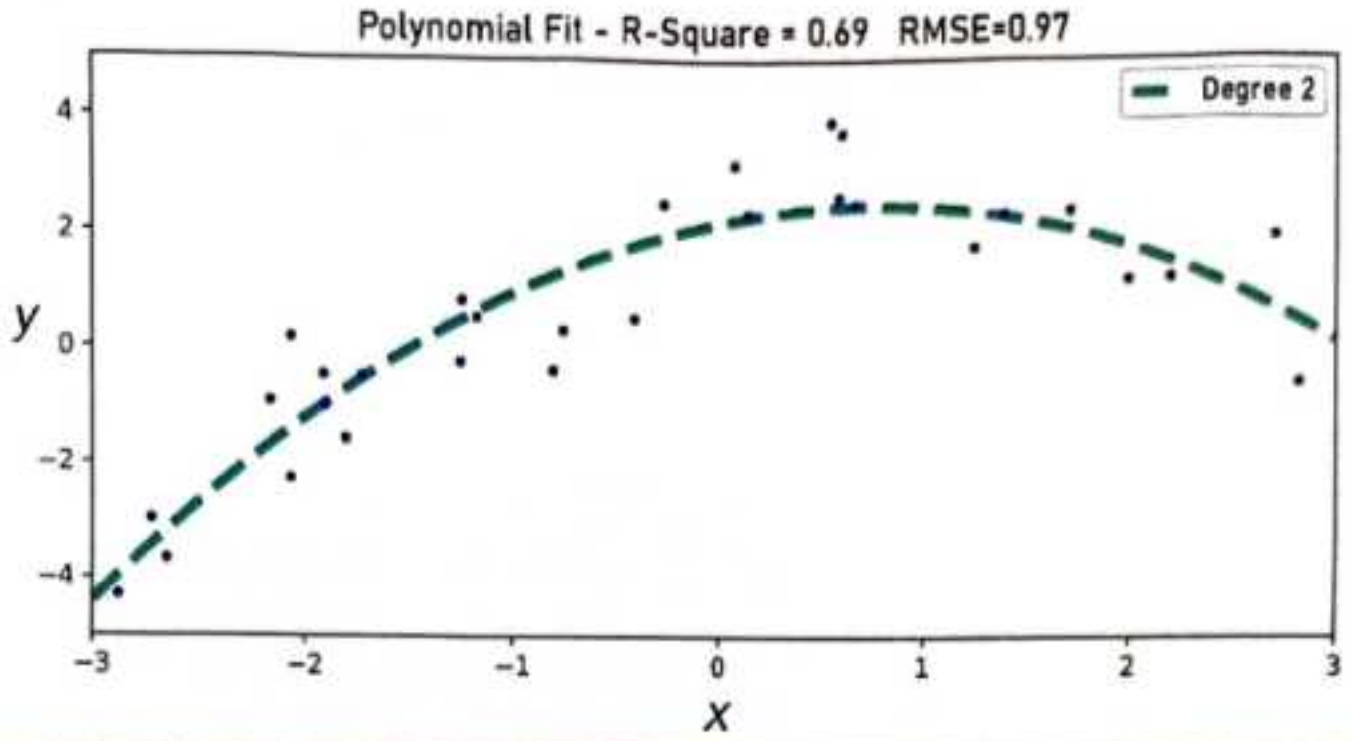


Figure 6.2.1 Polynomial fit (Degree 2) to the dataset

For this model, we see that the metrics are fairly good on the training set and we are getting comparable numbers on the test set, indicating that the model is generalizing well and neither underfitting nor overfitting.

Ensuring similarity of the dataset for training and testing

Random Sampling

We had mentioned earlier that when the training and test split is done, the examples in each split should be randomly sampled from the original data set.

To explain this, let us turn our attention to the insurance dataset once again. We noticed that whether a person is a smoker and/or is obese has a highly significant effect on the healthcare cost. We also noticed that people who do not smoke and are not obese have low healthcare costs. Those who smoke or are obese have moderately high costs. Those who both smoke and are obese have very high healthcare costs.

Let us say that in our dataset, customers who neither smoke nor are obese are listed first (*i.e.*, these customer profiles were the first few rows of the dataset). Those who smoke are listed next, next those who are obese, and next those who both smoke and are obese are listed. If we took the first 80% of the rows or examples from the dataset as our training set, we would probably include all

customers except those who were both smokers and were obese. The test dataset would likewise include mostly customers who are smokers as well as obese.

Thus, the model would be trained without the fourth set of customers who have very different characteristics than those in the first three sets, while it would be tested almost exclusively on the fourth set. The model in such case would not perform well on the test set since the corresponding patterns were not used for training it.

The basic idea, thus behind a train/test split is that both should include examples with similar patterns. A way of achieving this objective is by random sampling of the dataset. The assumption is that if the sampling is indeed random, it will pick representative profiles for both the training and the test set. Hence, the dataset used for training the model would be similar to the dataset used for testing it.

6.3 HANDS-ON EXERCISE: TRAIN/TEST SPLIT ON THE INSURANCE PROBLEM

In the first attempt for building a model for the insurance problem, we had built the model on the whole dataset and then evaluated it on the whole dataset, without doing a train/test split. We are going to re-evaluate the insurance cost prediction problem by doing a train/test split to see if our model is overfitting, underfitting, or adequately fitting our data. **All models should be evaluated on the training and test dataset as shown below for completeness.**

Step 1.

Load the data, create the **additional features, and create the features** DataFrame and target.

```
In [1]: # Load the dataset into pandas dataframe and print the sample of first 5 rows

        import pandas as pd
        import numpy as np

        df = pd.read_csv('../Data/insurance_data.csv')

        # We create a new feature which is the product of the smoker and obesity features.

        df['smokOb'] = df['smoker'] * df['obese']
```

```
# Create the pandas dataframe with the features and the pandas series with the target.

x = df[['age', 'bmi', 'children', 'smoker', 'obese', 'sex', 'smokOb']]
y = df['charges']
```

Step 2.

Do the train/test split.

```
In [2]: # Do the train/test split.

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
        random_state=42)
```

Step 3.

Train the model with the training set.

```
In [3]: from sklearn import linear_model

# Let's create an instance for the LinerRegression model.
lr = linear_model.LinearRegression()

# Train the model on our train dataset
lr.fit(X_train, y_train)
```

```
Out [3]: LinearRegression(copy_X=True, fit_intercept=True, n_job=None,
         normalize=False)
```

Step 4.

Find the error metrics for the training set.

```
In [4]: # Compute the error metrics on the training set.
y_train_hat = lr.predict(x_train)

# Regression Evaluation Metrics
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_train, y_train_hat))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_hat)))
print('R_squared:', metrics.r2_score(y_train, y_train_hat))
```

```
# Create the pandas dataframe with the features and the pandas series with the target.

x = df[['age', 'bmi', 'children', 'smoker', 'obese', 'sex', 'smokOb']]
y = df['charges']
```

Step 2.

Do the train/test split.

```
In [2]: # Do the train/test split.

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
        random_state=42)
```

Step 3.

Train the model with the training set.

```
In [3]: from sklearn import linear_model

# Let's create an instance for the LinerRegression model.
lr = linear_model.LinearRegression()

# Train the model on our train dataset
lr.fit(X_train, y_train)

Out [3]: LinearRegression(copy_X=True, fit_intercept=True, n_job=None,
         normalize=False)
```

Step 4.

Find the error metrics for the training set.

```
In [4]: # Compute the error metrics on the training set.
y_train_hat = lr.predict(x_train)

# Regression Evaluation Metrics
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_train, y_train_hat))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_train, y_train_hat)))
print('R_squared:', metrics.r2_score(y_train, y_train_hat))
```

```
MAE: 2514.1908886233955
RMSE: 4504.98801708294
R-squared: 0.8593888467410522
```

Step 5.

Find the error metrics for the test set.

```
In [5]: # Compute error metrics on the test set.
        y_test_hat = lr.predict(X_test)

        print('MAE:', metrics.mean_absolute_error(y_test, y_test_hat))
        print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_test_hat)))
        print('R-squared:', metrics.r2_score(y_test, y_test_hat))

        MAE: 2377.7866280565795
        RMSE: 4284.029807145157
        R-squared: 0.8217838238537834
```

The R^2 on both the training and test dataset are similar and are fairly high. From this, we conclude that the model is reasonably fitting the data and neither underfitting nor overfitting it.

Fine-tuning the train/test split

Cross Validation

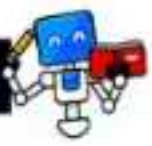
Cross validation is a more robust way of validating a model by taking the train/test split approach a step further. We explain this through an example.

In cross validation, the dataset is split into N subsets where N is usually a small number, such as 5 or 10. Let us assume N is 5. Subsequently, the model is built with all the data except the 1st set and evaluated with the 1st set. Next, it is built using all the data except the 2nd set, and evaluated using the 2nd set. This is repeated 5 times, each time using one of the 5 sets for evaluation and the rest for building the model. The average of the model accuracy over the 5 experiments is used as the overall model accuracy. Cross validation is considered to be a more robust measure of validation than the simpler single train/test split approach.

Recap of Steps in Machine Learning

We had presented at the end of Section 2.4, a summary of the high-level steps in a rudimentary machine learning pipeline. Given the additional concepts we have learnt since then, we present the high-level steps in a more sophisticated ML pipeline.

1. We load the dataset of interest.
2. We analyze the data – which includes correlation analysis of the features against the target.
3. We derive potential new features from our dataset.
4. We perform a train/test split.
5. We choose a general predictive model to represent the relationship between the independent and dependent variables and train the model with the training dataset.
6. We determine the model accuracy on the training and test dataset.
7. In case the model underfits or overfits the data, we go back and explore alternate models and/or the data.

CHAPTER 6  EXERCISES

Review Exercises

1. Why do we need two separate sets of data to train a model and then evaluate it?
2. How can we create the two sets of data for training and evaluating the model from a single dataset?

Investigative Exercises

1. Do you expect the model error to be higher on the training set or the test set? Why?

2. Suppose we have hourly temperatures of the city you live in. We have gathered this data over several years. Would random sampling be a good strategy to use if we want to work with a sample of the dataset? If yes, why? If not, why not?
3. We have introduced the concept of train/test split where the training subset is used to build a model and the test subset is used to evaluate it. The intuition provided is that we want to evaluate the performance of the model on data it has not seen before.

   Let us say we build a model from the training set and then evaluate it with the test set. We find the accuracy figures not high enough. We analyze why the model is not working well on the test set, tune the model, and re-evaluate it with the test set. In essence, our new model has now become dependent on the test set also, since we analyzed the shortcomings of the first model on the test set to build the second model. So, it is not completely accurate to say that our model has not been trained on the test set.

   In such a scenario, what would be a more robust approach of evaluating our model?

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#model-validation

7

Classification Problems

7.1 INTRODUCTION TO CLASSIFICATION

Just as Regression represents a class of problems where the target variable can take continuous values, Classification is another important class of problems where the target variable takes discrete values only. As we will see, this requires us to use learning algorithms which work differently, as well as measure the model accuracy differently compared to regression model error measurements.

radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	diagnosis
16.13	20.68	108.10	798.8	0.11700	M
14.92	14.93	96.45	686.9	0.08098	B
13.75	23.77	88.54	590.0	0.08043	B
13.40	20.52	88.64	556.7	0.11060	M
20.29	14.34	135.10	1297.0	0.10030	M

Figure 7.1.1 A sample of the dataset with characteristics of benign and malignant tumors

We provide a sample dataset for a classification problem below. The problem is to predict whether a patient's tumor is benign (target variable is given value 'B') or malignant (target variable is given value 'M'). Various characteristics of the tumor such as its mean radius, texture, perimeter, etc. are given as features. So, the dataset is very similar to a regression dataset. The only significant difference is that the target variable has discrete values and not continuous values.

Discrete, Continuous, and Categorical variables are explained in the Appendix E Chapter on Probability and Statistics. A variable is called **categorical** if it can take any value from a fixed set of values. The values of a categorical variable are said to be discrete.

NOTE

Machine learning algorithms typically work with numerical data. Hence, all categorical variables are converted into numbers before they are passed on to any ML algorithm. We have already seen a categorical variable called 'smoker' in the insurance cost problem where the condition was coded as 0 and 1. In case of the 'diagnosis' variable discussed above, we need to map each of the possible values into a different number. One possible mapping is B → 0, M → 1

NOTE

One should be cautious of the mapping of categorical variables to numbers. We could have mapped the values of the 'smoker' variable to numbers 2 and 3 also. The mapping used matters and different methods are used to generate the mapping. Why the mapping matters is beyond the scope of this book and we will use the simple mapping of n values to the numbers {0, 1, ..., (n – 1)}.

We would like to point out one important consequence of such mappings though. If we are given two numbers, '0' and '1', we quickly assume '1' is greater than '0'. But when we are talking about categorical values, there is usually no ordering amongst them. So, refrain from thinking of ordering when examining the mapped values of a categorical variable.

Each value of the outcome variable is called a class. For example, the tumor prediction problem has two classes, 'Benign' and 'Malignant'.

To reiterate, a **classification** problem is a prediction problem where the **target** variable is **categorical or discrete**. Note that the features may be continuous and need not be discrete. As mentioned at the beginning, classification problems require a different algorithm as compared to the regression problems and we would like to discuss the intuition behind this next, before going on to discuss some classification algorithms.

7.2 APPROACH FOLLOWED BY CLASSIFICATION ALGORITHMS

Classification algorithms usually work by predicting the probability of the example belonging to one of the possible classes that the dataset has. Class probabilities are subsequently converted into class labels.

Class probability is the probability that a given example belongs to one of the several classes of the target variable. If there are n classes, then an example i will have a class probability for every class, *i.e.*, $(p'_0, p'_1, ..., p'_{n-1})$ and the probabilities will add up to 1, *i.e.*, $p'_0 + p'_1 + ... + p'_{n-1} = 1.0$.

As an example, suppose our target variable has values ['cat', 'dog', 'cow']. The class probability for an example can be [0.3, 0.25, 0.45].

To convert class probabilities to class labels, usually, the class corresponding to the largest probability is used.

For the animal classification problem, given the class probabilities of an example as above, the predicted class will be 'cow' since it has the largest probability value.

In case of two-class classification problems, note that as per our discussion, the algorithm is expected to generate two probabilities — (p_0, p_1). Since $p_0 + p_1 = 1$, $p_0 = 1 - p_1$. Usually, the algorithm will generate p_1. p_0 can then be calculated by subtracting p_1 from 1. Also, since the two probabilities add up to 1.0, for one of them to be larger than the other, its value has to be more than ½ = 0.5. So if $p_1 > 0.5$, then the class is 1, else the class is 0.

NOTE From here onwards, we are going to consider mainly two-class classification problems, where we will use the labels '0' and '1' for the two classes. The discussions can be extended to classification problems containing more than two classes

7.2.1 Logistic Regression

In this section we discuss Logistic Regression, which is a model used for classification problems and not for regression problems, even though it contains the word regression in its name. To understand logistic regression, we need to understand the sigmoid function which will be discussed next followed by a discussion of the logistic regression algorithm.

The sigmoid function

The sigmoid function is given by the equation below:

$$y = \frac{e^z}{1 + e^z} \qquad \textit{Equation } \mathbf{7.2.1}$$

If we divide the numerator and denominator both by e^z, then we get the alternate form below:

$$y = \frac{1}{1 + e^{-z}} \qquad \textit{Equation } \mathbf{7.2.2}$$

As z grows large, *i.e.*, $z \to \infty$, $e^z \to \infty$. Hence, the numerator and denominator of y both tends to infinity and y tends to 1, *i.e.*,

$$\lim_{(z \to \infty)} y = \frac{e^z}{1+e^z} \to 1. \qquad \textit{Equation } \mathbf{7.2.3}$$

On the other hand, as z grows very small, *i.e.*, $z = -\infty$, $e^{-z} \to \infty$ and $e^z \to 0$. Hence, the numerator of y tends to 0 while the denominator tends to (1 + 0) or 1. Hence, y tends to 0, *i.e.*,

$$\lim_{(z \to -\infty)} y = \frac{e^z}{1+e^z} \to 0. \qquad \textit{Equation } \mathbf{7.2.4}$$

The graph of the function is given here.

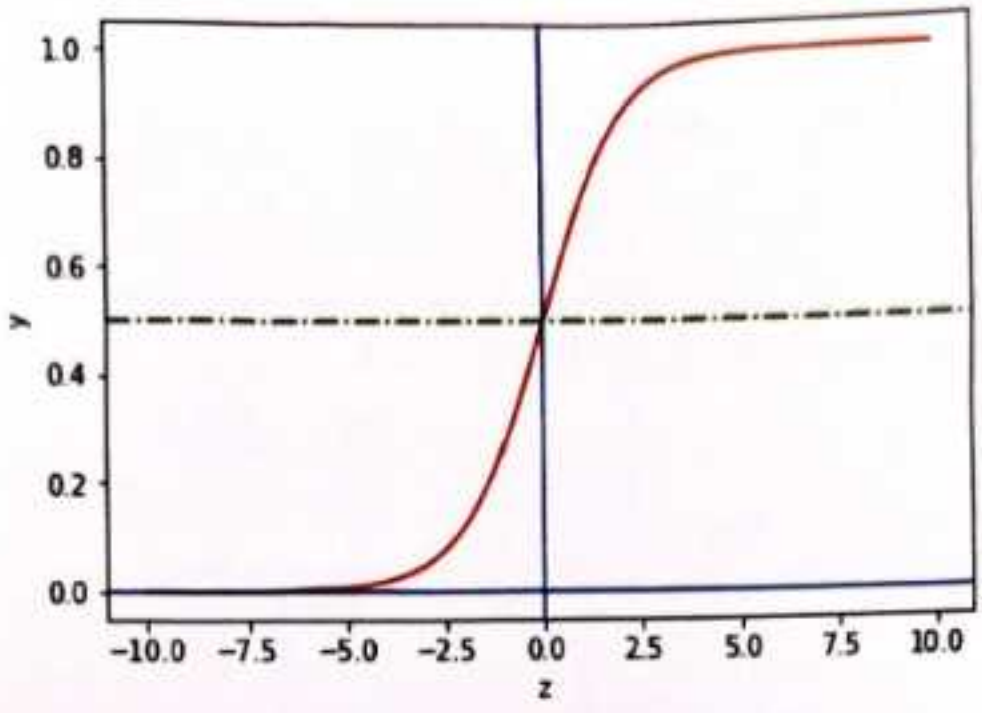


Figure 7.2.1 A plot of the Sigmoid function of Equation 7.2.1.

The characteristics of the sigmoid function can be summarized as follows:

- As the independent variable z becomes large, the function tends to the value 1.0.
- As the independent variable z becomes small, the function tends to the value 0.0.
- The function value is 0.5 when $z = 0.0$.
- The function is restricted in the range [0.0, 1.0].
- Also, notably, the function is somewhat linear around $z = 0.0$.
- Towards both extremes, the function becomes flat or almost parallel with respect to the z-axis.

Let us now consider a variant of the function where we replace the independent variable z with an independent variable x and a bias as below:

$$z = \beta_0 + \beta_1 \times x \qquad \textit{Equation } \mathbf{7.2.5}$$

Our sigmoid becomes:

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \times x)}} \qquad \textit{Equation } \mathbf{7.2.6}$$

Let us assign the following values $\beta_0 = -2$ and $\beta_1 = 1$, *i.e.*,

$$z = -2 + x \qquad \textit{Equation } \mathbf{7.2.7}$$

then the sigmoid becomes,

$$y = \frac{1}{1 + e^{-2+x}} \qquad \textit{Equation } \mathbf{7.2.8}$$

A plot of y as a function of x appears next.

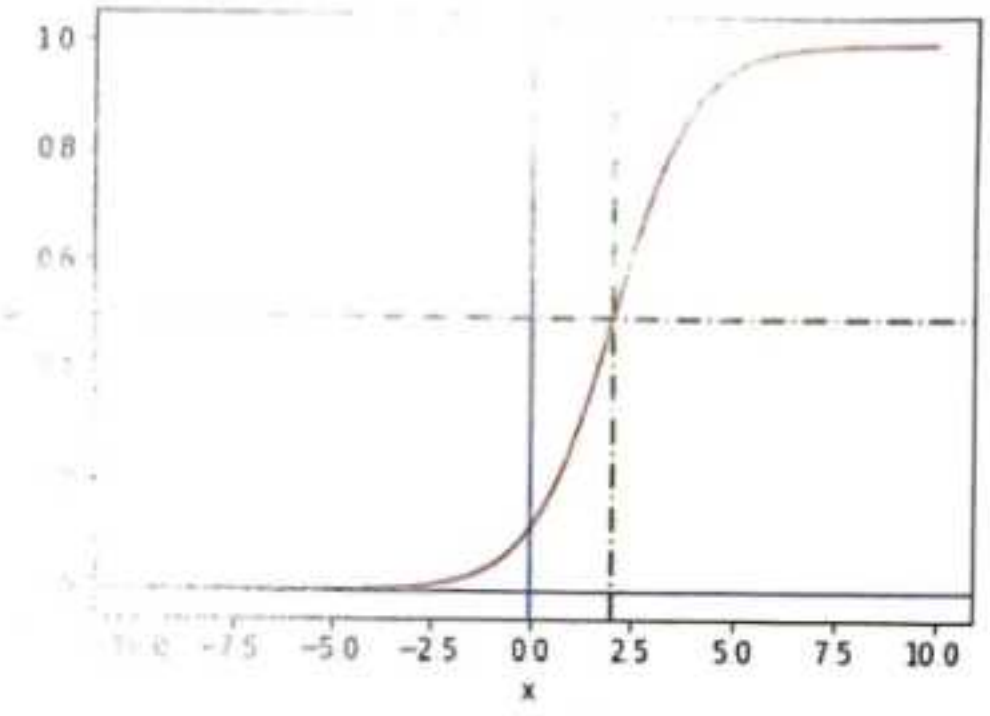


Figure 7.2.2 A sigmoid with a shift corresponding to Eq 7.2.8

Note that the graph in 7.2.2 looks similar to the one in 7.2.1, except it has shifted to the right by two units, *i.e.*, y is 0.5 when $x = 2$. This is because the power of e in Equation 7.2.8 is now $(-2 + x)$ and it takes on the value 0 when $x = +2$.

The parameters β_0 and β_1 not only control the value of x for which the function takes on the value 0.5, β_1 controls the slope of the function around the middle region. The sigmoid for $\beta_0 = -8$ and $\beta_1 = 4$, *i.e.*,

$$y = \frac{1}{1 + e^{-8+4x}} \qquad \textit{Equation } \mathbf{7.2.9}$$

is shown here:

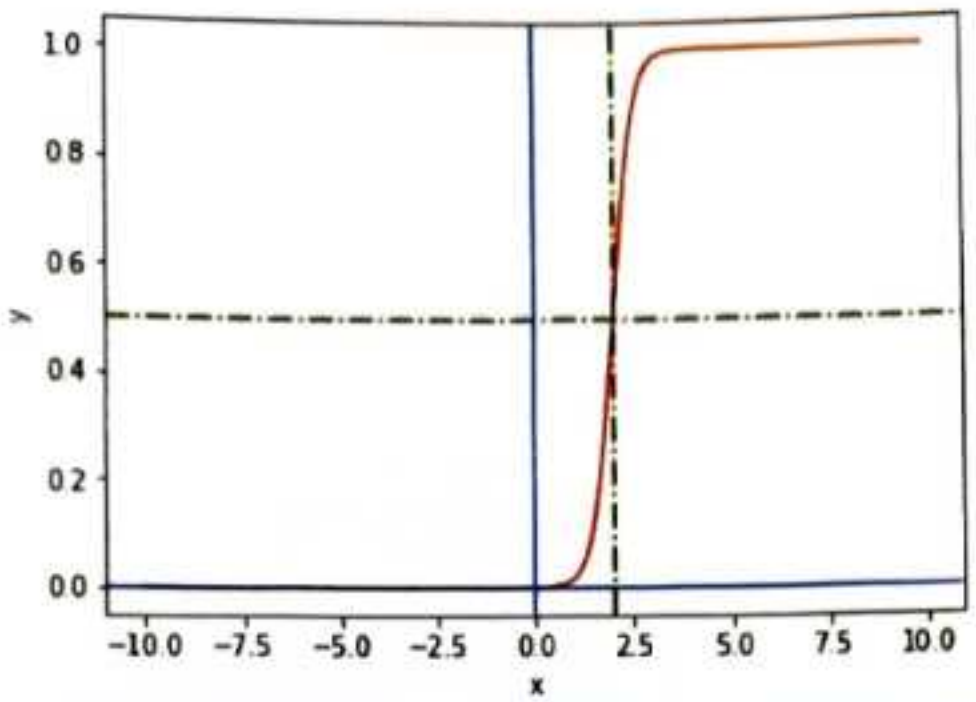


Figure 7.2.3 A plot of the sigmoid of Eq 7.2.9.

The function still takes on the value 0.5 when $x = 2.0$, however the transition from 0.0 to 1.0 is much sharper than the function in Figure 7.2.2.

We can extend the sigmoid function easily to work with multiple x_i's as below:

$$z = \beta_0 + \beta_1 \times x_1 + \ldots + \beta_n \times x_n \qquad \textit{Equation } \mathbf{7.2.10}$$

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 \times x_1 + \ldots + \beta_n \times x_n)}} \qquad \textit{Equation } \mathbf{7.2.11}$$

Even though there are multiple x_i's, it should be clear that the range of y remains unchanged at [0.0, 1.0].

We will use this sigmoid function to build our first classifier in the next section.

The logistic regression classifier

We will continue our discussion assuming we have a two-class classification problem. We have mentioned that classifiers generate class probabilities per example. For a two-class classification problem, the classifier needs to generate a probability p_1 only for the class 1, the probability for class 0 is simply $p_0 = 1 - p_1$. We have also seen that the sigmoid function value ranges between [0.0, 1.0] and hence can be used as a probability.

The logistic regression classifier simply uses the sigmoid function to generate the probability for the class 1. More specifically, assuming we have n features ($x_1, \ldots, x_n$), the logistic regression classifier can be written as below:

$$\text{Probability (class = 1)} = p_1 = \frac{1}{1 + e^{[-(\beta_0 + \beta_1 * x_1 + \ldots + \beta_n * x_n)]}} \qquad \textit{Equation } \mathbf{7.2.12}$$

We know from the property of the sigmoid function that the value of the above expression will be between (0.0, 1.0), *i.e.*, it can be used as a probability. We need to determine the beta values such that when the features are of an example belonging to class 1, p_1 value should be greater than 0.5. Since when the value of p_1 is more than 0.5, as per the classification approach discussed, we predict that the example belongs to class 1.

For the regression problem, we defined the RSS metric and used it to evaluate model error over all the samples in the training dataset. Then we went further to say that a parameter optimization algorithm would efficiently tweak the parameters so as to minimize the model error. Finally, the parameter values which minimized the RSS would define the model for the given dataset.

The problem of machine learning in the case of logistic regression is very similar to the linear regression problem but with a twist. We still need to determine the model parameters, *i.e.*, the β's so that the model generates probabilities which accurately predict the class or label for each example. However, the question is whether RSS is an adequate metric to measure a classification model's errors.

7.3 A VISUAL REPRESENTATION OF LOGISTIC REGRESSION

We will provide a visual of how logistic regression works by using a synthetic two-class classification problem with a single feature. Let's consider a simplified version of the tumor prediction problem. $y = 0$ indicates a benign tumor and $y = 1$ indicates malignancy. x is a feature and is the diameter of the tumor. Values of y versus x are given in the table in Figure 7.3.1.

x	y	x	y	x	y
1	0	9	1	15	1
2	0	10	0	16	1
3	0	11	0	17	1
4	0	12	1	18	1
5	0	13	1	19	1
6	0	14	0	20	1
7	0				
8	0				

Figure 7.3.1 Dataset used to illustrate logistic regression with two classes (y can be 0 and 1)

The (x, y) values are plotted in the figure in Figure 7.3.2. The points have been colored with two different colors depending upon their class, 0 or 1. The ones colored blue correspond to benign cases, red corresponds to malignant cases.

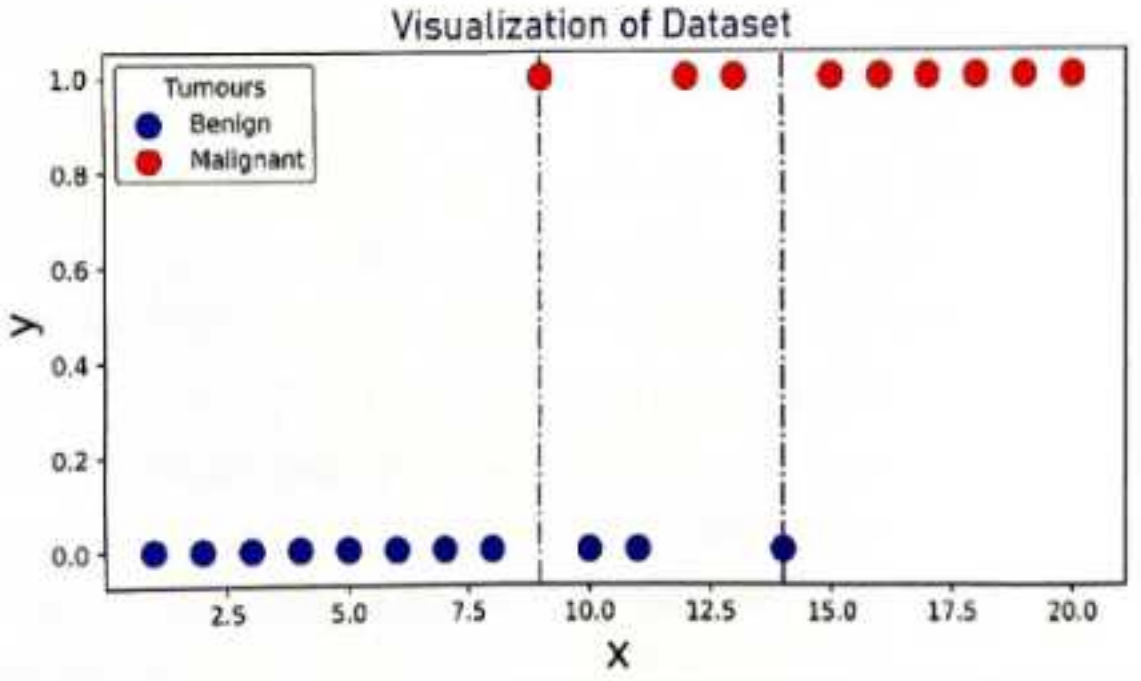


Figure 7.3.2 A graphical illustration of the dataset of Figure 7.3.1

We see that when x (or the tumor radius) is less than or equal to 8, then it is always benign. When x is more than 15, it is always malignant. The region of uncertainty is between [9, 15]. Of course, in real life, the pattern may be more random. Nonetheless, consider the sigmoid curve in Figure 7.3.3 below which is an attempt to fit the data in Figure 7.3.2.

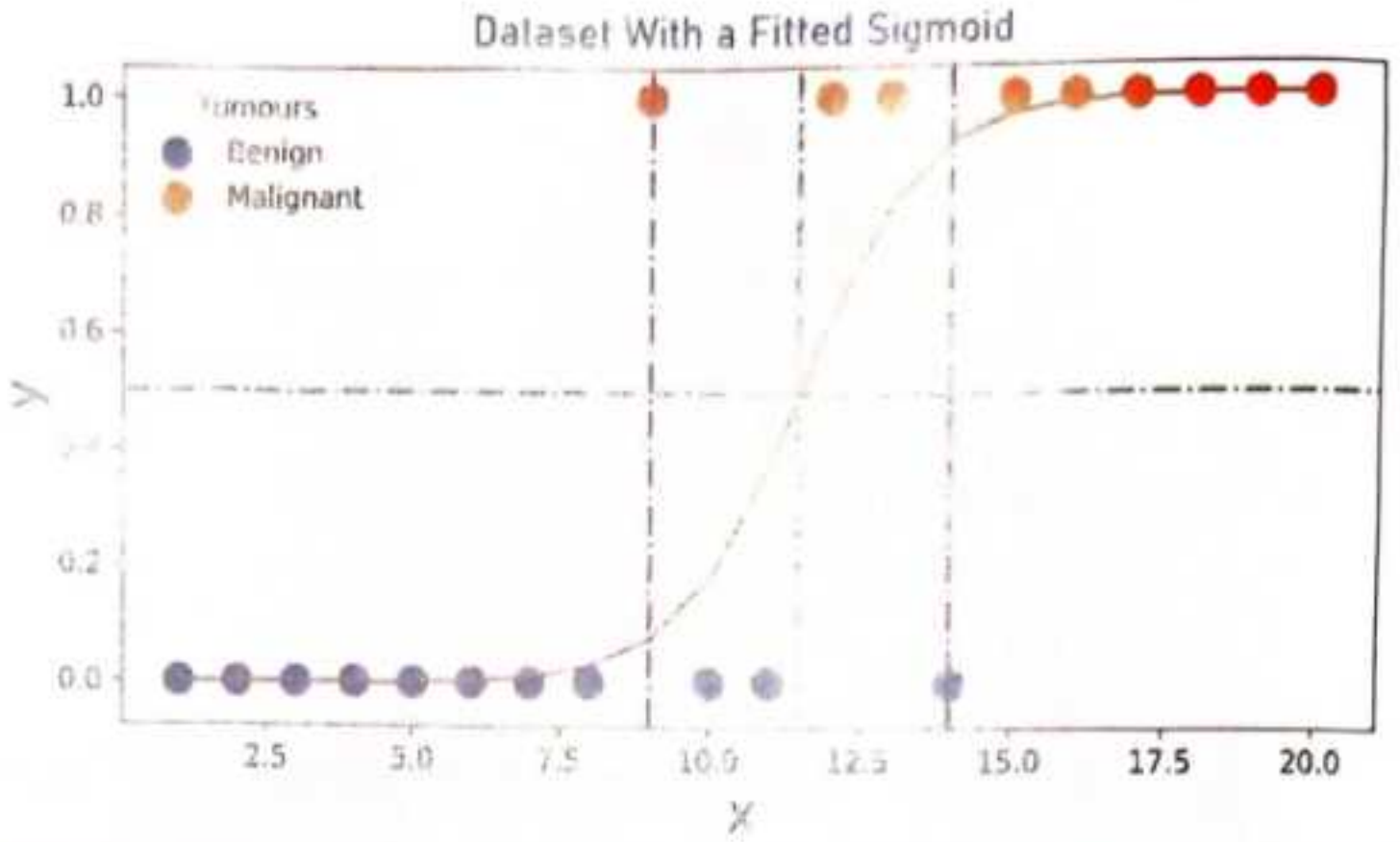


Figure 7.3.3 A sigmoid function to approximate the dataset of Figure 7.3.1

The center of the transition region of the sigmoid has been aligned with the center of region of uncertainty of the data ($x = 11.5$) and the width of the transition region of the sigmoid has been made to fit the region of uncertainty of the data which extends roughly 9 to 14. This sigmoid will generate very low values of probability when $x \leq 9$ and very high values when $x \geq 14$. These will lead to correct predictions of the class the example belongs to. In the transition region, the probabilities will be intermediate in value and will lead to some correct and some incorrect predictions.

This is how a sigmoid is used by a Logistic Regression classifier. The work involved in fitting a sigmoid to the data includes determining the parameters of the sigmoid such that the wrong predictions are minimized. Note that when there are more than 1 feature, the fitting problem becomes an optimization problem in 3 or more dimensions as opposed to the 2-dimensional problem illustrated. However, the principles remain the same. Logistic Regression can also be extended to more than 2 classes, but the discussion of the procedure is beyond the scope of this book.

Recap of Logistic Regression Classifier

1. A classifier generates probabilities for the examples in a dataset. If the example belongs to class 1, it should generate a probability $p_1 \geq 0.5$, else it should generate a probability < 0.5.
2. A logistic regression classifier uses a sigmoid function, given by Equation 7.2.12 to generate the probabilities.
3. The parameters of the sigmoid are derived so as to maximize the number of correct predictions of the classes of the examples by generating a probability $p_1 \geq 0.5$ when the example belongs to class 1, and a probability < 0.5 when the example belongs to class 0.

7.4 EVALUATING CLASSIFICATION MODEL ACCURACY

For regression problems, we had defined an error metric called RSS to measure the model prediction errors over all the examples in the training dataset. To evaluate the model accuracy, we had defined other metrics such as RMSE and R^2. The same metrics are not appropriate for evaluating classification models. We will discuss some of the metrics which are used for classification problems in the next few sections.

We have seen that the classification algorithms generate class probabilities. Class labels are derived from the class probabilities by selecting the label for that class for which the estimated probability is the highest. We illustrate this with a synthetic example consisting of 4 target classes. Note that since there are four classes, for each example, the classification algorithm will generate a probability for each of the four classes, *i.e.*, probabilities (p'_0, p'_1, p'_2, p'_3) for the i^{th} example.

X	Y	p_0	p_1	p_2	p_3	$\hat{y}$
features^0	$y^0 = 2$	$p^0_0 = 0.2$	$p^0_1 = 0.1$	$p^0_2 = 0.25$	$p^0_3 = 0.45$	$\hat{y}^0 = 3$
features^1	$y^1 = 1$	$p^1_0 = 0.35$	$p^1_1 = 0.4$	$p^1_2 = 0.15$	$p^1_3 = 0.1$	$\hat{y}^1 = 1$
features^2	$y^2 = 0$	$p^2_0 = 0.26$	$p^2_1 = 0.11$	$p^2_2 = 0.36$	$p^2_3 = 0.27$	$\hat{y}^2 = 2$
features^3	$y^3 = 3$	$p^3_0 = 0.15$	$p^3_1 = 0.18$	$p^3_2 = 0.28$	$p^3_3 = 0.39$	$\hat{y}^3 = 3$

predicted probabilities — derived label

Figure 7.4.1 Predicted probabilities for a multi-class classification and corresponding labels

Suppose we were to use the RSS using the actual class y and the predicted class $\hat{y}$ in the formula for RSS. Refer to the examples in Figure 7.4.1. We see that there are two errors, one is for example 0, where the predicted class is 3 while actual class is 2. The residual square error is $(3 - 2)^2 = 1$. Example 2 has also been predicted incorrectly. In this case, the residual square is $(0 - 2)^2 = 4$. It seems that RSS is penalizing the second error significantly more than the first. However, both errors are equally significant. So why is RSS penalizing the second error more than the first one? This is happening because the class labels being used have a numeric ordering while classes really do not have any order. We see that RSS based on class labels or their numeric value is not an appropriate metric for measuring classification errors.

We will consider an error metric called cross-entropy which has been shown to work well for classification problems. It is based on the observation that instead of evaluating the predicted class label for errors, we can also evaluate the predicted probabilities to measure model error. We will revert to two-class classification to explain the intuition behind cross-entropy.

Consider the Table in Figure 7.4.2 where we have listed values of a function $-\log(p)$, where p is a value between 0 and 1. When p is between 0.0 and 1.0, $\log(p)$ is a negative number. $\log(0)$ is $-\infty$ and $\log(1)$ is 0.0. So, as p changes from 0.0 to 1.0, $-\log(p)$ changes from a very large positive number to a small positive number. In general, we can say that value of $-\log(p)$ for p greater than 0.5 is low, while values of $-\log(p)$ for values of p less than 0.5 are high. This can be seen in Table given in Figure 7.4.2.

When an example belongs to class 1, if the model predicted p_1 is greater than 0.5, then we accept the example as belonging to class 1. The closer the model predicted p_1 is to 1.0, the more confident we are of the model's prediction. Hence, it would be appropriate to assign a lower value to the model error, the closer p_1 is to 1.0. But that is exactly what $-\log(p_1)$ does. Hence, $-\log(p_1)$ is a reasonable error metric for examples belonging to class 1.

Likewise, $-\log(p_0)$ is a reasonable error metric for examples belonging to class 0.

Class	p_0	p_1	$-\log(p_0)$	$-\log(p_1)$
1	0.001	.999	0.001	6.91 (High error)
1	0.01	.99	0.01	4.6 (High error)
1	0.2	.80	.0.22	1.61 (High error)
...				
0	.9999	.0001	9.21 (High error)	0.0001
0	.999	.001	6.91 (High error)	.001
0	.90	.1	2.3 (High error)	.10
0	.80	.2	1.61 (High error)	0.22

Figure 7.4.2 Illustrating variation of the $-\log(p_0)$ and $-\log(p_1)$ with p_0/p_1

From the table above one can observe that Class = 1, $-\log(p_1)$ has significantly higher values, whereas for the Class = 0, $-\log(p_0)$ has significantly higher values. Hence, if we take the sum of all errors of class 1, which is negative $\log(p_1)$ and take the sum of all errors of class 0 , which is negative $\log(p_0)$, and average them out, then we have arrive at a cost function which our optimizer will try to minimize.

The **cross-entropy** loss function is formally defined below:

$$J(\beta) = -\frac{1}{m} \times \left(\sum_{y_i=0} \left(\log\left(p_0^i\right) \right) + \sum_{y_i=1} \left(\log\left(p_1^i\right) \right) \right) \qquad \textit{Equation } \mathbf{7.4.1}$$

Note that for the i^{th} example, if the class if 0, then the term $-\log(p'_0)$ is added to the metric and if the class is 1, then the term $-\log(p'_0)$ is added to the metric. The sum of all the terms is finally divided by $1/m$. It has been found that if the model parameters are determined to minimize the cross-entropy metric, then a

high-quality model is obtained. One can consider the cross-entropy metric as the classification equivalent of the RSS metric for the regression problem.

7.4.1 Determining Optimal Model Parameters (Aka Training The Model)

We had briefly discussed how optimal model parameters are determined for a regression model in Section 2.3. We would like to repeat the discussion for our classification model.

We have decided a model for the classification dataset – Logistic Regression in this case – which uses the sigmoid function. This model has parameters β which determine the class probabilities for the different examples. We have defined an error metric called the cross-entropy which is a measure of the model error over all the training examples.

Training the model thus involves finding the values for the model parameters β which minimize the cross-entropy metric. There are ways of efficiently determining the optimal parameters to minimize model error, for example, by using the gradient descent method or one of its variants.

7.4.2 Evaluating Model Quality

Recall that for regression, we used RSS to determine the optimal values of the model parameters. However, for human evaluation of the resultant model, we defined metrics such as RMSE and R^2. We will consider a very simple metric for Classification problems called Accuracy in this section. *A more sophisticated method for evaluating a classifier called the Confusion Matrix will be discussed in a subsequent chapter.*

Accuracy is simply defined as the number of correct predictions by the total number of predictions. So, if there are m examples, and the number for which the labels predicted correctly are n, then:

$$accuracy = \frac{n}{m} \qquad \textit{Equation } \mathbf{7.4.2}$$

We will soon see that this metric is not adequate in some situations. But for now, we will use the accuracy metric in our first Hands-on Exercise.

7.5 HANDS-ON EXERCISE: CLASSIFICATION WITH LOGISTIC REGRESSION

Building a classification model is not too different from building a regression model. Some of the principles that we had discussed for regression such as overfitting, underfitting, and train/test split also applies to classification models equally. So, we will reuse those ideas in this Hands-on Exercise.

The example we will use is the tumor classification problem where we need to classify whether a tumor is benign or malignant [*Source*: UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer]

Step 1.

Load the Data.

```
In [3]: # Importing required Libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [4]: # Loding the dataset into pandas dataframe.
        df = pd.read_csv('../Data/Breast_Cancer_Diagnostic.csv')
```

Step 2.

Check the features and the target variable.

```
In [5]: # print all the available features.
        df.columns

Out [5]: Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_
                mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
                'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
                'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
                'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
                'fractal_dimension_se', 'radius_worst', 'texture_worst', 'perimeter_worst',
                'area_worst', 'smoothness_worst', 'compactness_worst', 'concavity_worst',
                'concave points_worst', 'symmetry_worst', 'fractal_dimension_worst',
                'Unnamed: 32'],
               dtype='object')
```

The data has 30 odd features. We will use only 10 of them which we will select next. The target variable is 'diagnosis'.

Step 3.

Explore the features to be used.

For simplicity purposes, we have selected the 10 features which measure mean. One can consider all using all the features also.

In [6]:
```
# We will use only 10 of the 30 odd features. To get a feel for the features,
# take a sample from the dataset.
sample = df.sample(n=5, random_state=10)
```

In [7]:
```
# print the first five features that we will use and the target variable from the sample.
sample[["radius_mean", "texture_mean", "area_mean",
        "smoothness_mean", "diagnosis"]]
```

Out [7]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	diagnosis
172	15.460	11.89	102.50	736.9	0.12570	M
553	9.333	21.94	59.01	264.0	0.09240	B
374	13.690	16.07	87.84	579.1	0.08302	B
370	16.350	23.29	109.00	840.4	0.09742	M
419	11.160	21.41	70.95	380.3	0.10180	B

In [8]:
```
# print the next five features from the same sample.
sample[['compactness_mean', 'concavity_mean', 'concave points_mean',
        'symmetry_mean', 'fractal_dimension_mean', 'diagnosis']]
```

Out [8]:

	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	diagnosis
172	0.15550	0.203200	0.10970	0.1966	0.07069	M
553	0.05605	0.039960	0.01282	0.1692	0.06576	B
374	0.06374	0.025560	0.02031	0.1872	0.05669	B
370	0.14970	0.181100	0.08773	0.2175	0.06218	M
419	0.05978	0.008955	0.01076	0.1615	0.06144	B

```
In [9]: # Retain the 10 features and the target variable.
        df = df[['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
                 'smoothness_mean', 'compactness_mean', 'concavity_mean',
                 'concave points_mean', 'symmetry_mean', 'fractal_dimension_
                 mean', 'diagnosis']]
```

Step 4.

Check the data for missing values and other anomalies.

If there are missing values or other anomalies, they need to be fixed. We do a simple check for nulls only.

```
In [10]: # Check for nulls.
         df.columns[df.isnull().any()]

Out [10]: Index([], dtype='object')
```

We find that there are none to worry about.

Step 5.

Check the number of examples belonging to each class.

Sometimes, there can be many examples belonging to one class and only a few belonging to the second. Such datasets are called unbalanced. *Unbalanced datasets might require special handling (discussion of which is beyond scope of this book).*

```
In [11]: # Count the number of malignants and benigns in the dataset.
         df['diagnosis'].value_counts()

Out [11]: B    357
          M    212
          Name: diagnosis, dtype: int64
```

Our dataset seems reasonably balanced.

Step 6.

Separate the data into the features and target.

This will be required for training our logistic-regression model.

```
In [12]: # Load the features to a variable X
         # X is created by simply dropping the diagnosis column and retaining all others
         X = df.drop('diagnosis', axis = 1)

         # Load the target variable to y
         y = df['diagnosis']
```

Step 7.

Do the train/test split.

```
In [13]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)
```

Step 8.

Train the logistic regression model.

This is no different than what we did with Linear Regression. Except, we use a Logistic Regression model in place of Linear Regression.

```
In [14]: 1 # Let's create an instance for the LogisticRegression model and then train it with the training set
         2 from sklearn.linear_model import LogisticRegression
         3 Classifier = LogisticRegression(solver='liblinear')
         4 Classifier.fit(X_train, y_train)

Out [14]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                             intercept_scaling=1, l1_ratio=None, max_iter=100,
                             multi_class='warn', n_jobs=None, penalty='12',
                             random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                             warn_start=False)
```

Step 9.

Predict with the test set.

```
In [15]: 1 # Getting predictions from the model
         2 y_test_hat = Classifier.predict(X_test)
         3
         4 # Compare the predicted values with the actuals.
         5 Results = pd.DataFrame({'Actual': y_test, 'Predictions': y_test_hat})
         6 Results.head(5)
```

Out [15]:

	Actual	Predictions
421	B	B
47	M	B
292	B	B
186	M	M
414	M	M

Step 10.

Compute **model accuracy.**

```
In [17]: ▸ from sklearn.metrics import accuracy_score
          print(accuracy_score(y_test, y_test_hat))

          0.8888888888888888
```

Accuracy at 88% is fairly good.

Step 11.

Model **accuracy on training set.**

This will give us an idea if there is any underfitting or overfitting going on.

```
In [18]: ▸ # Get the predictions from the model for the training set.
          y_train_hat = Classifier.predict(X_train)
          print(accuracy_score(y_train, y_train_hat))

          0.9120603015075377
```

Accuracy is 3 percentage points better for the training set. It is not too different from what we got for the test set. Slightly better accuracy for the training set is on par for the course. We have just built our first classifier.

Step 12.

Checking model predicted probabilities.

Instead of directly finding the predicted class labels, we can also ask the classifier to return the predicted probabilities as below. It should be noted that whenever the predicted probability that the example belongs to class M is less than 0.5, the example is labeled as 'B', else it is labeled as 'M'.

In [18]:

```
# Getting probability predictions from the model.
y_test_hat_proba = Classifier.predict_proba(X_test)

print(y_test_hat_proba.shape))
```

```
(171, 2)
```

In [19]:

```
# The first column is the probability that the example belongs to the "B" or
# Benign class and the second column is the probability that the example belongs
# to the "M" or malignant class.

y_test_hat_proba[0:5,:]
```

Out [19]:

```
array([[0.62331451, 0.37668549],
       [0.83278577, 0.16721423],
       [0.95077545, 0.04922455],
       [0.01205439, 0.98794561],
       [0.1463116 , 0.8536884  ]])
```

In [20]:

```
1  # Compare the predicted values with the actuals along with the predicted
2  # probability that the class is "M".
3
4  pls = y_test_hat_proba[:,1]
5
6  Results = pd.DataFrame({'Actual': y_test, 'Predictions': y_test_hat, 'Prob(Class = M)': pls})
7
8  Results.head(5)
```

Out [20]:

	Actual	Predictions	Prob(Class = M)
421	B	B	0.376686
47	M	B	0.167214
292	B	B	0.049225
186	M	M	0.987946
414	M	M	0.853688

CHAPTER 7 EXERCISES

Review Exercises

1. When is a prediction problem a classification problem? Illustrate with two examples.
2. Classification models generate probabilities for the target variable. Explain. Why does this make sense?
3. How can class probabilities be converted into class labels?
4. Show that the range of a sigmoid is between 0 and 1.
5. How do the parameters β_0 and β_1 in Equation 7.2.6 control the shape and the position of the sigmoid?
6. How can we build a classifier for a dataset with multiple features by using a sigmoid? Assume a target variable with 2 categories.
7. RSS is not an appropriate error metric to minimize for a classifier. Why?
8. Explain the intuition behind using the cross-entropy error metric for building classifiers.
9. Explain the accuracy metric for evaluating the accuracy of a classifier.

Investigative Exercises

1. What are datasets with unbalanced classes? When can such datasets arise?
2. You are given a classifier, C2, which works only with two classes, say A and B. But you need to build a model for a dataset where the target variable has four classes, say A, B, C, and D. How can you use C2 to build a classifier for the given dataset?
3. Consider again a classifier for a 2-class classification problem with classes 0 and 1. Hence, the target variable y can have values 0 and 1. The classifier also outputs class labels, yhat, as 0 or 1. Show that RMSE in this case is the same as accuracy where accuracy is defined by the formula in Equation 7.4.1.

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#classification

8

Digging Deeper into Classifier Accuracy: The Confusion Matrix

8.1 INTRODUCTION

We have evaluated the accuracy of a classifier by comparing the number of correct predictions as a percentage of the total number of predictions. However, the percentage of correct predictions by each class of the target variable may be different from the overall percentage of correct predictions. Hence, it may be useful to find out the percentage of correct predictions by a class. A confusion matrix allows us to do just that, and a bit more. We will discuss the confusion matrix in detail in this chapter. Again, for illustrating the concept, we will use a 2-class classification problem.

Let us go back to the tumor prediction problem. We have two classes, benign and malignant. For the model that we built in Section 7.5, the corresponding confusion matrix is given below.

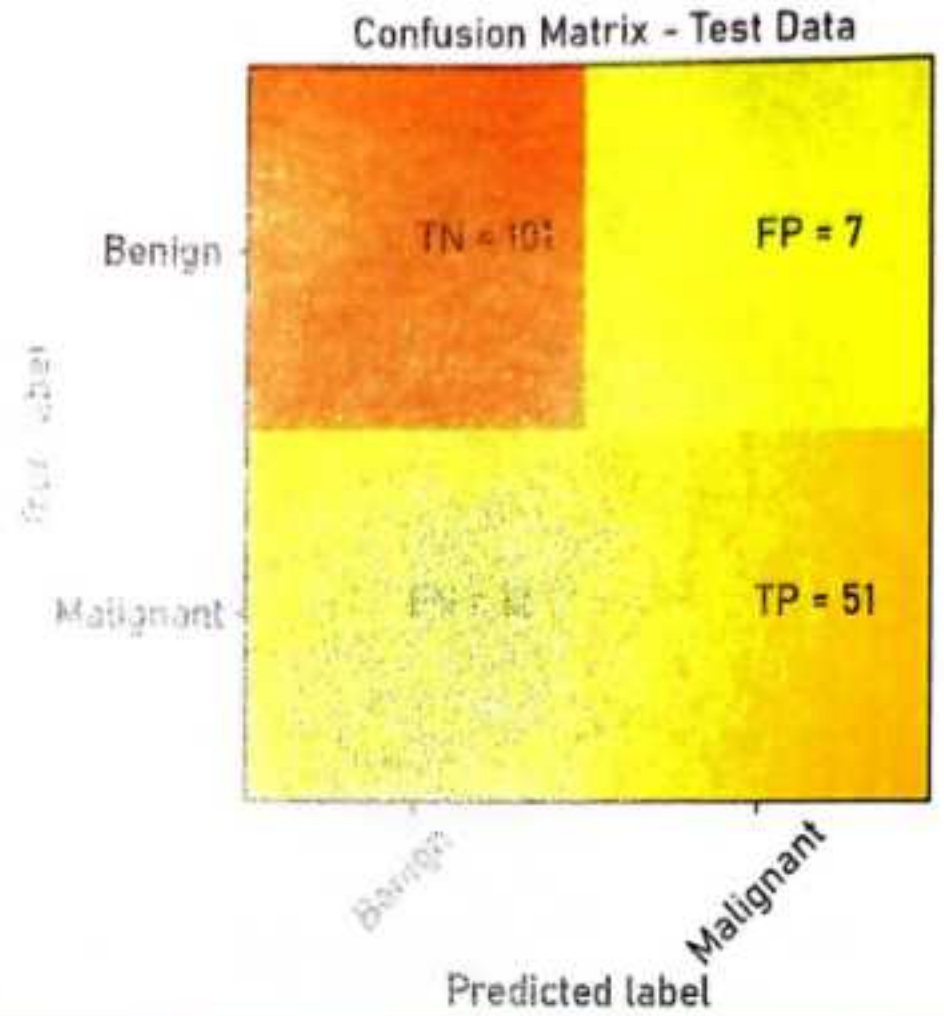


Figure 8.1.1 A confusion matrix for a 2-class classifier

The rows correspond to the true or actual classes of the examples in the test data. Consider the row which is labeled as 'Benign'. We see that there are two numbers, 101 and 7. Hence, in our test data, we have 108 (101 + 7) examples of benign tumors. Consider the row labeled as 'Malignant'. As per the matrix, we have 63 (12 + 51) examples of malignant tumors in our dataset. This also implies that the total number of examples in our dataset is 171 (108 + 63).]

The columns on the other hand correspond to values predicted by the model. The first column corresponds to predictions of 'Benign'. If we add up the numbers in this column, we find that our model has predicted that 113 (101 + 12) of the examples are benign tumors. As per the second column, our model has predicted that 58 (7 + 51) of the cases as malignant tumors.

Consider next the top left diagonal entry. It corresponds to examples whose true label is benign and predicted label is also benign. These are examples of benign cases which have been correctly predicted. The top right entry on the other hand corresponds to examples whose true label is 'Benign' but predicted label is 'Malignant'. These are incorrect predictions by our model. So, of the 108 examples which are benign, our model has predicted 101 of them correctly which

as a percentage is 93% (101*100/108). The remaining 7% (7*100/108) have been predicted incorrectly.

If we similarly evaluate the entries in the second row of the matrix, we see that the bottom left entry corresponds to examples where the actual label was 'Malignant' but was predicted to be 'Benign'. These are wrong predictions. The bottom right entry corresponds to 'Malignant' cases which were correctly represented as 'Malignant'. The prediction accuracy for the 'Malignant' class can now be calculated as 80% (51*100/63) while 20% of the examples are mis-predicted.

Finally, overall accuracy is 88% ((101 + 51)/(108 + 63)). We note that accuracy for the malignant class (80%) is much less than the overall accuracy. This was hidden by the higher accuracy (93%) of the benign class.

This is how a confusion matrix can be used to get insights into per class accuracy in addition to overall accuracy discussed earlier. Each entry in the matrix has a technical name for the 2-class classification case. Since these terms are very widely used, we spend some effort in explaining them.

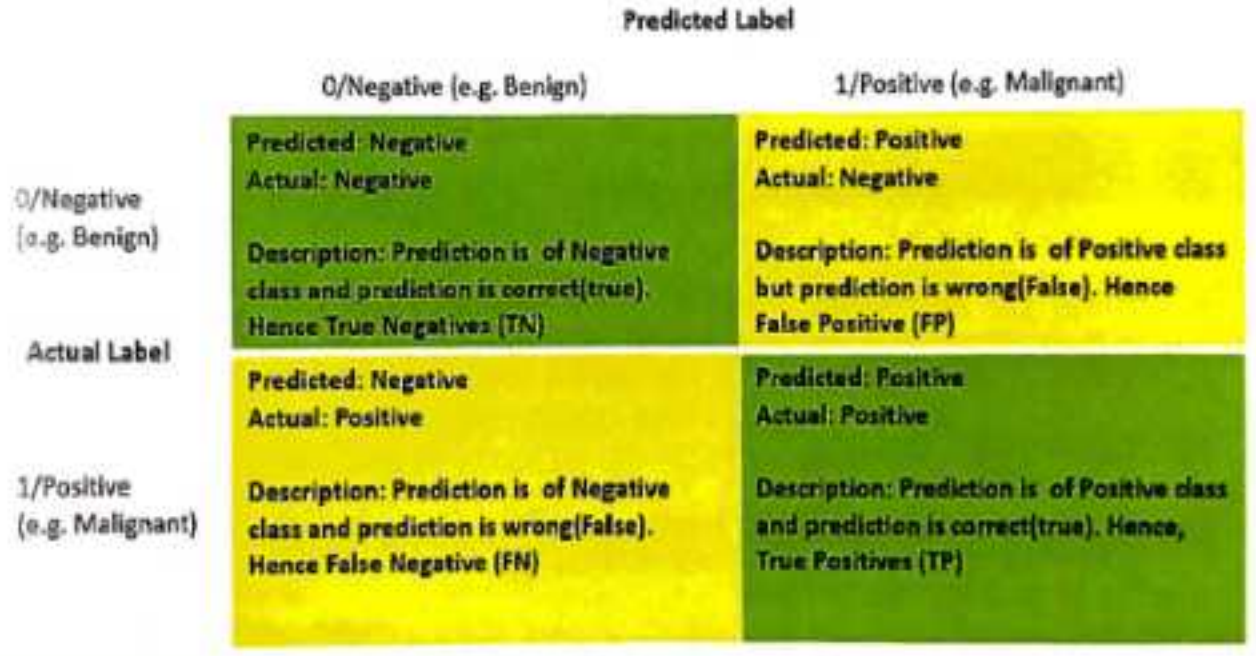


Figure 8.1.2 The interpretation of the various quadrants of a confusion matrix

We show a generic confusion matrix for the 2-class classification problem in Figure 8.1.2. Compare it with the specific one for the tumor prediction problem in Figure 8.1.1. Note that in this representation, the class corresponding to the 0th row and 0th column is called the Negative or 0 class. The class corresponding to the second row and the second column is referred to as the positive or 1 class.

The positive or '1' class is usually the class which is of greater interest (and has nothing to do with any sentiment associated with the class, as one will probably associate malignancy with negativity).

The cell on the top left corresponds to the negative class which has been predicted correctly or these are **true** predictions of the **negative** class. Hence, these are called the **true negatives (TN)**. Similarly, the cell on the top right corresponds to the false positives (FP), the one on the bottom left corresponds to false negatives (FN), and the one on the bottom right to the true positives (TP).

The accuracy for the negative class only, also known as the specificity, can be thus expressed as given here.

$$accuracy\ (negative\ class) = Specificity = \frac{TN}{TN + FP} \qquad \textit{Equation } \mathbf{8.1.1}$$

The accuracy for the positive class only, also known as the recall or sensitivity, can be thus expressed as below.

$$accuracy\ (positive\ class) = Recall\ of\ Sensiticvity = \frac{TP}{TP + FN} \qquad \textit{Equation } \mathbf{8.1.2}$$

The overall accuracy, simply called the accuracy, can be expressed as below.

$$accuracy\ (overall) = Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad \textit{Equation } \mathbf{8.1.3}$$

It should be remarked that there are convenient routines for generating the confusion matrix given the true labels and the predicted labels as we will see in the subsequent section.

There is one final accuracy metric we will discuss which is called **Precision**. The metric is defined as below.

$$Precision = \frac{TP}{TP + FP} \qquad \textit{Equation } \mathbf{8.1.4}$$

To understand the meaning of precision, consider again the confusion matrix for the tumor prediction problem. The number of correct and incorrect predictions for the benign class is different from that in the confusion matrix in Figure 8.1.1. Recall is 80%, which means if there are 100 malignant cases, our model will detect 80 of them. Our model is good in identifying the malignant cases. However, if

we look at the precision value which is 0.46, our model identifies many of the benign cases incorrectly. So, if our model says that an example is malignant, the model is wrong more than 50% of the time. So based on model predictions, we can catch most of the malignant cases (high recall). But if the model says someone is malignant, we do not have confidence that the tumor is actually malignant (low precision). So precision is a measure of the confidence that the prediction given by a classifier is indeed a correct prediction.

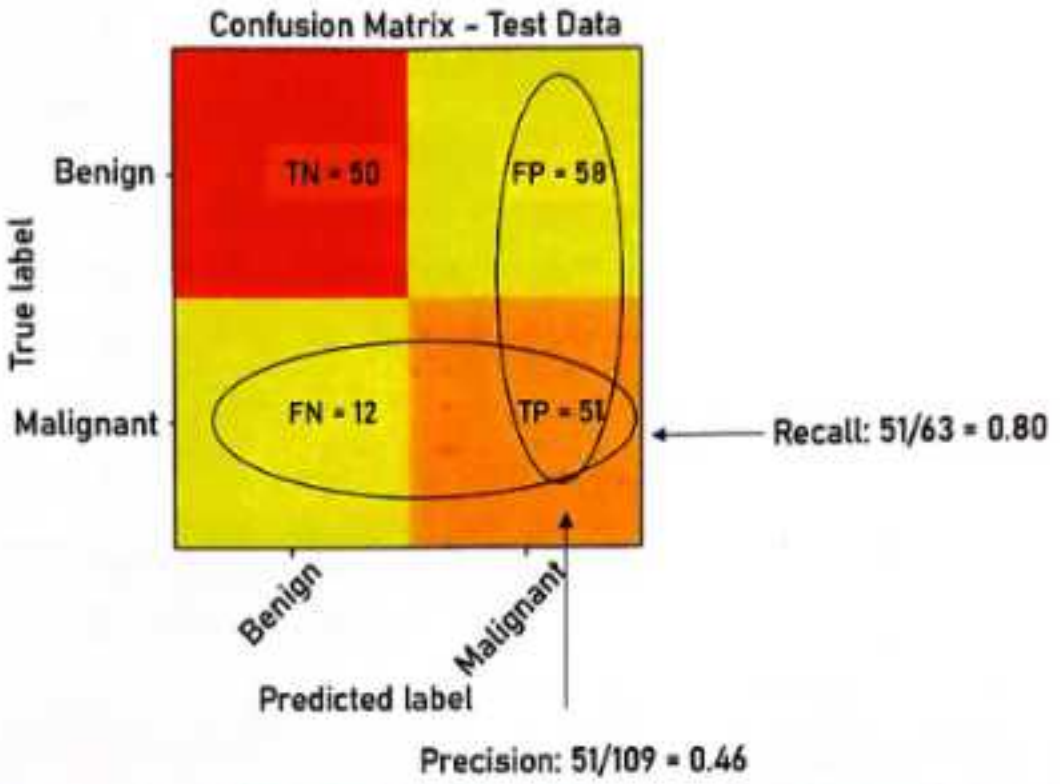


Figure 8.1.3 Illustration of Precision and Recall

Precision and Recall are also combined into a single metric which is called the F_1 score. F_1 is the harmonic mean of precision and recall and one can get a high F_1 score only when both precision and recall are high.

$$F_1 = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{TP}{TP + \frac{FN + FP}{2}} \qquad \textit{Equation } \mathbf{8.1.5}$$

With all these metrics handy, we are in a much better position to evaluate a classifier.

8.2 HANDS-ON WITH CONFUSION MATRIX

Deriving the confusion matrix is simple since there is handy function for computing it. Computing the recall, specificity etc is slightly more cumbersome. The code for all of the accuracy calculation for our tumor classifier is shown below.

Step 1.

Build the model and predict with it on the test set.

These steps are a repeat of the Hands-on Exercise in Section 7.5.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset into pandas dataframe.
df = pd.read_csv('../Data/Breast_Cancer_Diagnostic.csv')

# Retain the 10 features and the target variable.
df = df[['radius_mean', 'texture_mean', 'perimeter mean', 'area_mean',
         'smoothness_mean', 'compactness_mean', 'concavity mean',
         'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
         'diagnosis']]

# Load the features to a variable X
# X is created by simply dropping the diagnosis column and retaining all others
X = df.drop('diagnosis', axis = 1)

# Load the target variable to y
y = df['diagnosis']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)

# Let's create an instance for the LogisticRegression model and then train it with the
training set.
from sklearn.linear_model import Logistic Regression
Classifier = Logistic Regression()
Classifier.fit (X_train, y_train)

# Getting predictions from the model
y_test_hat = Classifier.predict (X_test)
```

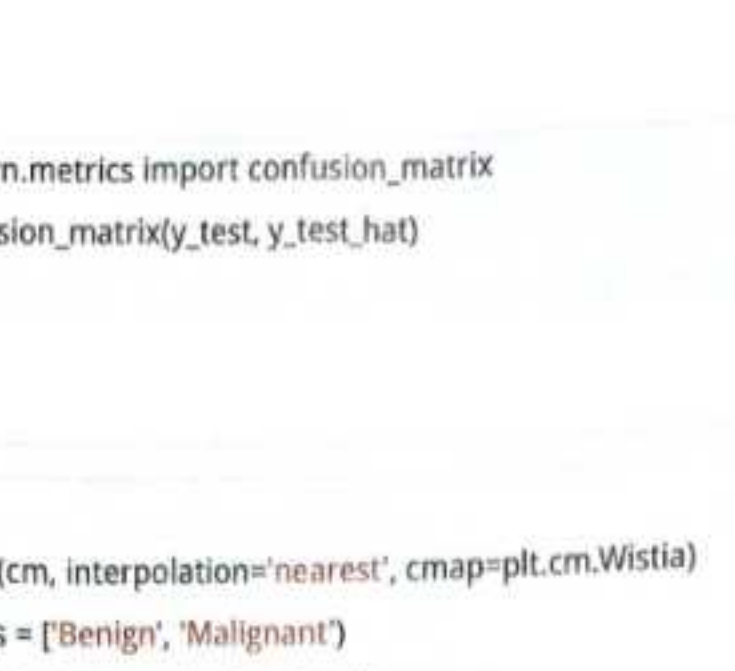
tep 2.

Generate the confusion matrix.

The Sklearn library has a handy function for computing the confusion matrix. We also provide some code to pretty print it — this is not necessary but helps in interpretation.

In [2]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_test_hat)
print (cm)
[[101 7]
 [12 51]]
```

In [3]:
```
plt.clf()
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
classNames = ['Benign', 'Malignant']
plt.title('Confusion Matrix - Test Data')
plt.ylabel('True label')
plt.xlabel('Predicted label')
tick_marks = np.arange (2)
plt.xticks(tick_marks, classNames, rotation=45)
plt.yticks (tick_marks, classNames)
s = [['TM', 'FP'], ['FM', 'TP']]
for i in range (2):
    for j in range ( 2):
        plt.text(j, i, str(s[i][j])+" = "+str(cm[i][j]))
plt.show()
```



Step 3.

Calculate common error metrics for a 2-class classifier.

In [5]:
```
1 from sklearn.metrics import classification_report
2 print( Classification_report(y_test, y_test_hat) )
```

	precision	recall	f1-score	support
B	0.89	0.94	0.91	108
M	0.88	0.81	0.84	63
accuracy			0.89	171
macro avg	0.89	0.87	0.88	171
weighted avg	0.89	0.89	0.89	171

As one can see above, it is easy to report the class-wise and overall metrics of precision, recall, and f1-score by simply using the convenient reporting function in Python's Sklearn.metrics libraries. The results can be verified by taking individual values of the confusion matrix and performing the appropriate calculations.

8.3 IMPORTANCE OF CLASS-WISE ACCURACY

If all the metrics, recall, specificity, precision improve due to modeling steps (maybe by feature engineering, maybe a different classifier), then there is nothing not to like. But often, one of them can be improved only at the cost of the other. In such cases, which metrics should you focus on?

The answer is somewhat simplified if the assumption is that the positive class is the more important class. Let us consider the tumor malignancy prediction challenge. The reasoning used is as follows. If malignancy is not detected correctly, then the patient condition could worsen and could potentially lead to mortality. If benign tumors are detected incorrectly as malignant, it could lead to psychological trauma and additional expenses due to extra tests, but not mortality. Hence, inaccuracy in detection of benign cases may be more acceptable than inaccuracy of detection of the malignant cases. So, if we assume that malignant class is more important and is the positive class, then we would prefer to improve recall over specificity.

Let's consider a second case of email spam detection. If non-spam emails are incorrectly predicted as spam, then they would go to the spam folder. If spam emails are incorrectly predicted as non-spam, then they would go to the inbox. If we agree that non-spam emails going to the spam folder is less acceptable than spam

emails appearing in the inbox, then it implies that detecting non-spams accurately is more important. If we assume non-spam emails as the negative class and the spam emails as the positive class, then in this case the negative class is more important to us. So, we would want to trade off better specificity with lower recall.

When is precision important? Consider the telecaller department of a business which needs to make calls to possibly tens of thousands of potential customers. A very large percentage of the calls are unsuccessful and only a few of the customers agree to buy the product being sold. Thus, a lot of time and resources are consumed in making the calls, but the returns are relatively low. Let our classes be not-buyer (negative) and buyer (positive). The telecalling department using a model to identify likely buyers of insurance will call from those predicted by the model as buyers (positive class) *i.e.*, it will consider the customers in the right-hand side column in Figure 8.3.1. Out of these, those in the True Positives (TP) will buy the product, the False Positives (FP) will not. To minimize the effort spent in calling, the telecalling department would like a model which maximizes the percentage of true positives (predicted buyers who will actually buy) as a fraction of the total predicted to buy. But this ratio is nothing but the precision of the model repeated below for the sake of convenience.

$$Precision = \frac{TP}{TP + FP} \qquad \textit{Equation } \mathbf{8.3.1}$$

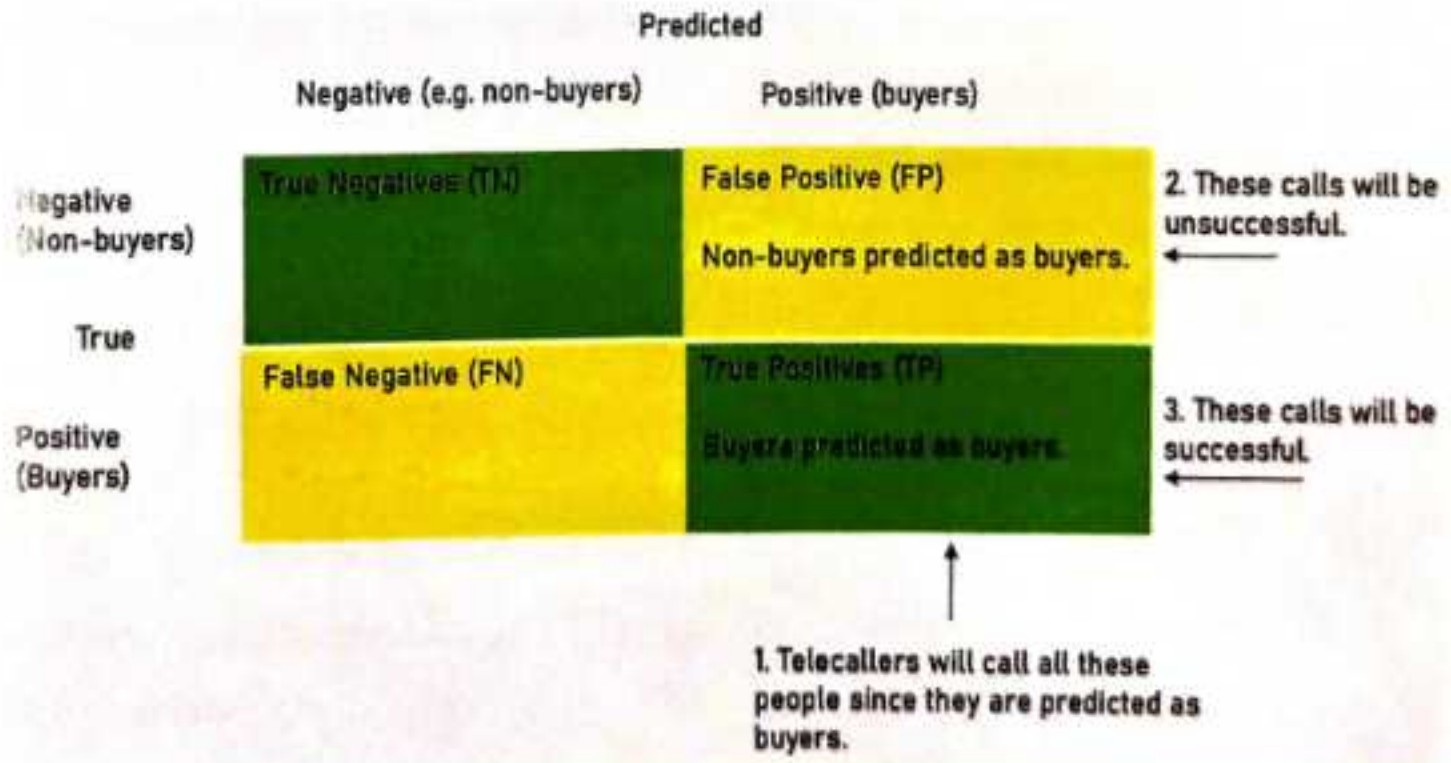


Figure 8.3.1 Illustrating the importance of Precision

CHAPTER 8 EXERCISES

Review Exercises

1. Accuracy is not an adequate metric to evaluate the quality of a classifier. Justify.
2. Explain a confusion matrix assuming a 2-class classifier.
3. What are the terms True Negative, True Positive, False Negative, and False Positive in the context of a confusion matrix? Draw a generic confusion matrix and label it.
4. What does precision, recall, and specificity measure?
5. Define accuracy in terms of FN, FP, TN, and TP.
6. In case of a 2-class classifier, we can define either one of the classes to be negative and the other to be positive. What is the usual practice?
7. Why is precision an important metric? Explain with an example.

Investigative Exercises

1. Suppose there are three classes, ['cow', 'cat', 'dog']. The confusion matrix will now be a 3 × 3 matrix. Draw the matrix and label the rows in the order the class labels appear in the list. Label the columns in the same order. Assume that row numbers are from 0 to 2 and column numbers are also 0 to 2. Write down the equation for recall and precision for the 'cat' class in terms of the elements of the matrix, *i.e.*, cm[i, j].
2. Let us say we have the confusion matrix given below for a 2-class classification problem.

   [99 1
   90 10]

   Let's assume that we are interested in the positive or class 1 more than the other. The model's precision is almost 91% (10/11). Is this model a good model? Discuss.
3. We want a model which predicts whether it is going to rain the next day. So the model has two classes, 'No-rain' (negative class or class 0) and 'Rain' (positive class or class 1). This model will be deployed in a region where it hardly rains. On the average, it rains 1 day out of every 100. The model

predicts every time that it will not rain. What is the model accuracy? What is its precision and recall? Discuss if the model is useful or not.

4. You are given a classifier which predicts class probabilities for a 2-class dataset, the classes being 0 and 1. The classifier predicts probability that the example is of class 1, *i.e.*, it returns pr(1) for a given example. You write an additional function which takes the predicted probability pr(1) and returns class labels.

```
function proba_to_labels_1(pr1):
        if (pr1 >= 0.5) return 1 else return 0
```

Another data scientist writes a slightly different converter function.

```
function proba_to_labels_2(pr1):
        if (pr1 >= 0.7) return 1 else return 0
```

What will be the likely implications on the precision and recall for the examples belonging to class 1 of using the second function?

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#confusion-matrix

9

Distance-Based ML Algorithms

9.1 INTRODUCTION

Let us consider a two-class classification problem with 2 features. We have a dataset which either belongs to the positive (1) or negative (0) class. And there are two features x_1 and x_2 which decide the class. A synthetic dataset is shown below in Figure 9.1.1 where points belonging to class 0 is colored blue, while those belonging to class 1 are colored red.

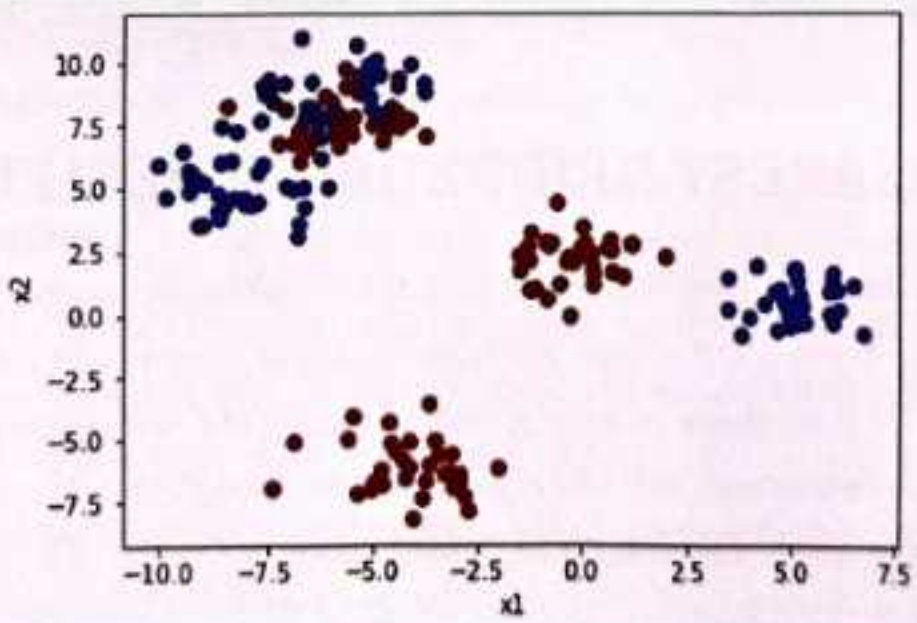


Figure 9.1.1 A synthetic dataset with two features x_1 and x_2 and two classes. The classes are indicated by the color of the plotted point.

Given new points such as p_1 and p_2 in Figure 9.1.2, we want a classifier to assign them to one of the classes. One straightforward approach could be to find the distance of the new points from the clusters made by the points in the training set. By this approach, p_1 is clearly closer to one of the red clusters, hence we classify it as 1. p_2 is a difficult one since it seems equally close to a blue and red cluster. Assuming it is closer to the blue cluster, we may classify it as belonging to class 0.

What we just did was use a distance metric to perform the required classifications. An algorithm which uses distance-based metric to perform classification is the K-Nearest Neighbors (KNN) algorithm. We will formalize KNN in the next section.

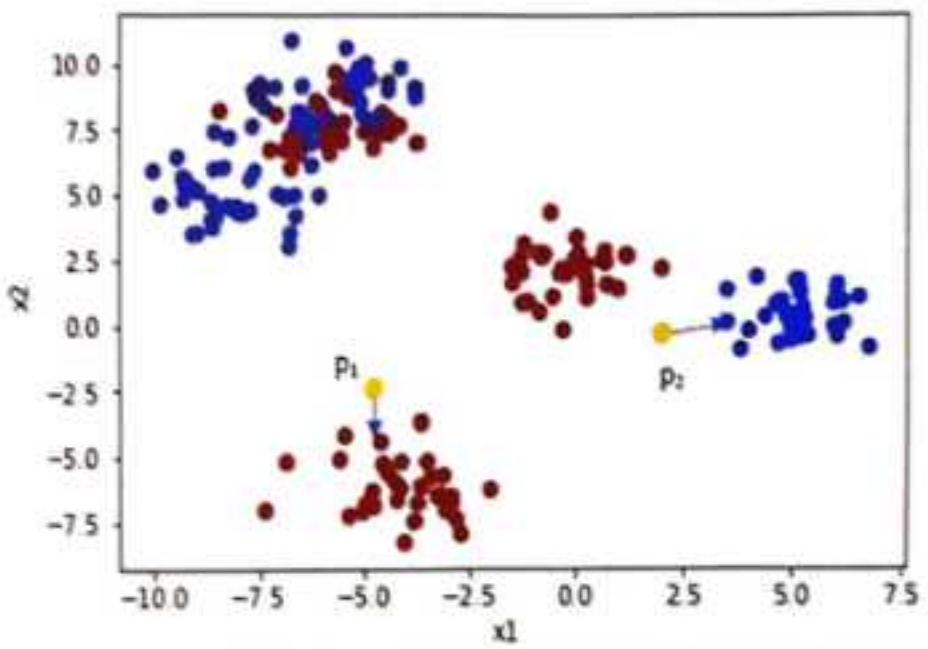


Figure 9.1.2 Illustration of use of distance for classification

9.2 K-NEAREST NEIGHBORS CLASSIFIER

The K-Nearest Neighbors (KNN) is a very simple classifier. Given a training set, it does nothing but stores the training set in memory (maybe in a table of some sort). That is, there is no training to figure out any parameters like linear and logistic regression did. The K in K-Nearest Neighbors is an integer that the user needs to specify as will be explained soon.

Let us say, the user has specified a value of 3 for *K*. To figure out the class of a new example (point), KNN finds the 3 points in the training set which are nearest to the new point. It next finds the class labels of each of these 3 nearest

points. The class label which occurs the maximum number of times amongst the three closest points is declared to be the class label of the new point.

It should be obvious that the number of points K which are evaluated is critical to the performance of the algorithm. If $K = 1$ as in Figure 9.2.1(*a*), then p_2 would be classified as red or 1. If $K = 3$ as in Figure 9.2.1(*b*), then p_2 would be classified as blue or 0.

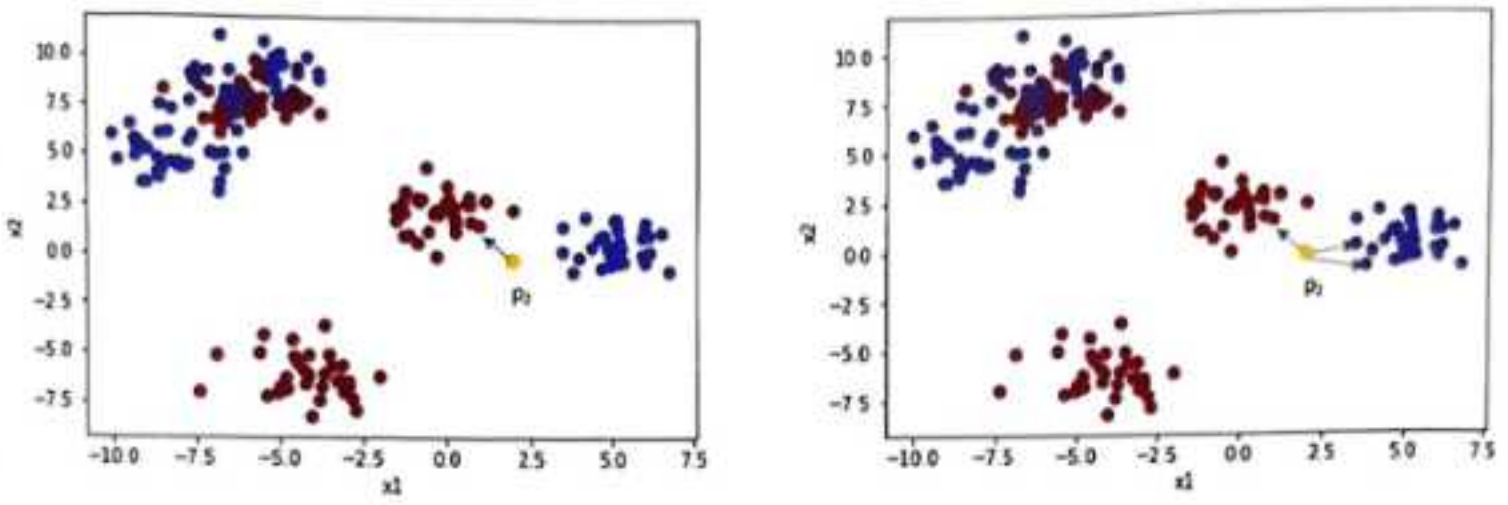


Figure 9.2.1 (a) The closest point used for deciding class of the point p_2 and (b) using the three closest points for deciding the class.

Since K has to be specified by the user, the value of K which gives the highest accuracy has to be determined. This is done as follows.

1. Do a train/test split of the given dataset.
2. For multiple values of K, use the KNN model to make predictions for the test set.
3. Evaluate the accuracy on the test set for each value of K.
4. Select the value of K which gives maximum accuracy.

Hyperparameters

The K parameter in KNN is not called a parameter but a hyperparameter. Contrast this with the parameters of a linear or logistic regressor. What is the difference between a hyperparameter and parameter?

One way of understanding the difference (not necessarily the only way) is as follows. The parameters in linear regression model are intrinsic to the model and are determined during model training. A hyperparameter on the other hand is determined during the testing phase.

The value of the hyperparameter which produces the most acceptable error is used as the hyperparameter value.

9.3 HANDS-ON WITH K-NEAREST NEIGHBORS CLASSIFIER

Distance Metric

The most familiar method of computing the distance between two points is the Euclidean distance. But it should be noted that other distance metrics such as 'Manhattan distance' also exist and may be used.

Minkowski distance is a generalization of the Euclidean distance. In Euclidean distance, the square of the difference of the coordinates is used to measure the distance. In Minkowski distance, the p^{th} power is used.p = 2 reduces Minkowski distance to Euclidean distance.

We will be applying KNN to the same Tumor diagnosis dataset that we have used earlier. Even though this might be a bit boring, it allows comparison of various classifiers on a given dataset. Such comparisons between different algorithms are a must for any machine learning project.

Caveat

There is a very important step called Scaling which needs to be applied to the data before running the KNN algorithm. We will discuss scaling in a subsequent chapter and show its effect on KNN. For this current Exercise, we will not include the scaling step.

Step 1.

Read the data.

Load the features to be used. Check for missing values. This is a repeat of the steps in Section 7.5.

```
In [1]: # Importing required Libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: # Loding the dataset into pandas dataframe.
        df = pd.read_csv('../Data/Breast_Cancer_Diagnostic.csv')
```

```
In [3]: # Retain the 10 features and the target variables.
        df = df[['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
                 'smoothness_mean', 'compactness_mean', 'concavity_mean',
                 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
                 'diagnosis']]
```

```
In [4]: # Check for nulls.
        df.columns[df.isnull().any()]
Out[4]: Index([], dtype='object')
```

Step 2.

Create the feature(X) and target(y) variables.

```
In [5]: # Load the features to a variable X
        # X is created by simply dropping the diagnosis column and retaining all others
        X = df.drop('diagnosis', axis = 1)

        # Load the target variable to y
        y = df['diagnosis']
```

Step 3.

the train/test split.

```
In [6]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=1
```

Step 4.

Train the classifier.

During training, KNN simply stored all the training points in an internal data structure. Also note that by default, Minkowski metric with p=2 implies that Euclidean distance will be used for classification. **Note that a value of K = 3 is being used.**

```
In [7]: # Let's create an instance for the KNN model and then train it with the training set.
        from sklearn.neighbors import KNeighborsClassifier

        model = KNeighborsClassifier(n_neighbors=3)

        # Train the model using the training sets. This does nothing other than store the points
        # in some internal data structure.
        model.fit(X_train, y_train)
Out[7]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                             weights='uniform')
```

Step 5.

Predict on the test set.

```
In [8]: 1 # Getting predictions from the model
        2 y_test_hat = model.predict(X_test)
        3
        4 # Compare the predicted values with the actuals.
        5 Results = pd.DataFrame({'Actual': y_test, 'Predictions' : y_test_hat})
        6
        7 Results.head(5)
```

Out [8]:

	Actual	Predictions
421	B	B
47	M	B
292	B	B
186	M	M
414	M	M

Step 6.

Evaluate Model Accuracy.

```
In [9]: 1 from sklearn.metrics import confusion_matrix, recall_score, precision_score
        2
        3 cm = confusion_matrix(y_test, y_test_hat)
        4 print(cm)
[ [98 10]
 [13 50] ]
```

```
In [10]: ▸ 1 from sklearn.metrics import classification_report
         2
         3 print(classification_report(y_test, y_test_hat))
```

	precision	recall	f1-score	support
B	0.88	0.91	0.89	108
M	0.83	0.79	0.81	63
accuracy			0.87	171
macro avg	0.86	0.85	0.85	171
weighted avg	0.86	0.87	0.86	171

Step 7.

Repeat with K = 7.

```
In [17]: ▸ 1 model = KNeighborsClassifier(n_neighbors=7)
         2
         3 # Train the model using the training sets.
         4 model.fit(X_train, y_train)
         5
         6 # Getting predictions from the model
         7 y_test_hat = model.predict(X_test)
         8
         9 cm = confusion_matrix(y_test, y_test_hat)
        10 print(cm)
        11
        12
[[100   8]
 [ 14  49]]
```

```
[18]: ▸ 1
       2 print(classification_report(y_test, y_test_hat))
```

	precision	recall	f1-score	support
B	0.88	0.93	0.90	108
M	0.86	0.78	0.82	63
accuracy			0.87	171
macro avg	0.87	0.85	0.86	171
weighted avg	0.87	0.87	0.87	171

Note that precision and specificity has improved while recall has degraded compared to the model with $K = 3$. Depending upon which metric is important in our application, we will need to decide between $K = 3$ or $K = 7$.

It is also instructive to compare the KNN with the Logistic Regressor in Section 8.2. The Logistic Regression classifier is giving better accuracy better for the dataset we have and for the hyperparameter values we have chosen. We will make KNN outperform Logistic Regression in a subsequent Hands-on Exercise

KNN for regression

KNN can be very easily applied to regression problems. Let us say $K = 3$. Then, for a new point for which the target y needs to be estimated, KNN will find the nearest 3 points and simply return the average of the y's of the three nearest points as the $\hat{y}$ for the new point.

CHAPTER 9 EXERCISES

Review Exercises

1. Define the distance between two points in a dataset. Assume that the dataset has features x_1, x_2, and x_3 and the target variable is y.
2. Distances between points in a dataset can be used for classification. Explain the general principle involved with a diagram.
3. Explain how KNN works for classification.
4. What does the K in KNN stand for? How is it determined?

Investigative Exercises

1. One can see in the output of Step 4 of the Hands-on Exercise that the KNN classifier has many hyperparameters such as algorithm, leaf_size, metric, etc. Run the same ML pipeline as given in Section 9.3 with different values of the hyperparameters.

Supporting Material

https://www.ascribe.ai/machine-learning/less-substance-based-ml-algorithms

Let's look at this data below:

Example	Age	Salary
Trainee	25	100000
Manager1	30	140000
Manager2	35	140000

The distance between Trainee and Manager1 is sqrt(5^2 + 40000^2)
= sqrt(1600000025) = 40000.0003
The distance between Trainee and Manager2 is sqrt(10^2 + 40000^2)
= sqrt(1600000100) = 40000.001

One can observe that the difference in age is hardly making any material difference to the distance of the Trainee from either of the Managers. Manager2 is 10 years older to the trainee, but this distance calculation is being dwarfed by the large magnitude of the squared salary term.

When we use an algorithm such as KNN which uses distance-based metrics to provide the predictions, as opposed to say Logistic Regression which does not use distance-based metrics, then we need to scale the features to make them comparable to each other.

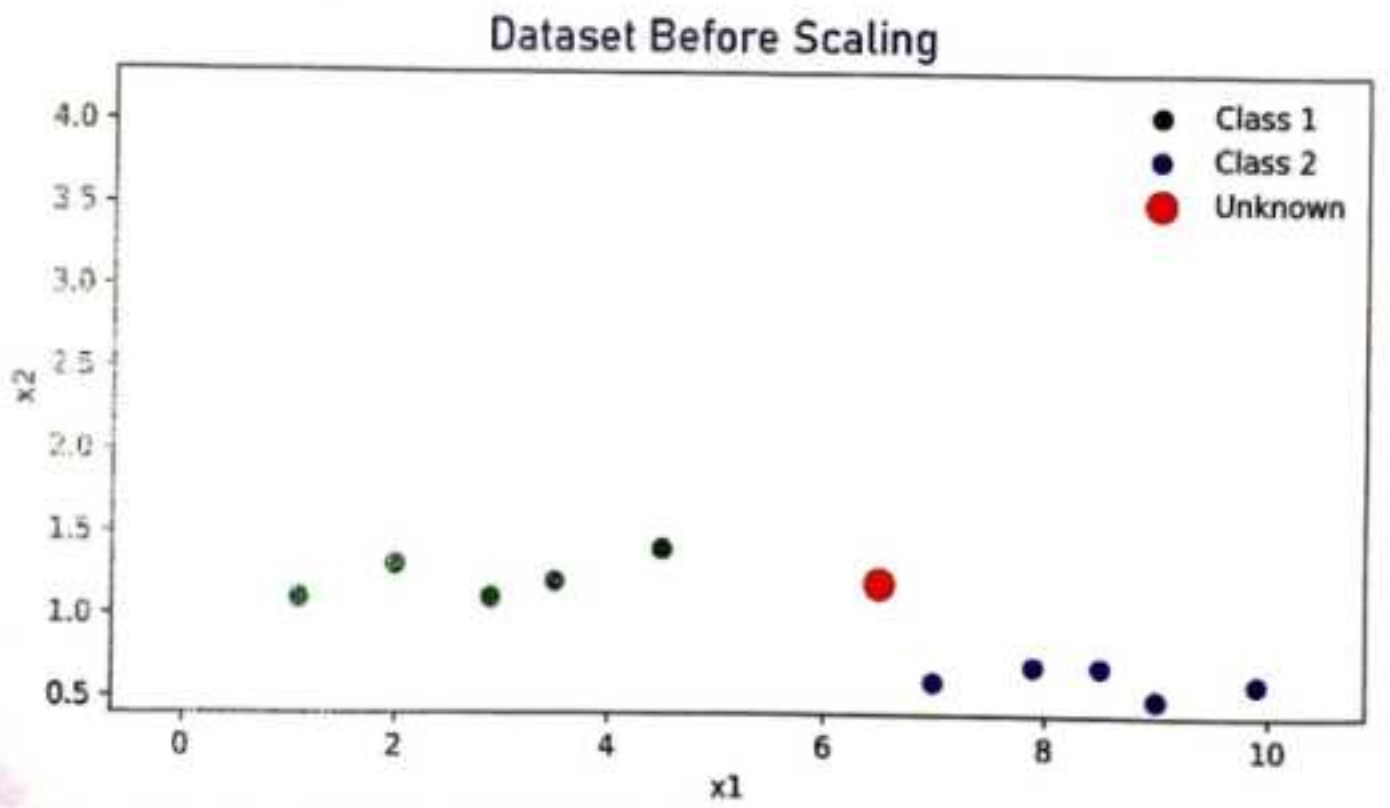


Figure 10.1.1 Synthetic dataset with two features x_1 and x_2. The feature x_1 has a large range while x_2 has a small range.

Let's consider a synthetic example to demonstrate the effect of scaling on classification results. Consider Figure 10.1.1 below which shows two features x_1 and x_2 along the horizontal and vertical axis. The data is categorized into two classes, the green and the blue class. We are given a new point, the red point, to classify. The red point appears and is closer (if we use Euclidean distance to measure closeness) to a blue point than a green point. Hence, if we were to use KNN with $K = 1$, we could classify our red point as belonging to the blue class.

Note that in Figure 10.1.1, the range of the x1 variable was about 10 (0 to 10) while the range of the x2 variable was comparatively small at 1.5 (0 to 1.5). Let us assume that the x1 variable was like our salary feature and we decided to scale it down to have a smaller range by using the equation below where we used a constant 3.19 which also happens to be the standard deviation of X1:

$$X1 \text{ Scaled} = \frac{X1}{3.19} \qquad \textit{Equation } \mathbf{10.1.2}$$

Since X2 had a smaller range, we decided to scale it up to make the values comparable to the values of X1. To do that, we use the formula below where the constant (0.33), the standard deviation of X2 was used.

$$X2 \text{ Scaled} = \frac{X2}{0.33} \qquad \textit{Equation } \mathbf{10.1.3}$$

The scaling applied to all the points including the green, red, and blue points in Figure 10.1.1 gives us the Figure 10.1.2.

Two things need to be noted. First, dividing each feature by its standard deviation makes the range of each feature very similar (0 to 3 for x1 Scaled and 1.5 to 4.5 for x2 Scaled). Second, the red point now appears closer to the green class. If KNN were applied to the scaled data with $K = 1$, it would classify the red point as belonging to the green class rather than the blue class.

Which classification is correct? The one without scaling or the one after scaling? Scaling removes the effect of the scale used to measure a parameter (*e.g.*, cm versus m, Fahrenheit versus Celsius, $ versus INR, etc.).

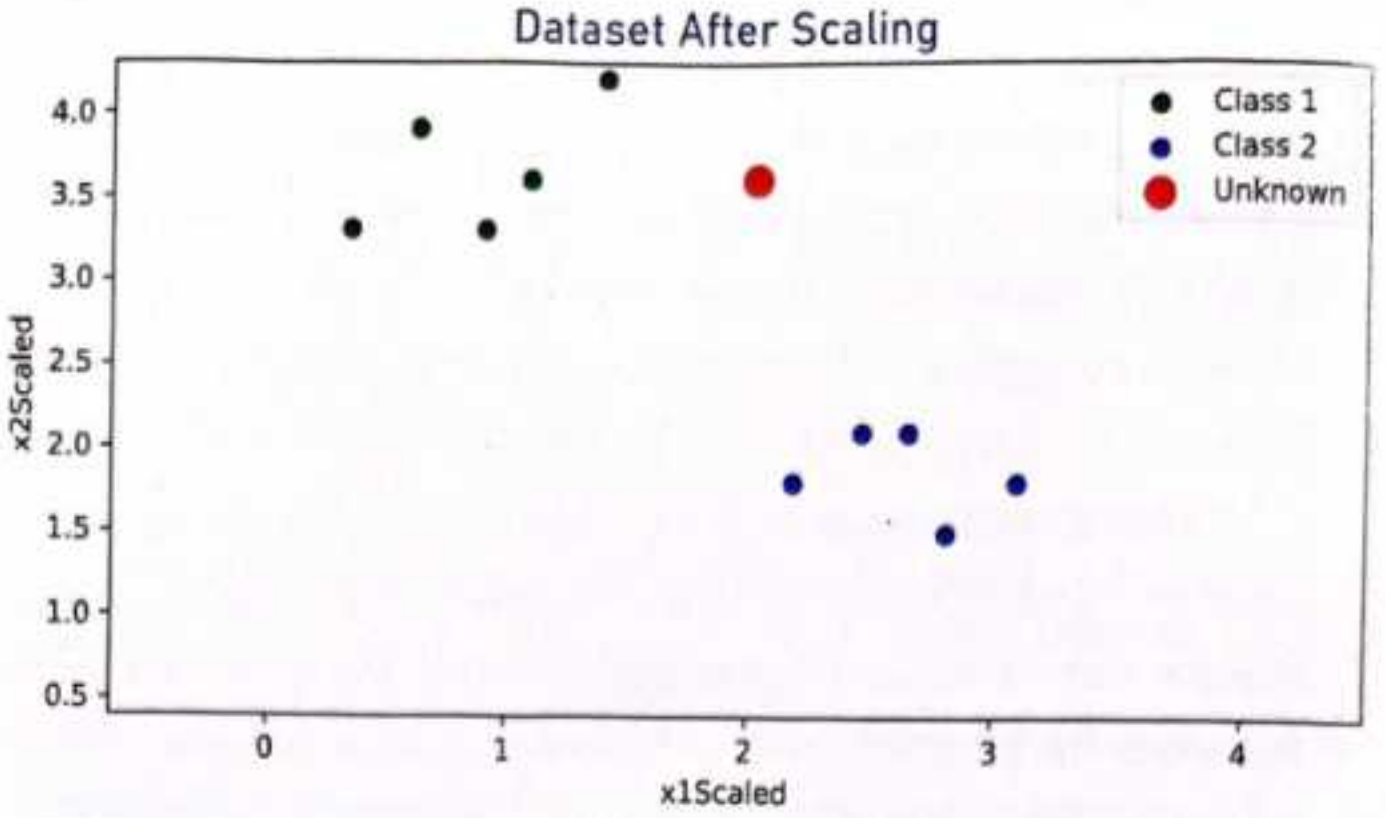


Figure 10.1.2 The dataset of Figure 10.1.1 scaled so that both features have similar ranges

In another interpretation, scaling removes the added importance features which have larger values get when distance-based measurements are used. Hence, in most cases, scaling is expected to give better results compared to using data without scaling. To be on the safe side, one can always build a model with and without scaling, and then determine model accuracy on a test dataset to determine whether scaling is effective for the application in hand.

10.2 SCALING FORMULAS

The formula we used for scaling, which is simply dividing the feature values with the standard deviation of the feature can be used for scaling purposes. However, the following two formulas are more commonly used for scaling purposes.

$$X \text{ scaled} = \frac{X - X_{mean}}{X_{SD}} \qquad \textit{Equation } \mathbf{10.2.1}$$

When we use the above formula, we say we have **normalized** the variable. By subtracting the mean value from each observation and then dividing the result by the standard deviation, the transformed variable can be shown to have a mean of 0.0 and a standard deviation of 1.0.

The second common method of scaling is called min-max scaling and uses the formula given below.

$$X \text{ scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad \textit{Equation } \mathbf{10.2.2}$$

Min-max scaling ensures that the range of the transformed variable is always between 0 and 1.

Both types of scaling, when applied to features which are continuous, have the effect of transforming their ranges to very comparable values. One can try both to evaluate their effect on the modeling results.

NOTE

Scaling is applied to features which are continuous variables.

For categorical variables which are converted to discrete numbers, it may be more appropriate to use one-hot encoding rather than scaling. There is also no benefit of applying scaling to the outcome variable even if it is continuous. It's the features which compete for importance when building a model, and not the outcome variable. So scaling is used to ensure that no feature is given additional importance for superfluous reasons like the scale it was measured in.

10.3 HANDS-ON EXERCISE: KNN WITH SCALING

We re-apply the KNN algorithm on the tumor prediction problem, this time with scaled dataset.

Step 1.

Load the dataset and prepare the train/test split.

In [1]:
```
# Importing required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Loding the dataset into pandas dataframe.
df = pd.read_csv('../Data/Breast_Cancer_Diagnostic.csv')
```

In [3]:
```
# Retain the 10 features and the target variable.
df = df[['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
         'smoothness_mean', 'compactness_mean', 'concavity_mean',
         'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
         'diagnosis']]
```

In [4]:
```
# Check for nulls.
df.columns[df.isnull().any()]
```

Out[4]:
```
Index([], dtype='object')
```

In [5]:
```
# Load the features to a variable X
# X is created by simply dropping the diagnosis column and retaining all others
X = df.drop('diagnosis', axis = 1)

# Load the target variable to y
y = df['diagnosis']
```

In [6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)
```

Step 2.

Train a KNN model with K = 7.

This is same as in the Hands-on Exercise in Section 9.3.

In [7]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

model = KNeighborsClassifier(n_neighbors=7)

# Train the model using the training sets.
model.fit(X_train, y_train)

# Getting predictions from the model
y_test_hat = model.predict(X_test)

cm = confusion_matrix(y_test, y_test_hat)
print(cm)
```

```
[[100   8]
 [ 14  49]]
```

Step 3.

Evaluate the model accuracy.

In [8]:
```
1 from sklearn.metrics import classification_report
2
3 print(classification_report(y_test, y_test_hat))
```

	precision	recall	f1-score	support
B	0.88	0.93	0.90	108
M	0.86	0.78	0.82	63
accuracy			0.87	171
macro avg	0.87	0.85	0.86	171
weighted avg	0.87	0.87	0.87	171

Step 4.

Repeat with scaled data.

Use standard scaling, *i.e.*, scaling with Equation 10.2.1.

In [9]:
```
# scale the dataset first. Only the features, and not the outcome needs to be scaled.

from sklearn.preprocessing import StandardScalar

scaler = StandardScaler()

sX = X.copy()
sX = scaler.fit_transform(sX)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(sX, y, test_size=0.30, random_state=1)
```

Step 5.

Evaluate the accuracy of model with scaled data.

In [10]:

```
model = KNeighborsClassifier(n_neighbors=7)

# Train the model using the training sets.
model.fit(X_train, y_train)

# Getting predictions from the model
y_test_hat = model.predict(X_test)

cm = confusion_matrix(y_test, y_test_hat)
print(cm)
[ [102   6]
 [ 5  58] ]
```

In [12]:

```
1  from sklearn.metrics import classification_report
2
3  print(classification_report(y_test, y_test_hat))
```

	precision	recall	f1-score	support
B	0.95	0.94	0.95	108
M	0.91	0.92	0.91	63
accuracy			0.94	171
macro avg	0.93	0.93	93	171
weighted avg	0.94	0.94	0.94	171

Recall of the Malignant class has improved by almost 14% to 0.92 while precision has improved by 5% points. The overall accuracy (f1-score) has gone up from 87% to 94%. Comparing it with our Logistic Regression classifier, we see that the scaled KNN is even better than the Logistic Regression classifier, and clearly, scaling had a major impact on the model quality.

10.4 OTHER USES OF SCALING

We have already shown that when distance-based metrics are used for deciding the outcome, scaling is an important pre-processing step which has the potential to improve model quality.

All machine learning models have an algorithm for training or fitting the model to the data. The training step essentially figures out the model parameters such that the values predicted by the model are close to the observed values of the target variable, *i.e.*, minimizes model error. A famous algorithm which is used fo finding optimal parameter values for parametric models like logistic regression as well as the sophisticated neural network models (which we have not discussed) is called gradient descent. Gradient descent, as well as its variants, run much faster when the data is scaled. The reason behind this is beyond the scope of this book. Hence, whenever algorithms such as gradient descent are used, parameter scaling is an important pre-processing step. This is especially important with large datasets which can run into millions of examples.

Scaling in some cases also helps with stability of the numerical algorithms such as gradient descent.

Even if a particular case does not benefit from scaling, scaling will most probably not hurt. Hence, most machine learning pipelines include a scaling step.

Scaling may also help with data visualization. If two features have very different range of values, visualizing them on the same graph is difficult. Scaling them brings their ranges to comparable values, so it is easier to fit the points into the graph.

One notable exception where scaling does not make a difference is for ML algorithms which are based on decision trees which we will discuss in the next chapter.

CHAPTER 10 EXERCISES

Review Exercises

1. What is meant by scaling a variable? Why is it necessary?
2. Explain min-max scaling and its properties.
3. Explain standardization and its properties.
4. For a regression problem, do we need to scale the target variable?

Investigative Exercises

1. A dataset has a single feature and a target variable. We want to use KNN for building a predictive model. Do you expect that scaling the feature will have any effect on the results? If yes, why and if not, why not?
2. Repeat the KNN based modeling with scaling, but with min-max scaler instead of the standard scaler.

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#feature-scaling

11

Decision Trees

11.1 INTRODUCTION

A tree is a data structure which is very often used in computer algorithms and software to store data for sorting and searching purposes. Trees can also be used to implement decision systems. These very same trees can be used to implement regressors and classifiers as we will soon see.

For our discussion, we will specifically consider Binary Decision Trees. Binary decision trees can be characterized as follows:

1. They have internal nodes and leaf nodes.
2. Internal nodes implement a condition which evaluates to 'True' or 'False'.
3. Internal nodes have two subtrees, one corresponding to the condition evaluating to 'True', the other for the condition 'False'.
4. All nodes including lead nodes have a parent, except for one node which is called the root node.
5. Leaf nodes do not have any children and store a set of values.

A binary decision tree is shown in Figure 11.1.1. Let us say we have a sports league where players' salaries are decided by the tree below. So as per the tree, if the player has less than 4.5 years of experience, then he/she will be paid 5.0 Lakhs (1 Lakh = 100,000). If experience is more than 4.5 years, then we will also have to look at the average run scored by the player. If the average runs are less than 50, then the player's salary should be 7.0 Lakhs, else it should be 10 Lakhs.

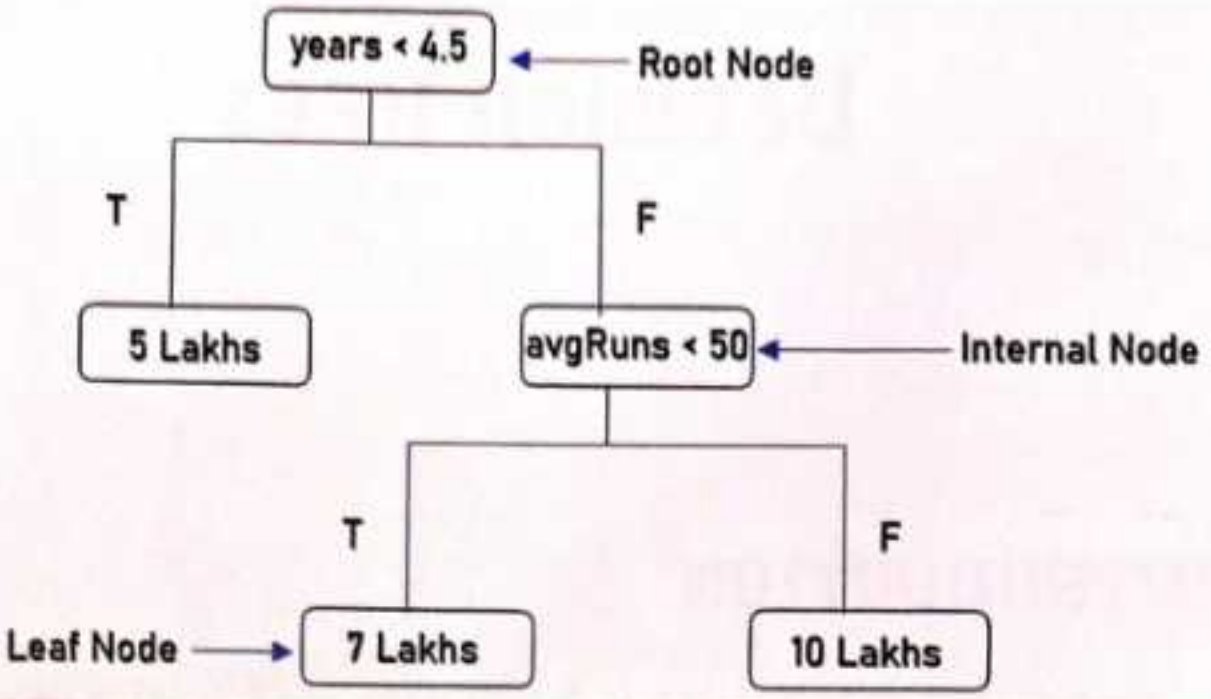


Figure 11.1.1 A decision tree to decide the compensation of a player

11.2 DECISION TREES AS CLASSIFIERS

A decision tree as illustrated before can be used as a regressor or classifier. For a regression or classification problem, we have a bunch of features and a corresponding target variable where the target variable takes on values which are dependent on the value of the features. If we use the features in the internal (or root nodes) to generate appropriate conditions and have the targets along with the target values in the leaf nodes, we have a regressor (or classifier if the target is a discrete variable). We will illustrate this idea next with a classification problem.

The dataset we will consider is the banknote authentication dataset available at the UCI repository.

(Source: https://archive.ics.uci.edu/ml/datasets/banknote+authentication).

	variance	skew	kurtosis	entropy	authentic
852	-4.886100	7.0542	-0.172520	-6.959000	1
1183	-3.574100	3.9440	-0.079120	-2.120300	1
543	-1.421700	11.6542	-0.057699	-7.102500	0
885	0.030219	-1.0512	1.402400	0.773690	1
1032	0.163580	-3.3584	1.374900	1.356900	1
262	1.811400	7.6067	-0.978800	-2.466800	0
158	2.263400	-4.4862	3.655800	-0.612510	0
663	3.779800	-3.3109	2.649100	0.066365	0
585	4.339800	-5.3036	3.880300	-0.704320	0
364	5.782300	5.5788	-2.408900	-0.056479	0

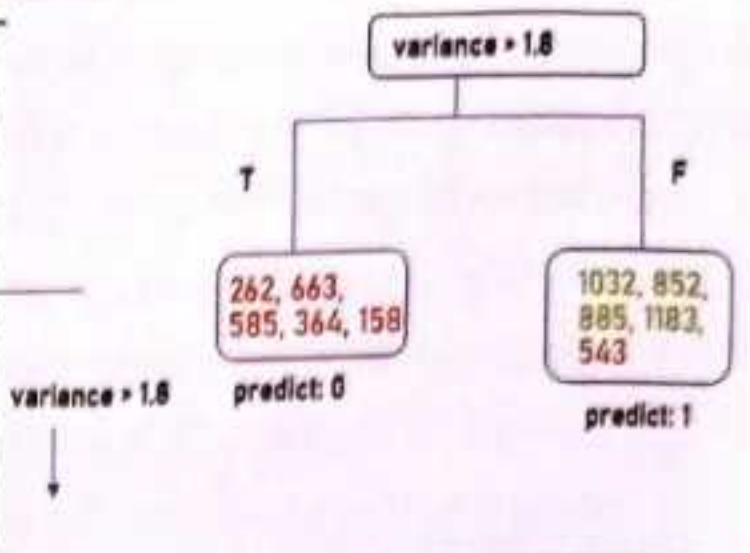


Figure 11.2.1 A partially built decision tree classifier for the data in the Table on the left

Images of 1372 banknotes, 762 of them forged and 610 authentic, were taken by an industrial camera. Four features of the images were extracted using an image processing technique called wavelet transform, the features being variance, skew, kurtosis, and entropy. These are various statistical measures of a distribution of data. We are aware of the variance metric, which is simply the square of the standard deviation. A curious reader can find out about the other measures from a book on Statistics but it is not required for the forthcoming discussion. Authentic notes have been labeled as '1' and forged notes as '0'. A sample of the dataset is shown in Figure 11.2.1. We have sorted the values according to their variance. For the examples which have been presented, if we use 'variance > 1.8' as a condition, then all those examples which satisfy the condition are forged (label = 0). Of those that do not satisfy the condition, all are authentic, except for one example (example id = 543). The decision tree in Figure 11.2.1. illustrates how the condition involving the one feature partitions the examples into different leaf nodes.

Let us assume we are satisfied with the simple decision tree created in Figure 11.2.1 as a basis for our classifier. The next step in using a decision tree as a classifier is to assign a label to each leaf node. Since all the examples in the leaf node on the left are of forged banknotes, we assign it a label of 0. For the leaf node on the right, since most of the examples are of authentic notes, we assign it a label of 1.

The simple decision tree in Figure 11.2.1 can be used to predict the class of all the examples used in building the tree (the training set) as well as of new examples (say, from the test set). This, in brief, is how decision tress are built and used for classification.

In general, a dataset will have many examples, and a simple decision tree based on a single feature will have many classification errors. In our simple example in Figure 11.2.1, the decision tree has one classification error. We next show how the decision tree can be extended to correctly classify the example on which it makes an error. We consider the set of examples which are in the right leaf node of the tree in Figure 11.2.1 in Figure 11.2.2 given here.

	variance	skew	kurtosis	entropy	authentic
1032	0.163560	-3.3584	1.374900	1.35690	1
885	0.030219	-1.0512	1.402400	0.77369	1
1183	-3.574100	3.9440	-0.079120	-2.12030	1
852	-4.886100	7.0542	-0.172520	-6.95900	1
543	-1.421700	11.6542	-0.057699	-7.10250	0

skew > 10 ↓

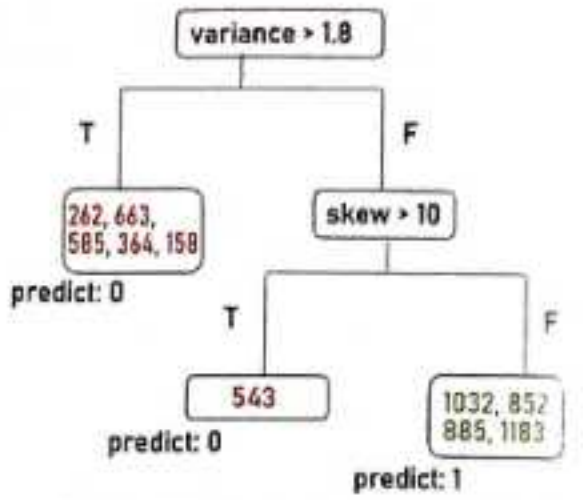


Figure 11.2.2 A decision tree, grown from the tree of Figure 11.2.1 to minimize the errors in the right node of the tree of Figure 11.2.1

We have sorted the above points based on the 'skew' feature. We now see that if we use the condition 'skew > 10', it will separate the example which corresponds to a forged note (id = 543, label = 0) from the rest which are authentic. Adding this condition to the tree of Figure 11.2.1 gives us the tree in Figure 11.2.2 which now perfectly classifies every example in our sample.

This shows how the multiple features in the dataset can be used to build a decision tree with reduced error.

It needs to be mentioned with regard to partitioning that a feature can appear in multiple nodes of the decision tree. However, the condition will be different if the feature appears on the same path (*e.g.*, skew > 10 or skew < −15). Further, the same condition (*e.g.*, kurtosis ≥ −0.1) can appear in multiple nodes as long as they are not on the same path in the tree.

It can be shown that decision trees can be used to perfectly classify any dataset if there is no limit on the tree depth and no constraints are placed on using the features in the conditionals of the internal nodes. This implies a 100% accuracy on the training set and is essentially a case of overfitting the data, a situation we had discussed in Chapter 5 which we want to avoid. We will discuss overfitting in decision trees as well as discuss how to avoid the overfitting in the next section.

NOTE

A decision tree classifier can generate a class probability during prediction, just like a logistic regression classifier. Consider an example for which the tree is used to predict the class it belongs to. The classifier will start from the root and use the feature values of the example to traverse to a leaf node. The leaf node will have multiple training examples associated with it, each associated with a given class. Let us say the leaf has 8 examples from class 0, and 2 from class 1. Then the probability that the new example belongs to class 0 or pr(Y=0) is 8/10 or 0.8. The probability that it belongs to class 1 or pr(Y=1) is 2/10 or 0.2. The label for the example will thus be declared as '0' since the probability pr(Y=0) is greater than 0.5.

11.3 OVERFITTING IN DECISION TREES

Let us first try to visualize how decision trees work in 2D, *i.e.*, when there are two features. In the example shown in Figure 11.3.1, the dataset has two features, x and y. Using the decision tree in the figure, all points whose y value is greater than y_1 (*i.e.*, for which $y > y_1$) corresponding to the green region R_2, is marked as R_2. All those points whose y value is less than y_1 and whose x value is less than x_1 corresponding to the blue region R_2 is also marked as R_2. But those whose y value is less than y_1 and whose x value is greater than x_1 corresponding to the yellow region R_1, is marked as R_1. Thus, the tree has split up the space into several rectangular regions. If the regions R_1 and R_2 corresponding to two classes where points in the region R_2 (both green or blue) belong to class with label '0' while the points in region R_1 belong to a class with label '1', then the decision tree becomes a classifier for our dataset.

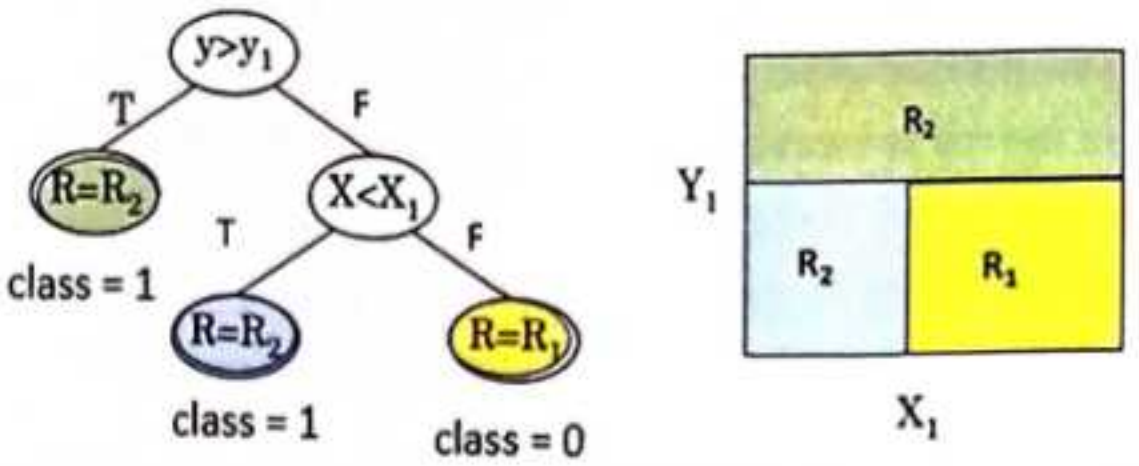


Figure 11.3.1 Illustration of the rectangular regions carved out in a two-dimensional space by a decision tree for two features

Classes for a dataset essentially define regions in the space defined by the features. In a complex case, the regions defined by a class may not be rectangular. However, the decision tree always works by approximating the non-rectangular regions with rectangular regions. Better the approximation, higher is the accuracy of the corresponding classifier. It can be shown that depending upon the conditions in the internal nodes; the complete region can be split into arbitrarily complex rectangular sub-regions, such that the error between the actual non-rectangular regions and the regions defined by the tree can be made infinitesimally small.

As another example, consider the set of points in Figure 11.3.2(*a*) defined by two variables or features, x_1 and x_2. The red points correspond to the *red* class, the blue points to a second class, the *blue* class. In Figure 11.3.2(*b*), we see how a logistic regression classifier would divide the 2D space. Since a logistic regression classifier essentially uses a straight line to divide the plane, its best effort is not very good, giving an accuracy of about 47%. The blue points in the red region and the red points in the blue region are misclassified by the classifier. In Figure 11.3.2(*c*), we show what a classifier based on neural networks does (It is a type of classifier we have not discussed but which is considered to be the most powerful classifier known as of this writing). It beautifully splits the region into 6 parts in a way which corresponds very well with the petals in the plot. The accuracy of this model is close to 90%. There are a few blue dots near the red petals and vice versa which get misclassified by this classifier. In Figure 11.3.2(*d*), we see the regions created by a variation of a decision tree (called random forest which we will discuss in a subsequent chapter). By creating fine-grained rectangular

regions to partition the place so that almost every point is classified correctly, it achieves an accuracy of 98%. However, the regions created look artificial and do not follow the pattern which the petals seem to indicate. This is a case of extreme overfitting exhibited by decision tree models.

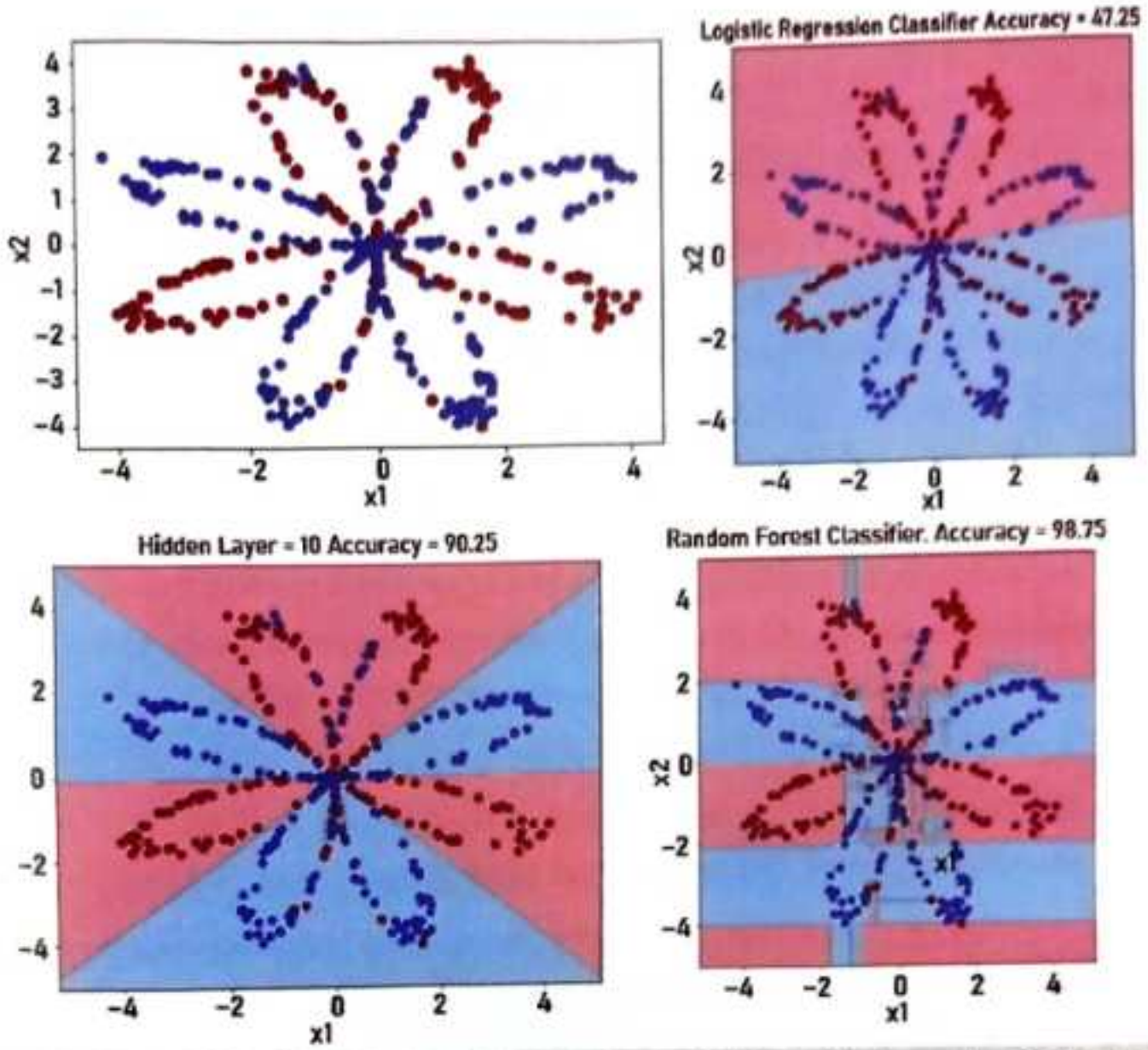


Figure 11.3.2 (*a*) Points belonging to two classes, indicated by the color of the point (*b*) the regions generated by a logistic regression classifier (*c*) regions generated by a neural network classifier (*d*) regions generated by a decision tree classifier

In the next section, we will see how such overfitting can be avoided.

11.4 PREVENTING DECISION TREES FROM OVERFITTING

We will discuss some ways in which decision trees can be prevented from overfitting. Before that, we want to mention that decision trees are almost

never directly used as classifiers in practice, but powerful classifiers based on decision trees such as random forest classifier discussed in the next chapter are frequently used. However, the ways to reduce overfitting in decision trees is equally applicable to derived classifiers such as random forests.

There are several ways of preventing overfitting in decision trees. We explain two heuristics below.

1. Limiting tree depth

We have seen in Figure 11.2.2, that a tree of depth two provides a perfect classification while a tree of depth 1 in Figure 11.2.1 has an error for one of the points. In general, forcing a limit on the depth of the tree produces a model which has more training error, but which generalizes better, thus reducing overfitting.

2. Enforcing a minimum on the size of the leaf samples

Another way of reducing overfitting is to specify a minimum number of samples for the leaves of the tree. If the minimum sample size was specified as 3, the tree of Figure 11.2.2 would be disallowed. The tree in Figure 11.2.1 would still be a candidate. This again has the effect of creating a tree which produces more error on the training set since it can't produce perfect regions due to the constraint. But as shown above, it generalizes better and reduces overfitting.

Both the options of minimum leaf sample size and limiting tree depth limits how small the regions created by a decision tree can be. This tends to prevent the tree from fitting to noise or creating experimental errors in the data.

Even when constraints are enforced on decision trees to prevent overfitting, decision trees tend to have a large variance. If there is a large dataset for training and two large subsets are derived from it and two different trees are created from each subset, the two trees can be substantially different. To address this issue, instead of using a single decision tree, Ensemble models which use multiple decision trees are used in practice. We will discuss ensembles and an ensemble model based on decision trees called Random Forests in the following chapters.

CHAPTER 11 EXERCISES

Review Exercises

1. Explain how a decision tree functions. Assume a binary decision tree.
2. We have a classification dataset with features and a target variable. A decision tree is being used as the classifier. What will the internal nodes of the tree have? What will the leaf nodes contain?
3. A new point (characterized by its features) is given. A decision tree exists for classifying points of this type. How will the tree be used to classify the point? How will the class for the point be generated? How will the probability that the point belongs to this class be determined?
4. Decision trees tend to break up the space into rectangular regions. Explain this statement by using a dataset with two features.
5. What are some of the ways in which overfitting in decision trees can be reduced?

Investigative Exercises

1. A decision tree can be used for regression also. When a new example is provided, how will the decision tree be used to calculate the value of the target variable for the example?
2. Can limiting the number of leaf nodes reduce overfitting in a decision tree? If so, illustrate with a simple example.

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#decision-trees

12

Ensemble Models

12.1 INTRODUCTION

Let us say we have three different predictors, P_1, P_2, and P_3 for the same task. All three, given information about a situation, predict the class to be 0 or 1. We can then create one optimal predictive model P which combines the best of the three predictors P_1, P_2, and P_3. The optimal predictor P predicts the class to be 1 if two or more of the base predictors predict the class to be 1. And similarly, P predicts the class to be 0 if two or more of the base predictors predict the class to be 0. This is shown in Figure 12.1.1 below.

	P1	P2	P3	P
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

Figure 12.1.1 Three independent predictors (P_1, P_2, P_3) and a predictor (P) based on an ensemble of the three

Now let us assume that each of the predictors have prediction accuracy of more than 50%. We will assume that the accuracy of all of them is 70%, though this assumption is not necessary for the discussion to follow. We can then show that the predictor P is more accurate than either of the three individual predictors. To understand why that is so, refer to Figure 12.1.2.

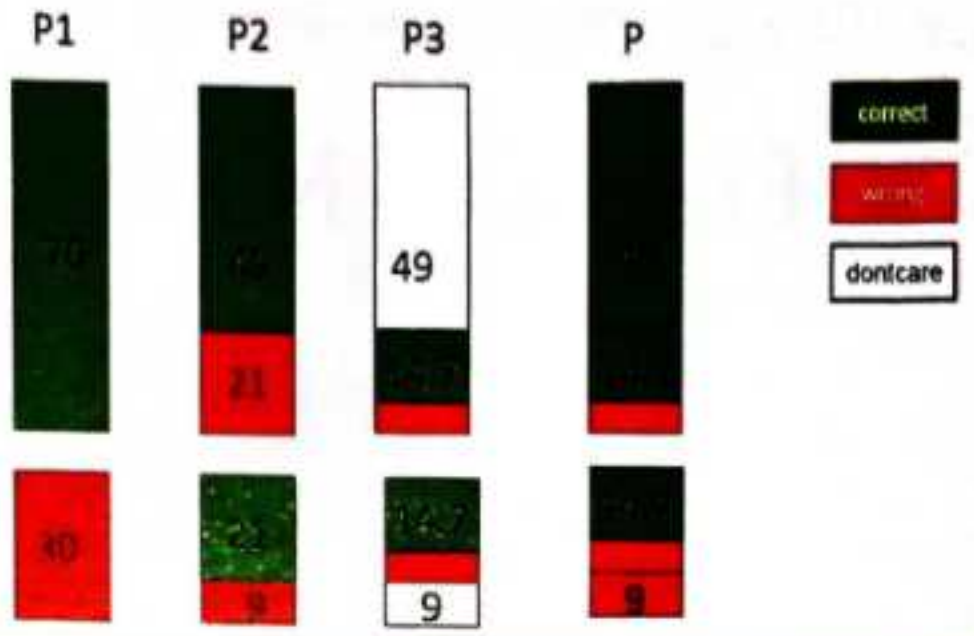


Figure 12.1.2 Illustration of a case where an Ensemble gives higher quality predictions than the base models

Suppose there are 100 examples. P_1 predicts 70 of them correctly and 30 of them incorrectly, as shown in Figure 12.1.2 Of the 70 that P_1 predicts correctly, P_2 will predict 70% of them correctly, that is of these 70, P_2 will predict 49 (0.7 * 70) correctly. These 49 will be thus predicted correctly by P also, regardless of the prediction made by P_3 on these 49 cases.

Of the 70 predicted correctly by P_1, P_2 will predict 30% of them, or 21 of them incorrectly. This is again shown in Figure 12.1.2. But of these 21, P_3 will predict 70% of them, or 14.7(0.7 * 21) correctly. Since P_1 and P_3 agree on these 14.7 examples, and they are correct, P will be correct on these 14.7.

Next consider the 30 examples predicted incorrectly by P_1. P_2 will predict 70% of them or 21(0.7 * 30) correctly. Out of these 21, P_3 will further predict a 70% of them or 14.7(0.7 * 21) correctly. Since on these 14.7, both P_2 and P_3 agree, and are correct, P will predict these correctly.

P will thus predict 78.4 (= 49 + 14.7 + 14.7) of the cases correctly. The accuracy of P is thus greater than 78%, which is higher than the accuracy of the individual predictors which is 70%.

The above argument shows how a set of predictors with accuracy greater than 50% can be used as an ensemble to build a predictor with higher accuracy than each individual predictor.

Another argument favoring using an ensemble is as follows. It can be shown that if there are *n* independent distributions from the sample population and each has a variance of σ^2, then the variance of the mean of the distributions is given by σ^2/n. Hence, averaging of an ensemble can be used to reduce the variance of a model which essentially generates a model with less variance, *i.e.*, which is less likely to overfit.

However, there is the caveat that the models used in the ensemble should not be correlated, or if they are, the correlation should be low. As an example, consider the extreme case of correlation, where we use three base classifiers and each of them predict with 70% of accuracy, but when any of them predicts an example correctly, the remaining two also predicts correctly. Similarly, when one predicts incorrectly, the remaining two also predict incorrectly. This is the worst scenario, where all three predictions are perfectly correlated. In this case, it should be obvious that the ensemble will have the same accuracy, 70%, of either of the base classifiers.

The base models from which an ensemble is built need not all be of the same type. For example, we can build an ensemble from a logistic regression classifier, a KNN classifier, and a decision tree classifier. In the next chapter, we will discuss an ensemble model built entirely from decision trees called a Random Forest Classifier.

CHAPTER 12 EXERCISES

Review Exercises

1. What is an ensemble model? When does it make sense to use an ensemble?
2. There are three models. The predictions of the models correlate most of the time. In this case, does it make sense to create an ensemble? Explain.

Investigative Exercises

Supporting Material

13

Random Forest Classifier

13.1 INTRODUCTION

A Random Forest Classifier is an ensemble of many decision trees. The number of decision trees to have in a random forest is a hyperparameter which needs to be tuned for the dataset. The idea is that an ensemble of decision trees will have lower variance and higher accuracy than a single decision tree. We have seen the explanation of this phenomenon in Chapter 12.

To build a random forest, we need decision trees for the same overall dataset, but the trees should not be correlated, as explained in Chapter 12 or should have low correlation. In practice, it is difficult to ensure that the base trees have zero correlation. Heuristics discussed below are used to minimize the correlation between decision trees used in a random forest ensemble model.

Let us say that we have decided to build a random forest comprising of N decision trees where 'N' is usually a large number and can be set to 100 or more for large datasets. To build each of the 'N' decision trees, the training dataset is sampled using random sampling techniques 'N' times. If the training dataset has 10000 examples, each of the 100 decision tress might be built from say 6000 examples which are sampled from the set of 10000. This ensures that the dataset

used for building each tree is different (though there will be some overlap), thus hopefully leading to 'N' decision trees which are not significantly correlated.

To further reduce the correlation, random forests limit the number of features that can be used for each decision node to be a subset of the features. If there are 10 features, maybe 4 features are used for each decision node in the first level of the tree. A different set of 4 features may be used for the second level, and so on. In the second tree of the ensemble, the same strategy is used, but the features used for the first level could be different from the features used in the first level in the first tree. Thus, different random subsets of the features are used at different levels of the same tree as well as different trees. This has a strong effect on de-correlating the trees.

Along with constraining the minimum leaf size and tree depth as discussed in Section 11.4 which reduces overfitting, random forests have proved to be powerful predictors both for classification as well as regression.

We will now demonstrate usage of random forests in the Hands-on Exercises below.

13.2 HANDS-ON EXERCISE: RANDOM FORESTS FOR BANK NOTE CLASSIFICATION

Step 1.

Read and explore the data.

In [1]:
```
# Importing required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Loading the dataset into pandas dataframe.
df = pd.read_csv('data_banknote_authentication.csv')
```

In [3]:
```
# print all the available features.
df.columns
```

Out[3]:
```
Index(['variance', 'skew', 'kurtosis', 'entropy', 'authentic'], dtype='object')
```

```
In [4]: # Check for nulls.
        df.columns[df.isnull().any()]
Out[4]: Index([], dtype='object')
```

```
In [5]: # Count the number of malignants and benigns in the dataset.
        df['authentic'].value_counts()
Out[5]: 0   762
        1   610
        Name: authentic, dtype: int64
```

```
In [6]: df.sample(n=5, random_state=55).sort_values(['variance'])
```

Out[6]:

	variance	skew	kurtosis	entropy	authentic
1180	-2.21830	-1.2540	2.9986	0.36378	1
769	-0.89409	3.1991	-1.8219	-2.94520	1
1353	0.11592	3.2219	-3.4302	-2.84570	1
239	2.39520	9.5083	-3.1783	-3.00860	0
722	4.84510	8.1116	-2.9512	-1.47240	0

Step 2.

Build the Model using Random Forest.

Default hyperparameters are used except for the leaf sample size.

```
In [7]: # Load the features to a variable X
        # X is created by simply dropping the diagnosis column and retaining all others
        X = df.drop('authentic', axis = 1)

        # Load the target variable to y
        y = df['authentic']
```

```
In [8]: from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)
```

```
In [9]: # Let's create an instance for the LogisticRegression model and then train it with the training set.
        from sklearn.ensemble import RandomForestClassifier
        Classifier = RandomForestClassifier(random_state=0, min_samples_leaf=10)
        Classifier.fit(X_train, y_train)
```

Out[9]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=10, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                       oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Step 3.

Evaluate the Model.

Evaluation is done both on the training and test dataset to check for overfitting.

In [10]:
```
# Getting predictions from the model
y_test_hat = Classifier.predict(X_test)

# Compare the predicted values with the actuals.
Results = pd.DataFrame({'Actual': y_test})
column = pd.DataFrame({'Predictions': y_test_hat})
Results = Results.join(column.set_index(Results.index))
Results.head(5)
```

Out[10]:

	Actual	Predictions
1240	1	1
703	0	0
821	1	1
1081	1	1
37	0	0

In [11]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_test_hat))
```
0.9830097087378641

In [12]:
```
# Get the predictions from the model for the training set.
y_train_hat = Classifier.predict(X_train)
print(accuracy_score(y_train, y_train_hat))
```
0.9875

The accuracy of the training set and the test set are both very high. Hence, we conclude that our model is not overfitting and that it is simply a very good model.

Step 4.

Check the Confusion Matrix.

```
In [13]: ▸ from sklearn.metrics import confusion_matrix, recall_score, precision_score

         cm = confusion_matrix(y_test, y_test_hat)
         print(cm)
         [[229   6]
          [  1 176]]

In [14]: ▸ # Assigning Variables for convinience
         TN = cm[0][0]
         FP = cm[0][1]
         FN = cm[1][0]
         TP = cm[1][1]

         recall = TP / float(FN + TP)
         print("recall:", recall)

         precision = TP / float(TP + FP)
         print("precision:", precision)

         specificity = TN / (TN + FP)
         print("specificity:", specificity)
         recall: 0.9943502824858758
         precision: 0.967032967032967
         specificity: 0.9744680851063829
```

Step 5.

Find out Feature Importance.

The random forest implementation in Sklearn produces data on the importance of the features used to build the classifier. This can be very useful in getting insights into the relationship of the features of the target variable and can have applications in real life. The code below prints the features in their order of importance.

```
In [15]: ▸ feature_importances = pd.DataFrame(Classifier.feature_importances_,
                                              index = X_train.columns,
                                              columns=['importance']).sort_values('importance',

           feature_importance
```

Out[15]:

	Importance
Variance	0.623634
skew	0.202934
kurtosis	0.133198
entropy	0.040235

13.3 HANDS-ON EXERCISE: CONTROLLING OVERFITTING IN RANDOM FORESTS

In general, random forests by their very nature of being an ensemble are fairly good at not overfitting to the data. However, if we are not careful, models based on random forests can overfit. We will use the planar dataset we had used in Section 11.3 to show an instance of random forests overfitting and ways to reduce the overfitting.

We had seen how the random forest classifier had overfit the data in Section 11.3, creating multiple small regions in the 2D space resulting in a very high accuracy on the training set. We will see how controlling the depth, leaf size, as well as number of decision trees used (ensembling) has an effect on reducing overfitting.

Step 1.

Load the Libraries.

The Sklearn library contains some datasets which we will use for this exercise. The planar_utils library contains some helper functions for using the Sklearn datasets, for example load_planar_datasets.

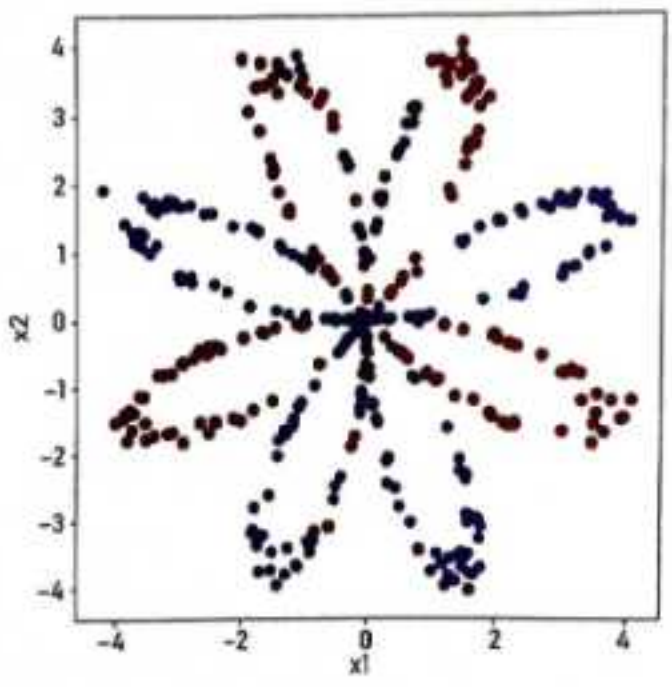
In [1]:
```
# Package imports
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import sklearn.datasets
import sklearn.linear_model
from planar_utils import *
```

```
%matplotlib inline

np.random.seed(1)
# set a seed so that the results are consistent

import warnings
warnings.simplefilter('ignore')
```

Step 2.

Load and Plot the Data.

```
In [2]: X, Y = load_planar_dataset()

plt.figure(figsize=(6, 6))

# Visualize the data
plt.scatter(X[0, :], X[1, :], c=Y.ravel(), s=40, cmap=plt.cm.Spectral);
plt.ylabel('x2')
plt.xlabel('x1')
plt.show()
```



There are two classes, shown by the red and blue colored dots.

Step 3.

Create a Test Set.

To create the test set, we have used the points in the training set with some random noise added. The labels are kept unchanged. The idea is to keep the test data set like the training dataset. In the earlier exercise, we had no such separate test set.

```
In [3]: X_test = X + np.random.rand(2,X.shape[1])/4
        Y_test = Y
```

Step 4.

Instantiate and Train a Random Forest Classifier.

```
In [4]: from sklearn.ensemble import RandomForestClassifier

        rclf = RandomForestClassifier(random_state=0, n_estimators=4)
        rclf.fit(X.T, Y.T)
```

```
Out[4]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_
                    depth=None, max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=4, n_jobs=None,
                    oob_score=False, random_state=0, verbose=0, warm_start=False)
```

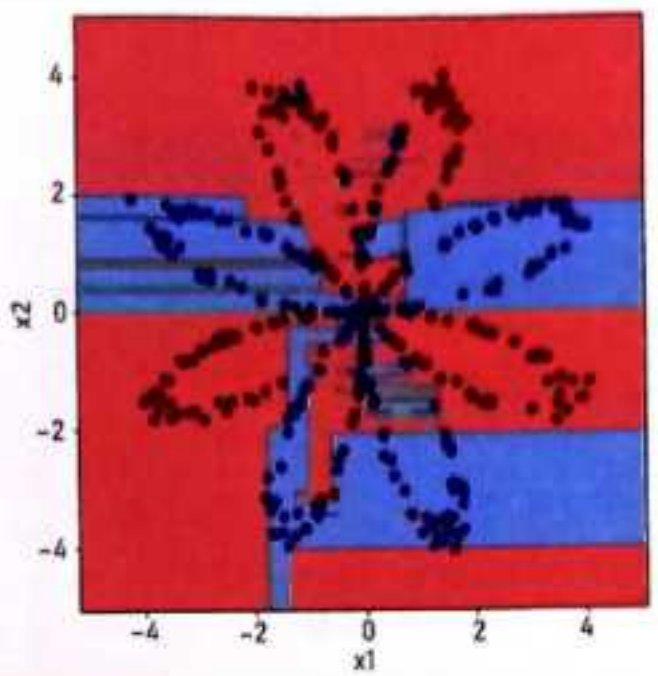
Note that the only hyperparameter specified when instantiating the model is 'n_estimators' which determines the number of trees used by the classifier to reduce overfitting. We have purposely kept this number to a low value to produce overfitting. All other hyperparameters have been kept at their default value.

Plot the Contours of the Regions Generated by the Classifier.

With the many unnaturally created regions, the overfitting should be obvious from the visualization.

```
In [5]: # Plot the contours of the regions created by the classifier.

        plt.figure(figsize=(6, 6))

        # Set min and max values and give it some padding
        x_min, x_max = X[0, :].min() - 1, X[0, :].max() + 1
        y_min, y_max = X[1, :].min() - 1, X[1, :].max() + 1
        h = 0.01

        # Generate a grid of points with distance h between them
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

        # Predict the function value for the whole grid
        Z = rclf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
```

```
# Plot the contour and training examples
plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral,alpha=.5)
plt.ylabel('x2')
plt.xlabel('x1')
plt.scatter(x[0, :], X[1, :], c=Y.ravel(), cmap=plt.cm.Spectral)
plt.show()
```



Step 6.

Evaluate Accuracy on the Training Set and Test Set.

This is a non-visual way of checking for overfitting as explained in Chapter 5. An extremely high accuracy on the training set and a significantly lower accuracy on the test set is a clear sign of overfitting.

```
In [5]: predictions = rclf.predict(X.T)
        s1 = np.dot(Y.ravel(),predictions)
        s2 = np.dot(1-Y.ravel(), 1-predictions)
        tot = float(s1) + float(s2)
        accuracy = tot/float(Y.size)*100
        print('Random Forest Classifier. '+' Training Accuracy ='+str(accuracy))
        Random Forest Classifier. Training Accuracy =94.25
```

```
In [7]: predictions = rclf.predict(X_test.T)
        s1 = np.dot(Y_test.ravel(),predictions)
        s2 = np.dot(1-Y_test.ravel(), 1-predictions)
        tot = float(s1) + float(s2)
        accuracy = tot/float(Y_test.size)*100
        print('Random Forest Classifier. '+' Test Accuracy ='+str(accuracy))
        Random Forest Classifier. Test Accuracy =79.0
```

Step 7.

Build a New Model to Avoid Overfitting.

The depth of the trees has been limited to 6 and the minimum samples for a leaf node has been set to 2. The number of decision trees to be used has been increased to 8. Note that these values in the first model were *no_limits*, 1, and 4 respectively.

```
In [8]: ▸ rclf = RandomForestClassifier(random_state=0,
                                       max_depth=6,
                                       min_samples_leaf=2,
                                       min_samples_split=2,
                                       n_estimator=8)

          rclf.fit(X.T, Y.T)

Out[8]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_
                    depth=6, max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=2, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=8, n_jobs=None,
                    oob_score=False, random_state=0, verbose=0, warm_start=False)
```

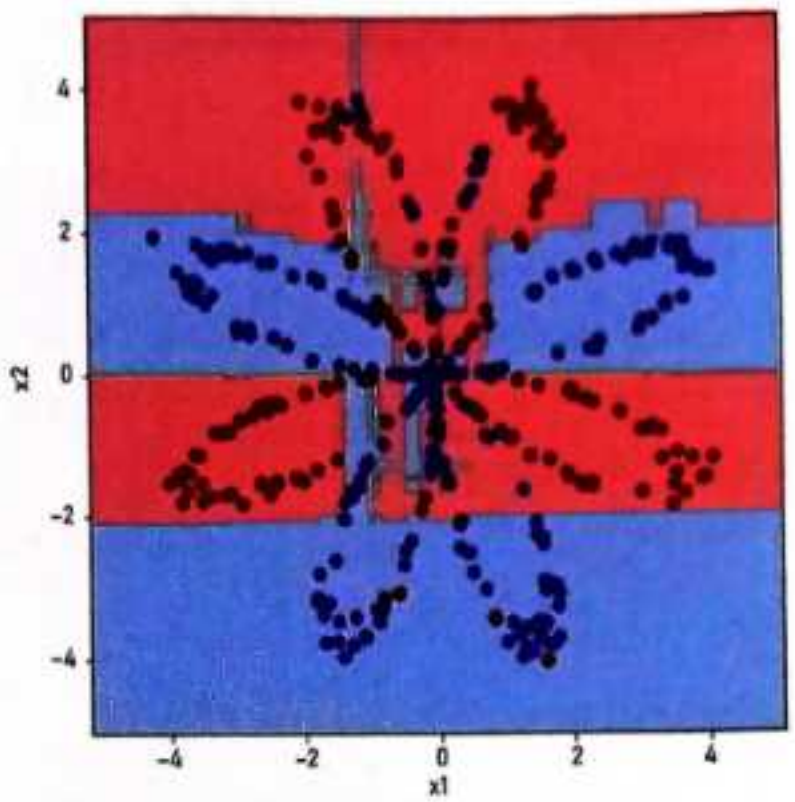
Step 8.

Plot the Contours of the Regions Generated by the Classifier.

This model also shows some overfitting as is evident in some small regions into which the space has been divided, but the overfitting is significantly less than that for the default model.

```
In [9]: ▸ # Plot the contours of the regions created by the classifier.

          plt.figure(figsize=(6, 6))

          # Set min and max values and give it some padding
          x_min, x_max = X[0, :].min() - 1, X[0, :].max() + 1
          y_min, y_max = X[1, :].min() - 1, X[1, :].max() + 1
          h = 0.01

          # Generate a grid of points with distance h between them
          xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

```
# Predict the function value for the whole grid
Z = rclf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot the contour and training examples
plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral,alpha=.5)
plt.ylabel('x2')
plt.xlabel('x1')
plt.scatter(x[0, :], X[1, :], c=Y.ravel(), cmap=plt.cm.Spectral)
plt.show()
```



Step 9.

Evaluate Accuracy on the Training Set and Test Set.

Training set accuracy has reduced, but test set accuracy has improved. Even in this model, the higher training accuracy also reflects that the model is still overfitting, though less so.

```
In [16]: predictions = rclf.predict(X.T)
         s1 = np.dot(Y.ravel(),predictions)
         s2 = np.dot(1-Y.ravel(), 1-predictions)
         tot = float(s1) + float(s2)
         accuracy = tot/float(Y.size)*100
         print('Random Forest Classifier. '+' Training Accuracy ='+str(accuracy))
         Random Forest Classifier. Training Accuracy =89.25
```

```
In [11]: ▸ predictions = rclf.predict(X_test.T)
          s1 = np.dot(Y_test.ravel(),predictions)
          s2 = np.dot(1-Y_test.ravel(), 1-predictions)
          tot = float(s1) + float(s2)
          accuracy = tot/float(Y_test.size)*100
          print('Random Forest Classifier. '+' Test Accuracy ='+str(accuracy))
          Random Forest Classifier. Test Accuracy =84.0
```

CHAPTER 13 EXERCISES

Review Exercises

1. A random forest is an ensemble of decision trees. For an ensemble to be effective, the individual models should not be correlated. How is it ensured that the decision trees in a random forest classifier are not correlated?
2. What are the advantages of a random forest classifier over a decision tree?
3. How is feature importance derived in a random forest model?

Investigative Exercises

1. Try various hyperparameter values to understand their effect on the model accuracy. There is a Sklearn function GridSearchCV, which automates the parameter tuning process. Try using that function to get the best possible values for the model accuracy. An alternative is to try RandomizedSearchCV, also available in Sklearn.

14

Unsupervised Learning

Regression and Classification problems belong to a class of problems called **Supervised** learning problems. In both regression and classification, the historical or training data provided to build a predictive model includes data on the variable to be predicted, *i.e.*, the outcome or target variable. When data is provided on the variable to be predicted, then the problem is a **Supervised** learning problem.

Unsupervised learning problems are those where the dataset does not have any target or outcome variable.

If there is no data on what needs to be predicted, then what can such algorithms do?

Suppose a customer has selected a few products to purchase. We would like to provide suggestions about other products that he should or could be buying. The data we have is historical data of what customers have bought. In the historical data, if we find repeat patterns of milk being bought with bread, and if the customer who is currently shopping has bought milk, then our algorithm can suggest buying bread. We cannot say that if milk is the value of a feature, then bread is the value to be predicted because vice versa, *i.e.*, if the customer has bought bread, he could buy milk. So, the role of the feature and predictor quickly gets changed. Rather, in this case, we have patterns in our historical data, and we

define rules saying that if certain patterns occur very often, then they can be used for making certain recommendations. This problem formulation goes by the name of Association Rule Mining and is an unsupervised learning problem. Even though we do not discuss association rule mining in this book, it would be apt to mention that the rules in association rule mining are automatically inferred from the data (*i.e.*, machine-learned) and are not created manually by human beings.

In unsupervised learning, the models still perform predictions. But there is no prior data on the outcomes. We need to define the outcome or prediction we desire and accordingly come up with an algorithm for doing so.

In the next section, we are going to discuss a form of unsupervised learning called Clustering and show how it can be used for say, targeted marketing campaigns. Following that, we are going to study a simple recommender system which is built using another unsupervised technique called Collaborative Filtering.

14.1 CLUSTERING

Suppose we have images of different kinds of animals. And we have an algorithm which groups the images into multiple groups, such that images in the first group correspond to cows, images in the second group corresponds to dogs, and so on. Our algorithm does not know that the first group corresponds to cows; it merely believes that the images in the group are highly related. There are three points to be noted. The dataset is grouped or clustered into multiple groups. Second, even though arbitrary clusters can be created, our algorithm has created clusters which are meaningful. Third, the algorithm did the clustering without being provided any hint on what the clusters could be—so it must be an unsupervised learning algorithm.

We will discuss a clustering algorithm in the next section called K-means. In the Hands-on Exercise, we will provide a motivational example to show how clustering can be used in a practical application of customer segmentation.

14.2 CLUSTERING WITH K-MEANS

The K in K-means stands for the number of clusters to be created. K-means does not know how many clusters to create. It must be provided the number of clusters.

K, to create. Given the value of K, K-means creates K clusters which satisfy some tightness criteria. The intuition is, points which are close to each other, or are tightly packed, should belong to a cluster. And these clusters should then have some meaning associated with them which can be useful in applications.

We will discuss the tightness criteria used by K-means, the algorithm itself, and how the clusters created can be used meaningfully.

As an illustration of the tightness of clusters, consider the two different clustering formats of a set of points in two dimensions in this section. Visually, if we had to create two clusters where the criteria for cluster formation is that points in a cluster should be close to each other, then arguably, the second clustering would be considered better than the first cluster.

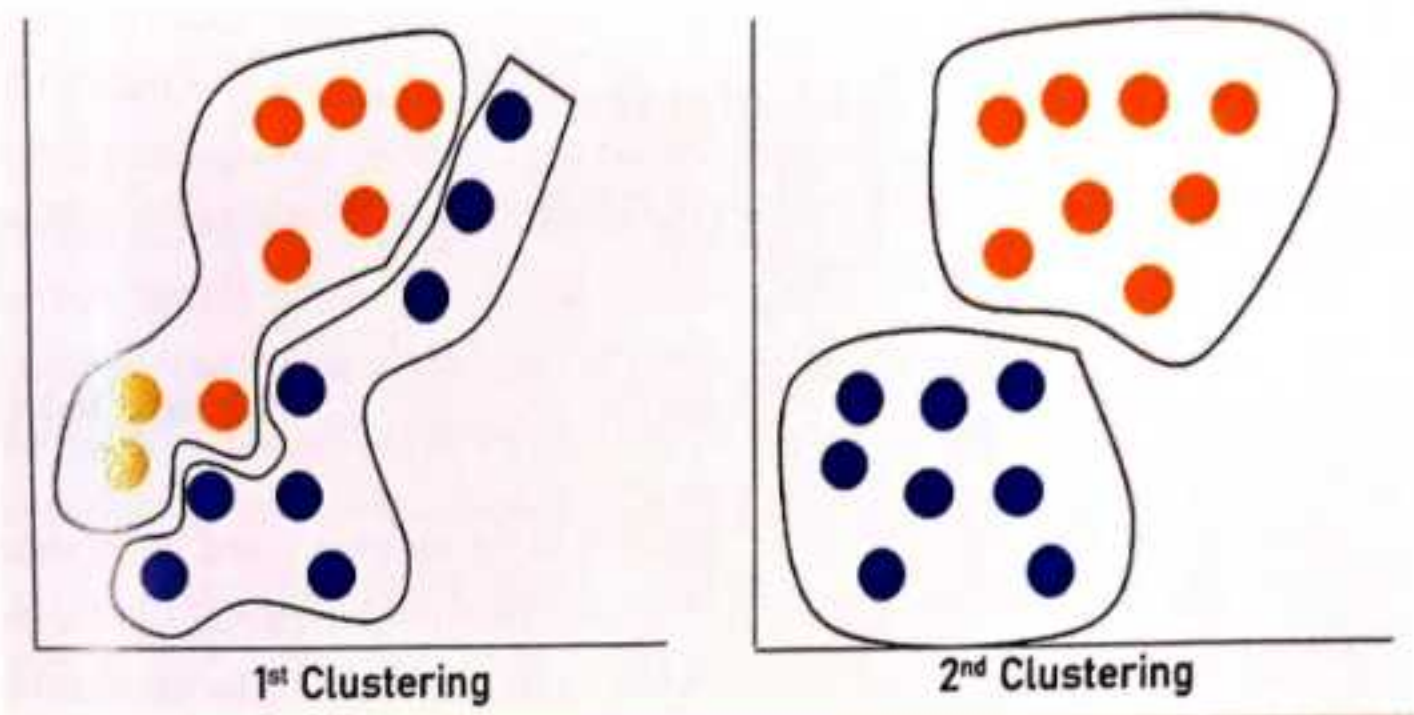


Figure 14.2.1 Two different clusters for the same set of points

K-means uses a clustering criterion to determine the most optimal clustering of a dataset. **K-means itself does not have any way of knowing if the clusters are significant or not**. It is for the human interpreter to determine if there is some significance to the clusters generated by K-means as will be illustrated with a Hands-on Exercise. But before that, we will discuss some preliminaries which are required to understand the K-means algorithm, and then describe the algorithm itself.

14.2.1 Preliminaries to Understand the K-Means Algorithm

1. Centroid

The term centroid is used to define a central point in a triangle. A triangle in two-dimensional space is defined by three points (x_1, y_1), (x_2, y_2), and (x_3, y_3). The centroid of the triangle (x, y) is defined as given below:

$$C(x, y) = \left(\frac{x_1 + x_2 + x_3}{2}, \frac{y_1 + y_2 + y_3}{2} \right) \qquad \textit{Equation } \mathbf{14.2.1}$$

The same equation of centroid of a triangle can be extended to when there are more than three points. Given K clusters, we will denote the centroid of the k^{th} cluster C_k by (x^k, y^k).

2. Within-Cluster Sum of Squares (WCSS)

We define a term called the Within-Cluster Sum of Squares WCSS for a cluster C_k below:

$$WCSS(C_k) = \sum_{i \in C_k} (x_i - x^k)^2 + (y_i - y^k)^2 \qquad \textit{Equation } \mathbf{14.2.2}$$

where (x^k, y^k) is the centroid of the points in the cluster C_k, and $|C_k|$ is the number of points in the cluster. WCSS is essentially the sum of the square of the distances between the points in the cluster and its centroid. It should be obvious that if the points are tightly clustered, then WCSS will be a small number and vice versa.

3. Inertia

We define inertia simply as the sum of the WCCS values for a group of clusters. If there are K clusters, then inertia of the clusters is given by:

$$inertia = \sum_{k=1}^{K} WCSS(C_k) \qquad \textit{Equation } \mathbf{14.2.3}$$

Given K, the number of clusters, and n points, there are many ways to cluster the n points into K clusters. The K-means algorithm creates K clusters such that the sum of the WCSS values for the K clusters is minimal, *i.e.*, it tries to create K clusters such that the inertia of the K clusters created is less than the inertia of

clustering of the *n* points into some other *K* clusters. It should be mentioned at the onset that K-means is based on heuristics and does not guarantee a clustering with minimum inertia. It merely finds a clustering with low inertia.

4. Outline of the K-means algorithm

K-means algorithm is based on heuristics which reduces the inertia value by redistributing the points amongst the clusters iteratively.

1. Randomly pick *K* points from the sample points as initial cluster centers.
2. Assign each point to the nearest of the *K* points. This creates *K* clusters.
3. Repeat until the change in inertia is less than the tolerance value.
4. Recompute the centroid of each of the *K* clusters.
5. Reassign each point to the nearest centroid.

After step 2, *K* clusters are created. The centroids calculated in Step 4 will be different in general from the initial *K* points selected as cluster centers. These centroids are now designated as the new cluster centers. Hence, Step 5 will lead to a reassignment of the points to the new cluster centers, *i.e.*, the clustering will change. This reassignment of the points will reduce the inertia iteratively until the new centroid remains very close to the centroid from the previous iteration, in which case the reduction in inertia will be less than the tolerance value and the algorithm will stop.

Even though it can be proved that K-means leads to a reduction in the inertia, K-means due to its heuristic nature will usually not find the clustering which has minimum inertia, *i.e.*, it finds a local optimum but not a global optimum. It should also be noted that the final clustering formed by K-means depends upon the initial set of *K* points selected as the cluster centers. To reduce the effect of the initial selection on the clustering, the K-means algorithm as discussed earlier can be run *M* times, each time with a different set of *K* initial points. The clustering with the lowest inertia from the *M* runs can be used as the final clustering. There are implementations of K-means which works as described.

14.2.2 The Elbow Method for Selecting K

We have mentioned that the number of clusters of *K* needs to be specified to the K-means algorithm for it to generate the clusters. It should be obvious that if

there are N points, and we specify $K = N$, then each point will be its own cluster and inertia will be 0. Is this then the most desirable clustering? The idea behind clustering is to find similarity amongst samples and group them so that something meaningful can be done with each group. For example, if we want to do customer segmentation so that a business can create marketing strategies by segment, having too many clusters is not helpful. How many marketing strategies can one have? Having each point in its own cluster creates too many clusters and the notion of finding commonality in the points belonging to a cluster does not make sense. Hence, some intermediate value of K which is a small number is usually desirable. The definition of what is a good value of K is somewhat subjective. The Elbow Method can be used to provide some objectivity to the selection of a reasonable K.

Below, we show a plot of K versus inertia for a given dataset.

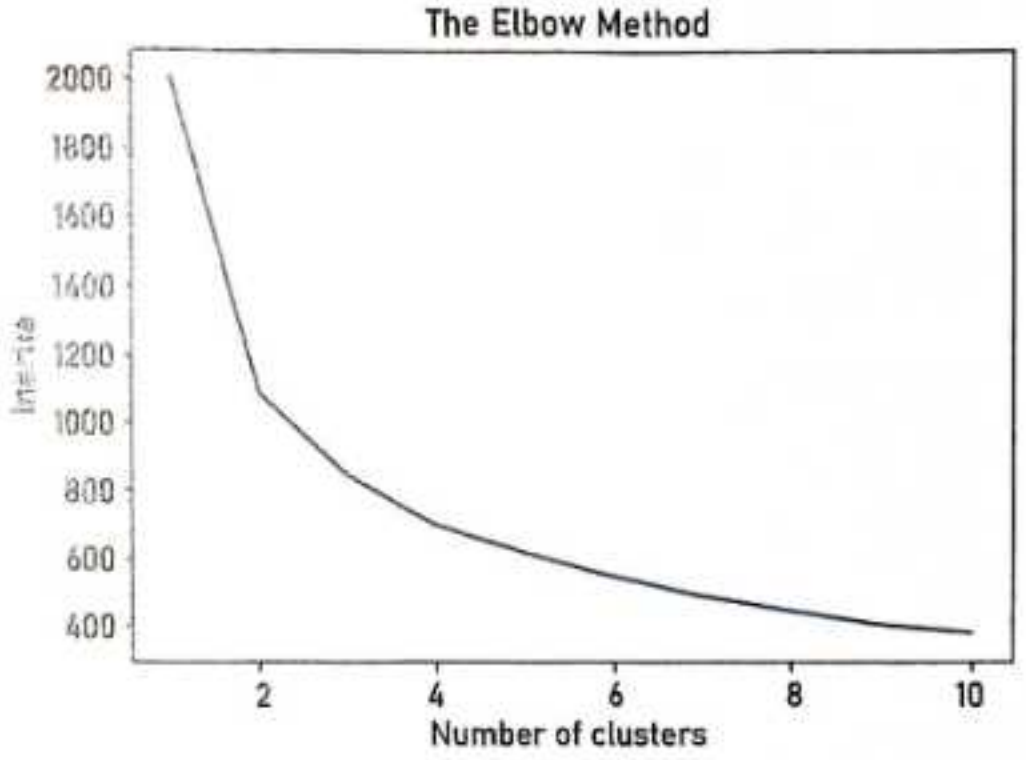


Figure 14.2.2 Illustration of the Elbow Method which is a plot of the inertia (Eq 14.2.3) versus number of clusters

We can see that the inertia drops steeply when the number of clusters is increased from 1 to 2. From 2 to 4, the drop is less steep, while from 4 onwards, the steepness of the drop reduces even further. Wherever there is a significant reduction in the steepness, an elbow is formed. The elbow points are usually a good choice for a value of K. From the above plot, the most significant elbow formation seems to be for $K = 2$. The next significant elbow seems to be at $K = 4$. The user might

consider a case where $K = 2, 3$, and 4, and verify which produces more meaningful clustering. What is meaningful is clearly subjective.

To summarize, to use K-means, we need to specify the number of clusters K to the algorithm. However, we do not know apriori a good value for K. So, we use K-means itself with different values of K and then use the Elbow Method to determine a suitable value of K. We will illustrate this with a Hands-on Exercise below.

14.3 HANDS-ON EXERCISE: CUSTOMER SEGMENTATION USING K-MEANS

We will apply K-means to a dataset of student performances on online tests (Source: The dataset is provided by CL Educate and it can be found on the book code/dataset source site)

The goal of the clustering is to identify different segments or clusters of students such that depending upon the characteristics of each group, different remedial actions can be taken for improving the performance of the students in that group.

Step 1.

Read the Data.

Each row corresponds to a student. The columns measure student performance along multiple dimensions. The dimensions we will use are ACC – student accuracy of percentage of answers marked correctly and AVG_ATT – the average number of times a student attempted a question.

In [1]:
```
# Importing the Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
%matplotlib inline
```

In [2]:
```
dataset = pd.read_csv('../Data/useravg2018.csv', index_col=0)
dataset.head()
```

Out[2]:

	USER_ID	AVG_PERC	MAX_PERC	TOP3_AVG	NMOCKS	ATT	OK	ACC	AVG_ATT
1	302344	62.063708	83.095410	80.68	12	636	427	67.14	53.000000
2	304203	95.452029	99.654278	98.91	5	379	281	74.14	75.800000
5	307798	57.080305	81.070345	65.56	5	258	165	63.95	51.600000
7	351294	49.714915	89.949537	75.48	13	790	445	56.33	60.769231
9	624217	50.327109	76.604507	65.43	7	309	203	65.70	44.142857

Step 2.

Analyze the Data.

Based on inputs from the academic team, we decide to use ACC and AVG_ATT for clustering the students.

In [3]:
```
dataset.describe().loc[['count', 'mean', 'min', 'std', 'max']]
```

Out[3]:

	USER_ID	AVG_PERC	MAX_PERC	TOP3_AVG	NMOCKS	ATT	OK	ACC	AVG_ATT
count	4.000000e+02	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000	400.000000
mean	1.472902e+06	62.943748	80.768828	75.486350	9.697500	561.232500	363.807500	64.122975	57.140308
min	3.023440e+05	16.367214	24.213554	22.410000	4.000000	145.000000	83.000000	50.000000	30.444444
std	3.157168e+05	19.552976	17.068313	18.841274	3.252297	226.536842	157.724198	6.930649	9.730095
max	1.945951e+06	99.512102	99.937840	99.860000	15.000000	1128.000000	780.000000	82.150000	85.000000

Step 3.

Scale the Data.

Since K-means uses distance measures, it is important that the data be scaled prior to applying the clustering algorithm.

In [4]:
```
##### Load the X variable with columns ACC and AVG_ATT
X=dataset[['ACC', 'AVG_ATT']].values
```

In [5]:
```
#Scale the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Step 4.

Use Elbow Method to find a good K.

K-means is run for *K* values from 2 to 10. Note that only '*X*' is passed to the fit function of the K-means instance. No '*y*' or target variable is supplied. The inertia is plotted as a function of the number of clusters. We decide to use $K = 5$ as the optimal number of clusters. This is subjective.

```
In [6]: ▸ from sklearn.cluster import KMeans
        wcss = [ ]
        cl=[ ]
        for i in range (1, 11) :
            kmeans = KMeans (n_clusters = i, init = 'k-means++', random_state = 42)
            kmeans.fit (X_scaled)
            wcss.append(kmeans.inertia_)
            cl.append(i)
```

```
In [7]: ▸ sns.pointplot(x=cl, y=wcss)
        plt.title('The Elbow Method')
Out[7]: Text(0.5, 1.0, 'The Elbow Method')
```

Out[2]:

	USER_ID	AVG_PERC	MAX_PERC	TOP3_AVG	NMOCKS	ATT	OK	ACC	AVG_ATT
1	302344	62.063708	83.095410	80.68	12	636	427	67.14	53.000000
2	304203	95.452029	99.654278	98.91	5	379	281	74.14	75.800000
5	307798	57.080305	81.070345	65.56	5	258	165	63.95	51.600000
7	351294	49.714915	89.949537	75.48	13	790	445	56.33	60.769231
9	624217	50.327109	76.604507	65.43	7	309	203	65.70	44.142857

Step 5.

Create Clusters for Selected K.

The fit function generates the clusters. The predict function merely generates the cluster number for each point in the dataset.

```
In [8]: ▸ # Fit the Algorithm and Predict
        kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
        y_kmeans = kmeans.fit_predict(X_scaled)
```

```
In [9]: ▸ # assign the cluster number to the data.
        dataset['cluster']= y_kmeans
        dataset.head( )
```

Out[9]:

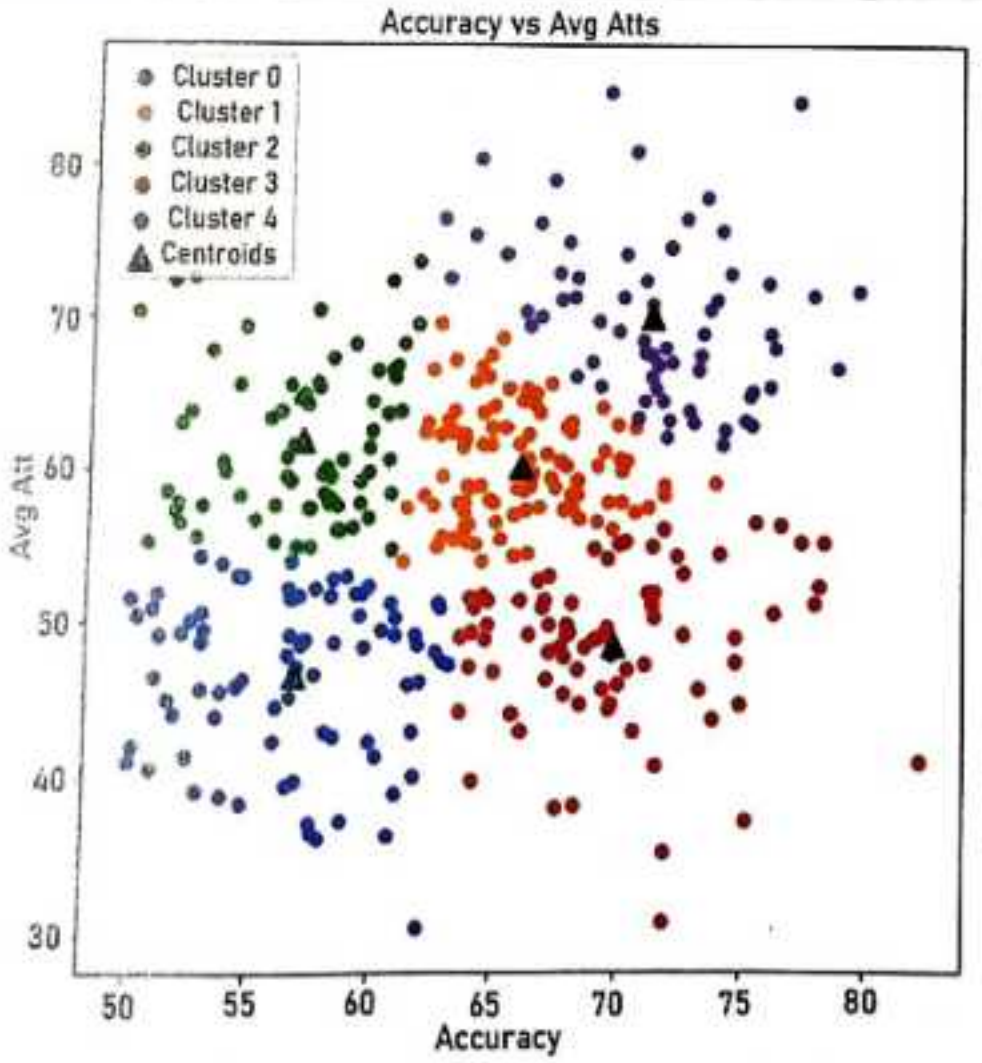
	USER_ID	AVG_PERC	MAX_PERC	TOP3_AVG	NMOCKS	ATT	OK	ACC	AVG_ATT	cluster
1	302344	62.063708	83.095410	80.68	12	636	427	67.14	53.000000	3
2	304203	95.452029	99.654278	98.91	5	379	281	74.14	75.800000	4
5	307798	57.080305	81.070345	65.56	5	258	165	63.95	51.600000	3
7	351294	49.714915	89.949537	75.48	13	790	445	56.33	60.769231	2
9	624217	50.327109	76.604507	65.43	7	309	203	65.70	44.142857	3

Step 6.

Evaluate the Clusters.

In [12]:

```
# Plot the 5 clusters
plt.subplots(figsize=(7, 7))

plt.scatter(dataset[dataset.cluster==0]['ACC'],dataset[dataset.cluster==0]['AVG_ATT'],
            s=30, label='Cluster 0')
plt.scatter(dataset[dataset.cluster==1]['ACC'],dataset[dataset.cluster==1]['AVG_ATT'],
            s=30, label='Cluster 1')
plt.scatter(dataset[dataset.cluster==2]['ACC'],dataset[dataset.cluster==2]['AVG_ATT'],
            s=30,label='Cluster 2')
plt.scatter(dataset[dataset.cluster==3]['ACC'],dataset[dataset.cluster==3]['AVG_ATT'],
            s=30,label='Cluster 3')
plt.scatter(dataset[dataset.cluster==4]['ACC'],dataset[dataset.cluster==4]['AVG_ATT'],
            s=30,label='Cluster 4')
#Plot the Cluster Centroids. We need the centroids unscaled.
kmc = scaler.inverse_transform(kmeans.cluster_centers_)
plt.scatter(kmc[:,0], kmc[:,1], c='black', s=100, marker='^', label='Centroids')
plt.title('Accuracy vs Avg Atts')
plt.xlabel('Accuracy')
plt.ylabel('Avg Att')
plt.legend(loc=2)
plt.show()
```



We use visualization to understand the clustering. This method is feasible when the number of features is 2. But when the number is more than 2, analysis by visualization becomes difficult, and analysis of the statistics by cluster as in Step 8 which soon follows might be more practical.

We may derive meanings from the clusters as follows.

Violet (Cluster 4): These students have high accuracy and take many practice exams (average attempts are high). Action: These students are doing well. We need to monitor them such that their accuracy is maintained going forward. We decide to label them as the 'High Achievers' segment.

Orange (Cluster 1): These students make attempts which are in the middle of the range, and their accuracy is also in the middle of the range. They should be asked to make more attempts. This would amount to more practice and improved accuracy. We decide to label them as the 'Near Achievers'.

Blue (Cluster 0): These students have low accuracy and make few repeat attempts on the preparatory exams. Action: They are making so few attempts that they may require mentoring to understand their circumstance and help them to get to make more attempts, *i.e.*, a more hand-on approach may be required with this segment than the possibly hands-off approach with Cluster 1. We decide to label this segment as the 'Remedial' segment.

Green (Cluster 2): These students make many attempts, but their scores are low. These students could be careless. Action: Provide them with special lessons which focus on rechecking answers for correctness. We label this class as 'Careless'.

Red (Cluster 3): These students have good accuracy but are not making many attempts. They could turn out to be better than our 'High achievers' if only they put in more effort and possibly took more practice exams. Action: Provide them with material to motivate and inspire them to the highest levels. We label this segment as the 'Conservative'.

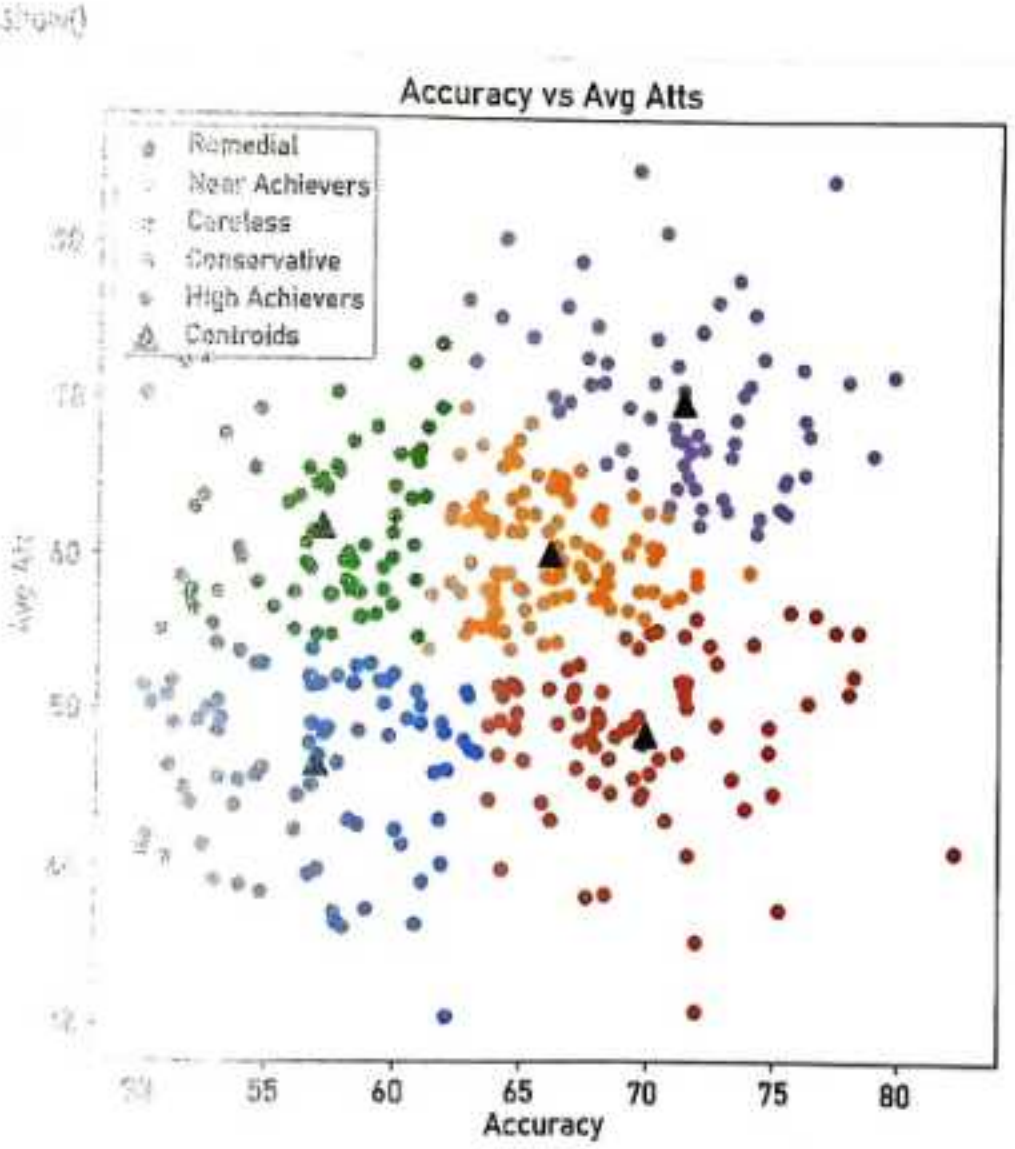
Step 7.

Relabel the Clusters.

This is purely to attach the meaning derived as above – which is subjective – to the derived clusters.

In [14]:

```
plt.subplots(figsize=(7, 7))

plt.scatter(dataset[dataset.cluster==0]['ACC'],dataset[dataset.cluster==0]['AVG_ATT'],
            s=30, label='Remedial')
plt.scatter(dataset[dataset.cluster==1]['ACC'],dataset[dataset.cluster==1]('AVG_ATT'),
            s=30, label='Near Achievers')
plt.scatter(dataset[dataset.cluster==2]['ACC'],dataset[dataset.cluster==2]['AVG_ATT'],
            s=30, label='Careless')
plt.scatter(dataset[dataset.cluster==3]['ACC'],dataset[dataset.cluster==3]['AVG_ATT'],
            s=30, label='Conservative')
plt.scatter(dataset[dataset.cluster==4]['ACC'],dataset[dataset.cluster==4]['AVG_ATT'],
            s=30, label='High Achievers')

plt.scatter(kmc[: , 0) , kmc[: , 1], s=100, c='black', marker='^' , label='Centroids')
plt.title( 'Accuracy vs Avg Atts')
plt.xlabel('Accuracy')
plt.ylabel('Avg Att')
plt.legend(loc=2)
plt.show()
```



Step 8.

Evaluate the Clusters.

We evaluate the clusters, this time by inspecting their statistical properties. When the number of dimensions is more than 2, cluster evaluation by statistical properties might be more practical. The same conclusions we derived from a visual inspections of the clusters should be possible from the inspection of the mean values of the 'ACC' and 'AVG_ATT' features by cluster.

```
In [10]: ▸ clusters = dataset.groupby('cluster')
```

```
In [11]: ▸ cldata = clusters[['ACC', 'AVG_ATT']].mean()
         cldata
```

Out[11]:

cluster	ACC	AVG_ATT
0	56.805000	46.705698
1	65.982178	60.266925
2	56.998333	61.875983
3	69.795897	48.551371
4	71.328657	70.107743

14.4 COLLABORATIVE FILTERING AND RECOMMENDER SYSTEMS

Collaborative filtering is an unsupervised algorithm. Recommender systems are built using various techniques including collaborative filters.

Recommender systems are at the crux of many cutting-edge businesses today. Recommender systems are used by businesses to recommend products to its customers which the customer is likely to buy. Any online advertising such as those that appear in Internet search or recommendations generated by online merchants all fall into this category. In this section, we will go through a very simple recommendation engine which uses collaborative filtering. Production recommendation systems use many algorithms including, but not limited to collaborative filtering. To understand collaborative filtering, we need to understand the concept of correlations amongst product ratings.

Consider the table below in Figure 14.4.1. Each column corresponds to a book while each row corresponds to readers of the books. Each cell contains the reader rating for the corresponding book in a range of 0 to 4.

user	BK1	BK2	BK3	BK4	BK5
user1	3	0	1	0	4
user2	3	0	0	4	1
user3	0	3	4	3	1
user4	4	2	1	1	2
user5	1	1	1	2	0

Figure 14.4.1 User ratings for books

Consider the ratings of book 'BK2' and 'BK3'. The ratings given by the various users are quite close, though not exact. For example, user1 gives ratings of 0 and 1 respectively, both ratings being low on the scale of 0 to 4 and user3 rates the books 3 and 4, both ratings being relatively high on the scale. We say in such a case that the ratings are positively correlated.

Consider 'BK4' and 'BK5' next. We see that the ratings seem to be in opposite order, in the sense, when a user gives a low rating for one book, the rating for the second book seems to be high and vice versa. For example, user1 rates 'BK4' as 0 or low on the scale while she rates 'BK5' as 4 or high on the scale. On the other hand, user2 rates 'BK4' as 4 or gives a high rating while he gives 'BK5' a low rating of 1. We say in such a case that the ratings have negative correlation.

As a final case, consider ratings of 'BK3' and 'BK5' where user1 rates 'BK3' low and 'BK5' high and user3 does the reverse. We also find that user2 rates both the books at the lower end of the scale. We do not find a strong positive or negative correlation. We say in this case that the ratings have little or no correlation.

14.4.1 Collaborative Filtering Hypothesis

Collaborative filtering uses the correlation between the ratings as a measure of the *similarity* of the books. That is, collaborative filtering's central thesis is, if ratings of two products are correlated, then if a user likes one of the products,

then he/she will like the other also. Hence, the second product can be provided to the user as a recommendation.

The intuition is fairly straightforward. A large number of users have given similar ratings to two products, *i.e.*, users either like the two products or dislike the two products since the correlation is high. So, if a new user comes along and likes one of the products, we expect the user to like the other highly correlated product, since most users who have used both the products like both. Of course, if the new user dislikes one of the two products, we expect her to dislike the other one too.

Before delving further to see if collaborative filtering hypothesis works, we need a formal method to define the correlation between two sets of ratings. For this purpose, Pearson's Correlation Coefficient given in Equation 14.4.1 is commonly used. In the formula, x_i corresponds to rating for a given item while y_i corresponds to the ratings for a second item. The mean value of the ratings is $\bar{x}$ and $\bar{y}$ respectively.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 (y_i - \bar{y})^2}}$$

Equation **14.4.1**

We provide an intuitive understanding of why the above formula measures correlation as explained earlier. If two sets of ratings are positively correlated, and if one of the ratings, say x_i is much higher than the mean value, then we expect the corresponding rating y_i also to be much higher than the mean and vice versa. Hence, the product of the terms $(x_i - \bar{x})$ and $(y_i - \bar{y})$ in both cases will be large positive numbers. So, the value of the correlation coefficient r_{xy} will be a large positive number in this case and the sum of several such large positive numbers will be a large positive number.

In case of negative correlation, if $(x_i - \bar{x})$ is a large positive number, $(y_i - \bar{y})$ will be a large negative number and vice versa. Hence, the product will be a large negative number and the sum of several such large negative numbers will be a large negative number.

When there is no significant correlation, there will be some cases where the product will be positive and some where it will be negative, thus cancelling

each other out. Hence, the product terms will be close to zero and thus, the sum will be a small number.

Hence, we see that the above formula can be used as a means of computing correlations between the ratings. The actual values of r_{xy} lie between –1.0 and 1.0. A value close to 1.0 implies a large positive correlation. A value close to –1.0 imply large negative correlation. A value close to 0.0 implies insignificant correlation.

The result of applying the formula to every pair of items in our synthetic dataset of Figure 14.4.1 is given below in Figure 14.4.2. This matrix is called a **correlation matrix**. Note that the correlation matrix is symmetrical along the diagonal from top left to bottom right. This is because the correlation as measured by Pearson's coefficient is symmetric, *e.g.*, the correlation between BK1 and BK3 is same as the correlation between BK3 and BK1.

	BK1	BK2	BK3	BK4	BK5
BK1	1.000000	-0.490098	-0.742379	-0.384900	0.541736
BK2	-0.490098	1.000000	0.834441	0.121268	-0.328719
BK3	-0.742379	0.834441	1.000000	0.104257	-0.130435
BK4	-0.384900	0.121268	0.104257	1.000000	-0.729800
BK5	0.541736	-0.328719	-0.130435	-0.729800	1.000000

Figure 14.4.2 Correlation of the ratings between all pair of books of Figure 14.4.1

We state our **simple recommendation algorithm** based on the correlations as below.

1. For a user, $user_i$, find, books BK_j which the user likes.
2. Recommend books Bk_j to the user where BK_j has strong positive correlation (high positive correlation coefficient value) to BK_i.

NOTE

In practice, all users will not have read or rated all the books in the library. So, the correlation coefficient between two books Bk_i and Bk_j may be calculated based on ratings of users *userA*, *userB* and *userC*. While during recommendation for *userD*, if the user has read Bk_i but not Bk_j, then BK_j may be provided as a recommendation if it is highly positively correlated with Bk_i.

In sophisticated recommendation engines, more details of the item, such as genre, and author for books may be used. However, our simple model which simply uses ratings of the items is still powerful, as we will see in our next Hands-on Exercise.

14.5 HANDS-ON EXERCISE: RECOMMENDATION SYSTEMS

We apply the above algorithm on a dataset of book ratings available as a csv file from [Source: http://cseweb.ucsd.edu/~jmcauley/datasets.html#goodreads].

Step 1.

Load the Ratings Data.

Note that there are 212395 ratings available in the dataset. Each entry corresponds to ratings given by a single user for a single book. Note that this data has only 'book_id', 'user_id' and 'rating', and no other information about the books. Information of the books is available in a different file and will be used only for displaying results, not for finding recommendations.

```
In [1]: ▸ import pandas as pd
          import numpy as np

          import warnings
          warnings.simplefilter('ignore')
```

```
In [2]: ▸ pathToRatings = '../Data/bookRatings.csv'
          ratings = pd.read_csv(pathToRatings)
          ratings.head(n=5)
```

Out[2]:

	userId	itemId	rating
0	22	264	2
1	1138	264	5
2	1160	264	3
3	1217	264	3
4	1572	264	3

In [3]:
```
print("Number of ratings:", ratings.shape [0])
print("Unique users:", ratings['userId'].unique().size)
print("Unique books:", ratings['itemId'].unique().size)
```

```
Number of ratings: 212395
Unique users: 3000
Unique books: 1891
```

Step 2.

Load Book Details.

Each itemid corresponds to a book. The name of the book and author corresponding to a given itemid is present in a different file, and we load it into the items DataFrame. **We will use this for showing the results of the recommendation system, but not for computing the recommendations**.

In [6]:
```
pathToDetails = ' . . /Data/bookInfo. csv'
items=pd.read_csv(pathToDetails)
```

In [7]:
```
items.sample(n=5)
```

Out[7]:

	itemId	title	details
1782	1737	Glass Sword (Red Queen, #2)	Victoria Aveyard
1388	776	The Art Book	Phaidon Press
1344	2545	Dance with the Devil (Dark-Hunter #3)	Sherrilyn Kenyon
1678	1264	Queen of Shadows (Throne of Glass, #4)	Sarah J. Maas
1726	607	City of Heavenly Fire (The Mortal Instruments, ...	Cassandra Clare

Step 3.

Build the Ratings Table.

The recommendation table which can be seen in the output in this section is called a pivot table. It has the users as rows, the items as columns, and the ratings as entries in the table. The ratings data we loaded into the ratings DataFrame has users, the books they have rated, and the ratings, all as columns of a DataFrame. The ratings DataFrame is converted into the pivotable form below. There are many missing values or NaNs (where NAN stands for Not a Number), because the readers have not rated the corresponding book. In real life, we cannot expect every user to have rated every item.

```
In [6]: ▸ pivotTable = ratings.pivot_table(index=['userid'],columns=['itemid'],values='rating')
          pivotTable.shape
Out[6]: (3000, 1891)

In [7]: ▸ pivotTable.sample(n=5)
```

Out[7]:

itemid	1	2	3	4	5	6	7	8	10	11	...	2998	3105	3132	3150	3231	3345
userid																	
23975	NaN	5.0	1.0	4.0	2.0	NaN	5.0	NaN	5.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
40731	NaN	NaN	NaN	5.0	5.0	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN
18496	NaN	5.0	NaN	3.0	3.0	NaN	NaN	2.0	5.0	5.0	...	NaN	NaN	NaN	NaN	NaN	NaN
7438	5.0	4.0	NaN	NaN	5.0	NaN	5.0	5.0	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN
18954	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 1891 columns

Step 4.

Building the Correlation Table.

Correlations are calculated by ignoring the missing values. For example, if let's say book1 had 120 ratings and book2 had 150 ratings, but 70 users rated both book1 and book2, the rest rating either book1 or book2 but not both, then the correlation between the two books would be calculated from the ratings of the 70 users who had rated both. If there are too few ratings for a book, then the correlation values computed between the books may not be statistically significant. To ensure that the correlations are statistically significant, we can specify to the corr() function the minimum number of ratings (min_periods) that need to exist for two items for the corresponding correlation values to be computed. If less than this number of common ratings exist, then the correlation is not calculated and is left as a NaN. In our case, we have specified this minimum number as 250.

```
In [8]: ▸ corrTable = pivotTable.corr(min_periods=250)

In [9]: ▸ #View the corrtable
          corrTable.head()
```

Out[9]:

itemid	1	2	3	4	5	6	7	8	10	11	...	2998
itemid												
1	1.000000	0.234702	0.388496	0.031843	-0.015047	0.186350	-0.008045	0.066032	0.090484	0.245843	...	NaN
2	0.234702	1.000000	0.135253	0.119011	0.068083	0.150993	0.278121	0.012330	0.251825	0.165599	...	NaN
3	0.388496	0.135253	1.000000	0.008118	-0.090979	0.170944	0.028740	0.116441	0.174361	0.278934	...	NaN
4	0.031843	0.119011	0.008118	1.000000	0.333149	NaN	0.123229	0.273784	0.296222	0.172932	...	NaN
5	-0.015047	0.068083	-0.090979	0.333149	1.000000	NaN	0.164837	0.397155	0.151060	0.048222	...	NaN

5 rows × 1891 columns

It is seen that there are many NaN or unknown values in the correlation table. It implies that for the corresponding pair of books, there were less than 250 simultaneous user ratings.

Step 5.

Checking Correlated Items.

We are now ready to find books whose ratings are closely correlated to a book of our choice. Since itemIds are meaningless to us, we use some helper functions in making sense of the quality of the correlated items.

itemsFromIDs is a function which given a list of item ID's, returns the details of each item.

```
In [10]: ▸ def itemsFromIDs (items, IDlist) :
               df = pd.DataFrame(columns=items.columns)
               for id in IDlist:
                   item = items[items.itemId == id]
                   df = pd.concat([df, item], axis=0)

               df.reset_index(inplace=True, drop=True)
               return df
```

```
In [11]: ▸ itemsFromIDs(items, [1, 2])
```

Out[11]:

	itemId	title	details
0	1	The Hunger Games	Suzanne Collins
1	2	Harry Potter and the Philosopher's Stone	J.K. Rowling, Mary GrandPré

The function relatedRecos works with book names. It takes a book name, retrieves the correspond itemId from the correlation table, finds the item ids of items which are most highly correlated, and then returns the details of these correlated items.

```
In [13]: ▸ def relatedRecos (itemName) :
               ItemID = items[items.title == itemName]["itemid"].iloc[0]
               my_corr = corrTable.loc[ItemID]

               top10 = my_corr.dropna( ).sort_values(ascending=False)[: 10]
               top10itemIDs = list(top10.index)

               top10Items = itemsFromIDs(items, top10itemIDs)
               return top10Items
```

Consider the results for the book 'Harry Potter and the Chamber of Secrets'. Nowhere so far, while finding correlated books had we explicitly or implicitly linked this book with the other Harry Potter books in the dataset. However, correlation of the user ratings has most interestingly picked out several of the Harry Potter books in the dataset to be closely related to this book.

In [14]:
```
itemName = 'Harry Potter and the Deathly Hallows'

top10Recos = relatedRecos(itemName)
top10Recos
```

Out[14]:

	itemId	title	details
0	25	Harry Potter and the Deathly Hallows	J.K. Rowling, Mary GrandPré
1	27	Harry Potter and the Half-Blood Prince (Harry ...	J.K. Rowling, Mary GrandPré
2	24	Harry Potter and the Goblet of Fire	J.K. Rowling, Mary GrandPré
3	21	Harry Potter and the Order of the Phoenix	J.K. Rowling, Mary GrandPré
4	18	Harry Potter and the Prisoner of Azkaban	J.K. Rowling, Mary GrandPre, Rufus Beck
5	23	Harry Potter and the Chamber of Secrets	J.K. Rowling, Mary GrandPré
6	2	Harry Potter and the Philosopher's Stone	J.K. Rowling, Mary GrandPré
7	10	Pride and Prejudice	Jane Austen
8	17	Catching Fire (The Hunger Games, #2)	Suzanne Collins
9	26	The Da Vinci Code (Robert Langdon, #2)	Dan Brown

In the second illustration, just based on user ratings, the sequels *Catching Fire* and *Mockingjay* comes up as correlated to 'The Hunger Games'.

In [16]:
```
1 itemName = 'The Hunger Games'
2
3 top10Recos = relatedRecos(itemName)
4 top10Recos
```

Out[16]:

	itemId	title	details
0	1	The Hunger Games	Suzanne Collins
1	17	Catching Fire (The Hunger Games, #2)	Suzanne Collins
2	20	Mockingjay (The Hunger Games, #3)	Suzanne Collins
3	3	Twilight (Twilight, #1)	Stephenie Meyer
4	12	Divergent (Divergent, #1)	Veronica Roth
5	73	The Host (The Host, #1)	Stephenie Meyer

Whether the recommendations generated by a collaborative filter are worthy is somewhat subjective. But their use in production systems have borne out their effectiveness. That is, recommendations generated by collaborative filtering lead to more user interest than recommendations which do not use such systems.

Recommender systems in production are a lot more sophisticated. They not only use explicit data (for example, user ratings) but also use implicit data (for example how much time an audience spent on a movie in the case of a movie recommendation system). Further features such as genre, etc. which needs to be manually generated are also used.

CHAPTER 14  EXERCISES

Review Exercises

1. What is an unsupervised learning problem? How is it different from supervised learning? Why is it difficult to evaluate models for unsupervised problems but not so for supervised problems?
2. Clustering is an example of an unsupervised learning. What is clustering? How can clustering be useful? Explain with an example.
3. How can the tightness of a cluster be defined? How can the tightness of multiple clusters be defined?
4. Explain how K-means performs clustering. Assume that inertia is defined.
5. How is the *K* in K-means determined? Why is this determination subjective?
6. Is it necessary to scale the features before applying K-means to the data? Why?
7. K-means may need to be run multiple times (for the same *K*) to find the best possible K-cluster. Explain why?
8. Define correlation between two series of numbers. What does a positive and negative correlation signify?
9. What is the formula for Pearson's correlation coefficient? Can Pearson's correlation coefficient measure correlation between points on a circle?
10. What is the intuition behind collaborative filtering for generating recommendations?
11. Write the algorithm for a recommender system which uses collaborative filtering.

12. One of the parameters used for building a correlation table of users versus ratings of products is the minimum number of users who should have rated the product. Why is this parameter necessary?

Investigative Exercises

1. K-means implementation in Sklearn has a hyperparameter called init. What is this parameter used for?
2. During the training of a linear or logistic regression model, the model parameters are determined from the training data to minimize some loss function. When training a decision tree, the conditions to be used in the internal nodes are determined. Does the K-means algorithm have any model parameters? What do you think happens during the training phase of a K-means?
3. Rerun the recommendation system Exercise with a smaller and larger value for min_periods. Try to explain your observations.

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#unsupervised-learning

APPENDIX A

Programming Prerequisites : Python

Machine learning and data science applications can be written in any programming language. At the time of this writing, Python is widely considered as the most popular language amongst data scientists. Python was designed to make programming easy and interactive. Python is also highly extensible, which has given rise to many libraries or packages over the years. A few of these packages have turned out to be instrumental in Python's popularity as a language for data science and ML. These packages include **NumPy**, **SciPy**, **Pandas**, **Matplotlib,** and **Sklearn**.

A thorough discussion on Python could easily require a book with a few hundred pages. There are books solely devoted to NumPy, Pandas, and Matplotlib. It is not possible to cover each of these subjects in detail in a book on machine learning. We cover the bare essentials of Python and some of its packages in the Appendix with the goal of making it easy to understand the code we use in the Hands-on Exercises. We do expect the reader to go to other resources for some code which is used in the book but not explained. In all cases, such code should not be critical in understanding the concepts this book intends to explain.

There are several environments available to Python developers where they can develop their applications. The **Jupyter** Notebook is an open-source web application that allows creation and sharing of documents that contain live code, equations, visualizations, and narrative text. While Jupyter is very easy to

use, it is not ideal for industrial strength development and debugging. For that purpose, there are other IDE's including **Spyder** and **PyCharm** which offer a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and visualization capabilities of a scientific package. There are many other development environments around. We will use Jupyter notebooks available through **Anaconda**. Anaconda is a package distribution tool which makes it easy to keep the included software (including Jupyter notebook and Python) up-to-date.

For this book, all code is in Python written in a Jupyter notebook. Familiarity with some other programming language will help understand the material in the following sections.

Python is an interpreted language and can be run interactively. In the following sections below, the cells corresponding to 'In []' correspond to a code cell. When the code in the cell is executed, it immediately prints an output if the code prints an output. In the sections below, if the cell produces an output, the output appears immediately below the code cell.

APPENDIX A.1 BASIC DATA TYPES

Basic data types supported by Python include integer, float, strings, and Boolean. The type of the variable *do not* have to be declared in Python. The type is inferred from the type of the object that is assigned to the variable. In the code snippet below, *x* is an int, *y* is a float, and *z* is a string. Variable names have restrictions and *need to start with a letter of the alphabet*, and are also *case-sensitive*.

```
In [1]: ▸ x = 1
          y = 2.3
          z = "str"
          b = True # False
          print(type(x), type(y), type(z), type(b))
          <class 'int'> <class 'float'> <class 'str'> <class 'bool'>
```

Usual operations such as '+', '–', etc. apply to integers and floats. Logical operators 'and', 'or,' and 'not' are available for Booleans. A common operation on strings, for example, concatenation, is shown below.

```
In [2]: ▸ xyz = "str1" + "str2"
          print(xyz)
          str1str2
```

It should be noted that strings are immutable, that is, once they are assigned, they cannot be changed.

APPENDIX A.2 LISTS

A List is a collection data type and is used to store an ordered collection of objects. The type of the objects in a list need not be the same. List objects can be complex data types, for example, a list of lists is possible. Two examples of lists are shown below.

```
In [3]: ▸ x = [1, 2, 3]
          y = ["a", 2, 3.6]
```

Lists are mutable, which means that entries in lists can be changed. It should be noted that if there are 3 objects in a list, they are indexed starting from 0 to 2.

```
In [4]: ▸ y = ["a", 2, 3.6]
          y[0] = "b"
          print(y)
          ['b', 2, 3.6]
```

Common operations, appending to a list, concatenating two lists, and getting the length of a list are shown here.

```
In [5]: ▸ y = ["Bhuvaneshwar", "Shami", "Yadav"]
          y.append("Bumrah")
          print(y)
          ['Bhuvaneshwar', 'Shami', 'Yadav', 'Bumrah']
In [6]: ▸ x = ["Irfan" , "Ashwin"]
          z = x + y
          print(z)
          ['Irfan', 'Ashwin', 'Bhuvaneshwar', 'Shami', 'Yadav', 'Bumrah']
In [7]: ▸ print(len(z))
          6
```

Lists can be used to implement multidimensional arrays as lists of lists. However, we will defer to the section on NumPy for a discussion on NumPy arrays and Pandas for a discussion on DataFrames which are the most common implementation of arrays used in data science applications written in Python.

APPENDIX A.3 SLICING

Slicing is a very common operation in Python and applies to many types of collections. We will briefly explain slicing in this section. Slicing is essentially a means of accessing a subset of contiguous values from an ordered collection. In the code below, we show three examples of slicing, where (*a*) values of the list between two indices are accessed, (*b*) values of the list from the beginning up to but not including the specified index is accessed, and (*c*) values from the specified index till the end are accessed. Note that in the case of (*a*) and (*b*), the value of the second index specified is not included in the subset returned.

```
In [8]: ▸ y = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
          print(y[1:4])
          print(y[:4])
          print(y[4:])
          [1, 2, 3]
          [0, 1, 2, 3]
          [4, 5, 6, 7, 8, 9]
```

The same method can be used to access subsets of data in multiple dimensions.

APPENDIX A.4 FOR LOOPS

The syntax of 'for loops' is shown below where 'i' is the loop variable and 's' is a sequence. A sequence can be a list or a string. It should be noted that the 'for' statement ends with a semicolon (':').

```
In [9]: ▸ s = ["a", "b", "c"]
          for i in s:
              print(i)
          a
          b
          c
```

The *body* of the 'for' statement instead of being enclosed in braces or within a 'begin' and 'end' keyword as is done in several programming languages is simply indicated via indentation. As can be seen below, an assignment statement and a print statement are both indented by one tab from the 'for' statement. Hence, both belong inside the 'for' loop. The print("outside") statement is not indented, and hence, does not belong inside the 'for' loop.

```
In [10]: ▸ s = ["a", "b", "c"]
           for i in s:
               x = i + "_"
               print(x)
           print("outside")
           a_
           b_
           c_
           outside
```

A 'for' loop to be run a fixed number of times from a start value to an end value can be implemented using the range function as shown below.

```
In [11]: ▸ start = 0
           stop = 5
           step = 1
           for i in range(start, stop, step):
               print(i)
           0
           1
           2
           3
           4
```

The start value and the step value can be omitted and they would then default to 0 and 1 respectively.

APPENDIX A.5 IF STATEMENT

When 'if' statements are declared as below without a 'then' keyword, the statements in the block following the 'if' statement corresponds to the 'then' part of the condition and requires indentation by a tab as with the body of the 'for' loop.

```
In [12]: x = 10
         c = 100
         if (x < c):
             print(x, "is small")
         else:
             print(x, "is big")
         10 is small
```

APPENDIX A.6 DICTIONARIES

A dictionary is a very powerful data structure which consists of (key, value) pairs, much like a dictionary that we use for storing words and their meanings. The keys in a dictionary have to be unique. Dictionary items can be accessed using their keys as shown here.

```
In [13]: x = {"Name": "Subhrajit",
              "Phone": 9999900000,
              "Address": "Bangalore"}
         print(x["Name"])
         Subhrajit
```

Adding a new item and deleting from a dictionary can be done as shown here. The code below also shows how one can check whether a key exists in the dictionary or not.

```
In [14]: x["Company"] = "CareerLauncher"
         del x["Phone"]
         print("Company" in x)
         print("Phone" in x)
         True
         False
```

All the elements of a dictionary can be accessed as below, using the keys() method which generates a sequence which can be used in a 'for' loop.

```
In [15]: x = {"Name": "Subhrajit",
              "Phone": 9999900000,
              "Address": "Bangalore"}

         for key in x.keys():
             print(key, ":", x[key])
```

```
Name : Subhrajit
Phone : 9999900000
Address : Bangalore
```

APPENDIX A.7 TUPLES

A tuple is an ordered **immutable** collection of objects, where the objects can be of varied type as shown below.

```
In [15]: ▸ record = ("Subhrajit", 9999900000, "Bangalore")
           print(record[1])
           9999900000
```

While the obvious difference between a list and a tuple is that the former is mutable and the latter is immutable, there are other differences that arise which are more subtle. A list is typically used to store a large collection of objects, usually of the same type, and the collection grows and shrinks over the life of the program. A tuple, due to its immutability, usually stores a fixed, usually small number of items which are of different types.

A tuple can be visualized as storing the row values of a table as in an Excel sheet. The number of columns in a table is fixed, hence the number of elements in the tuple would be fixed if it were representing a row. The number of items or rows in a table however, is usually not fixed. So if we wanted to store the values in a column of the table, the list would be a more appropriate data structure.

Python functions can return more than one value through a tuple. The values of the tuple can be extracted into multiple variables. Both these capabilities are shown here.

```
In [17]: ▸ def dummy_function():
               name = "subhrajit"
               number = "9999999999"
               city = "Bangalore"

               return (name, number, city)

           fn_ret_vals = dummy_function()
           name, phone, city = fn_ret_vals
           print(city)
           Bangalore
```

APPENDIX A.8 FUNCTIONS

A function can be defined using the 'def' word, followed by the function name, and the arguments enclosed in parentheses followed by a semicolon, as shown below. The value to be returned is indicated via a 'return' statement. Once defined, a function can be called from anywhere within the code with the appropriate parameter values specified.

```
In [18]: ▸ def complexMathFn(x, y):
               z = x**2 + y - 10
               return z
In [19]: ▸ res1 = complexMathFn(2, 3)
           print(res1)
           -3
In [20]: ▸ res2 = complexMathFn(3, 2)
           print(res2)
           1
```

APPENDIX A EXERCISES

Review Exercises

1. Why do we not have to declare the type of variables in Python?
2. What are the basic data types in Python?
3. A string data type is immutable. What does immutable mean?
4. What is the list data structure in Python? Is it immutable?
5. Do the values in a list have to be of the same type?
6. If x and y are lists, what is $x + y$?
7. How is the first element of a list accessed?
8. If $y = [2, 5, 6, 4, -1, 3, 2]$, what is $y[1:3]$? What is $y[:2]$? What is $y[3:]$?
9. Give the syntax of a ***for*** loop in Python including the body of the loop.
10. Give an implementation of a 'for' loop which iterates over the loop body for the values 0 to 5.
11. Give the syntax of a dictionary in Python. What are the components of a dictionary?
12. Give the syntax for adding and deleting an element from a dictionary.

13. A tuple corresponds to a record in a database table. Explain.
14. Can an element be added to a tuple?
15. What is the advantage of using tuples for returning values from a function?

Investigative Exercises

1. There is a list L with N elements. How can one generate a list which contains the last *m* elements of L, m < N?
2. How can one check if a key is in a dictionary?
3. Write a code for merging two dictionaries. Check for duplicate keys.

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#python

APPENDIX B

Programming Prerequisites : NumPy

NumPy is a library in Python built for numerical computations on multidimensional arrays of numbers. The name NumPy stands for Numerical Python. NumPy defines its own datatypes such as *np.int64 (64 bit integer), np.float32 (32 bit float), and np.float64 (64 bit float), etc.*

To use any Python library, including NumPy, it has to be first imported as below. Usually, the import statement also gives the library a short name. For NumPy, the short name used is np.

```
In [1]: ▸ import numpy as np
```

APPENDIX B.1 DECLARING NUMPY ARRAYS

NumPy can be used to define multidimensional arrays, where the dimensions can be 1, 2, 3, or even higher using the np.array() functional call.

Let's first consider one-dimensional arrays. A one-dimensional NumPy array is like a list of numbers. In the cell below, a 1-dimensional array is declared. The dimensions of a NumPy array are stored in an attribute called shape. Note that the shape of the one-dimensional array is defined by only one number, 3 in this example, which is the number of elements in the array. To access an element, only one index value has to be used as in 'x[0]'. Finally, note that index numbers start from 0.

```
In [2]: x = np.array([1, 2, 3], dtypes='float32')
        print(x)
        print(x.shape)
        print(x[0])
        [1. 2. 3.]
        (3,)
        1.0
```

In the cell shown here, a two-dimensional array with 2 rows and 3 columns is declared. Note that a list of lists is used to initialize the NumPy array. Two indices are required to access an element of the 2-dimensional array.

```
In [3]: z = np.array([[1, 2, 3], [4, 5, 6]], dtype='float32')
        print(z)
        print(z.shape)
        print(z[1, 1])
        [[1. 2. 3.]
         [4. 5. 6.]]
        (2, 3)
        5.0
```

A two-dimensional array with one row is very much like a one-dimensional array, but requires two indices for accessing elements. The shape contains two numbers, the first being 1—which is the number of rows, and a second number which equals the number of columns or elements in the array, as demonstrated in the cell below.

```
In [4]: y = np.array([[1, 2, 3]], dtype='float32')
        print(y)
        print(y.shape)
        print(y[0, 1])
        [[1. 2. 3.]
        (1, 3)
        2.0
```

In the cell below, a three-dimensional array is declared. Note that the shape now returns a tuple with 3 values, stating that it is a 3 × 2 × 3 array.

```
In [5]: w = np.array([[[1, 2, 3], [4, 5, 6]],
                      [[11, 12, 13], [14, 15, 16]],
                      [[21, 22, 23], [24, 25, 26]]],
```

```
                dtype='float32')
print(w)
print()
print("Shape:", w.shape)
print()
print("Array element:", w[2, 0, 1])
[[[1.  2.  3.]
  [4.  5.  6.]]

 [[11.  12.  13.]
  [14.  15.  16.]]

 [[21.  22.  23.]
  [24.  25.  26.]]]

Shape: (3, 2, 3)

Array element: 22.0
```

There are functions for creating certain standard arrays with a given shape. The code below creates a 1-dimensional array of evenly spaced numbers (9 of them) between a start (–2) and end point (2).

```
In [6]: ▸ x = np.linspace(-2, 2, 9)
          print(x.shape)
          print(x)
          (9,)
          [-2.  -1.5 -1.  -0.5  0.  0.5  1.  1.5  2.]
```

A 1-dimensional array of random numbers, uniformly distributed between two given numbers (10 and 90 in the code in this section) can be generated as shown here.

```
In [7]: ▸ x = np.random.normal(10, 90, 10000)
          print(x.shape)
          print(x[0:10])
          (10000,)
          [  52.10938688    8.06713821   28.98639282  129.32024708  -11.106695
             -8.91538551   31.09720306  -40.51974123   65.43199231  -58.2162403
```

The code below creates an array of zeros with shape 2 × 3.

```
In [8]: shape = (2, 3)
        x = np.zeros(shape, dtype = 'float32')
        print(x)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

An array initialized with ones can be as easily created.

```
In [9]: shape = (3, 2)
        y = np.ones(shape, dtype = 'float32')
        print(y)
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

The 'shape' is an attribute of a NumPy array and is a tuple. The values of the 'shape' tuple can be accessed as shown below.

```
In [10]: w = np.array([[[1, 2, 3], [4, 5, 6]],
                       [[11, 12, 13], [14, 15, 16]],
                       [[21, 22, 23], [24, 25, 26]]],
                       dtype='float32')
print(w.shape[0], w.shape[1], w.shape[2])
```

```
3 2 3
```

APPENDIX B.2 RESHAPING NUMPY ARRAYS

Oftentimes, the shape of a NumPy array needs to be changed without changing its content. One such situation is when we want to convert a 2-dimensional array with 1 row and several columns into a 1-dimensional array. This is required when there are some functions which accept a 1-dimensional array as an argument while the data is available as a 1xN dimensional array.

In the code below, y is a 2-dimensional array, with 1 row and 3 columns, *i.e.*, has the shape 1x3.

```
In [11]: ▸ y = np.array([[1, 2, 3]], dtype='float32')
           print(y)
           print(y.shape)
           print(y[0, 1])
```

```
[[1. 2. 3.]]
(1, 3)
2.0
```

The ravel() function converts a multidimensional array into a single dimensional array as shown here. Note that the array y1D is printed within single square brackets while the 2-dimensional array y was printed within two square brackets. Also note that now to access the elements of y1D, only one index has to be used.

```
In [12]: ▸ y1D = y.ravel( )
           print(y1D)
           print(y1D.shape)
           print(y1D[1])
```

```
[1. 2. 3.]
(3,)
2.0
```

A more general function for reshaping an array is the reshape() function. To get the same effect as above, we can use the reshape() function as below.

```
In [13]: ▸ y1Dv2 = y.reshape( 3,)
           print(y1Dv2)
           print(y1Dv2.shape)
           print(y1Dv2[1])
```

```
[1. 2. 3.]
(3,)
2.0
```

Thus, ravel() is simply a convenient function to use when one needs to reshape any n-dimensional array into a 1-dimensional array.

APPENDIX B.3 OPERATIONS ON NUMPY ARRAYS

Numpy provides functions that implement many operations on NumPy arrays. The implementation of these operations is extremely fast. *Hence, whenever manipulating arrays, one should use NumPy operations instead of defining custom Python functions to perform the same operations.*

Two NumPy arrays of the same dimension can be added, subtracted, multiplied, etc. The operations work on the corresponding elements of the two arrays. Below is an illustration of the addition operation on two arrays of dimension 2x3.

```
In [14]: ▸ z1 = np.array([[1, 2, 3], [4, 5, 6]], dtype='float32')
           z2 = np.array([[7, 8, 9], [10, 11, 12]], dtype='float32')
           z = z1 + z2
           print(z)

           [[ 8. 10. 12.]
            [14. 16. 18.]]
```

The sqrt() function generates a matrix of the same dimension where an element in the result array is the square root of the corresponding element in the argument array.

```
In [15]: ▸ z = np.aqrt(z1)
           print(z)

           [[1.        1.4142135 1.7320508]
            [2.        2.236068  2.4494898]]
```

There are functions for finding statistical measures of an array, for example min and max values and the sum of all the elements.

```
In [16]: ▸ print(z2.min())
           print(z2.sum())
           print(z2.mean())

           7.0
           57.0
           9.5
```

The metrics can also be computed along a particular dimension. For example, setting axis to 1 computes the sum for each row. Setting axis to 0 would have computed the sum for each column.

```
In [17]: ▸ z2 = np.array([[7, 8, 9], [10, 11, 12]], dtype='float32')
           print( np.sum(z2, axis=1 )

           [24.  33.]
```

Functions exist for linear algebra operations such as matrix multiplication and matrix inverse. Note that for multiplying two matrices using the dot product as in AxB, if the dimension of A is MxN, then dimension of B has to be NxP. With this in mind, the code below shows matrix multiplication on two matrices which satisfy the above requirement, resulting in a matrix of dimension MxP.

```
In [18]: ▸ z1 = np.array([[1, 2, 3], [4, 5, 6]], dtype='float32')
           z2 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]], dtype='float32')
           z = np.matmul(z1, z2)
           print(z1.shape, z2.shape)
           print(z.shape)

           (2, 3) (3, 4)
           (2, 4)
```

A matrix inverse can be computed as easily. Keep in mind that inverses exist only for square matrices and the dot product of a matrix and its inverse is the identity matrix.

```
In [19]: ▸ z = np.array([[1, 2], [3, 4]], dtype='float32')
           zinv = np.linalg.inv(z)
           id = np.matmu(z, zinv)
           print(id)

           [[1.  0.]
            [0.  1.]]
```

APPENDIX B EXERCISES

Review Exercises

1. What is the NumPy library?
2. What is the difference between a 1-dimensional array and a 2-dimensional array with a single row?
3. What will the shape attribute return for a 1-dimensional array with 100 elements?
4. Give the syntax for creating a two-dimensional NumPy array from a list of lists.
5. Give the syntax for the NumPy function which can be used to generate random numbers following the normal distribution.
6. How do we create a 2-D NumPy array of 1's with 5 rows and 5 columns?
7. Give the syntax for converting a 2-D NumPy array into a 1-D array.
8. What is the ravel() function?
9. Write the code for finding the square root of all the elements of a NumPy array Z.
10. What is the NumPy function for computing the dot product of two matrices which are stored as NumPy arrays? What conditions would the matrices have to satisfy for the function to work?

Investigative Exercises

1. How can one find the minimum value of each column of a 2-D NumPy array?

APPENDIX

Programming Prerequisites : Matplotlib pyplot

Visualization is often very useful to understand the nature of the dataset. Visualizations often quickly allow us to figure out the relationship between different variables when the relationship is not explicitly known. Matplotlib is a visualization library for Python. It has two modules, pylab and pyplot of which the latter is used more commonly. In this section, we will provide a brief introduction to the pyplot module.

Pyplot is usually imported with the short name plt as below.

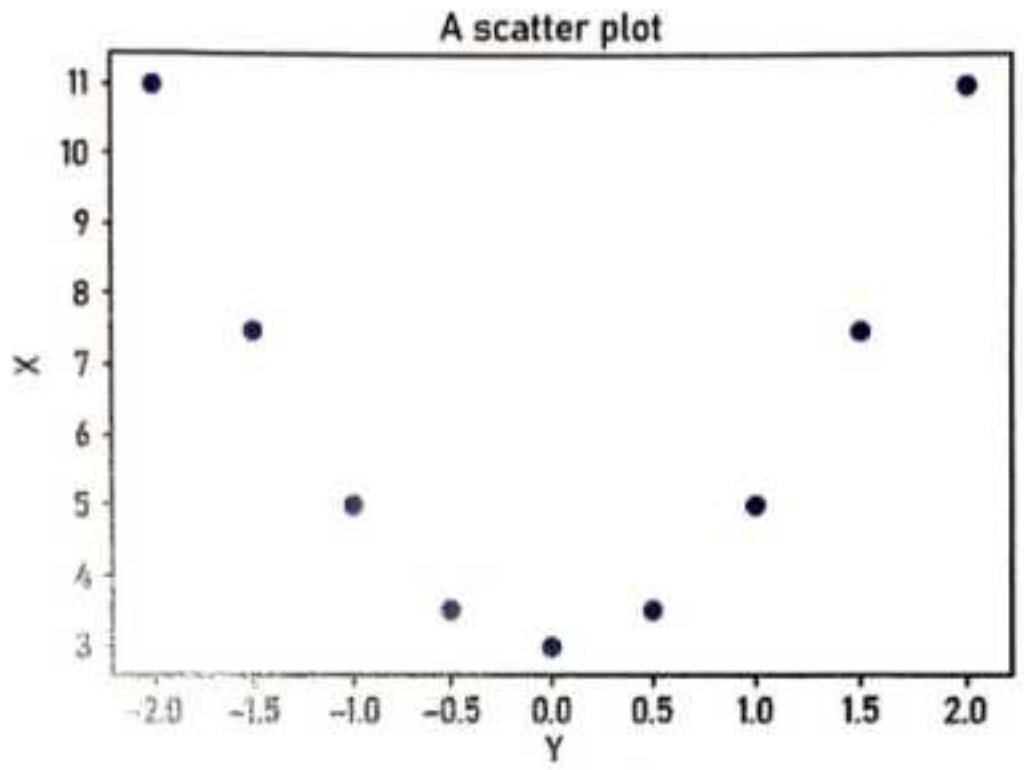
```
In [1]: ▸ import matplotlib.pyplot as plt
          import numpy as np
```

APPENDIX C.1 SCATTER AND LINE PLOTS

The simplest kind of plot which can be used to explore the relationship between variables X and Y simply plots the points (X_i, Y_i) where X_i and Y_i are the i^{th} values of the variables and Y_i corresponds to X_i. Note the optional labeling of the axes and the title for the plot.

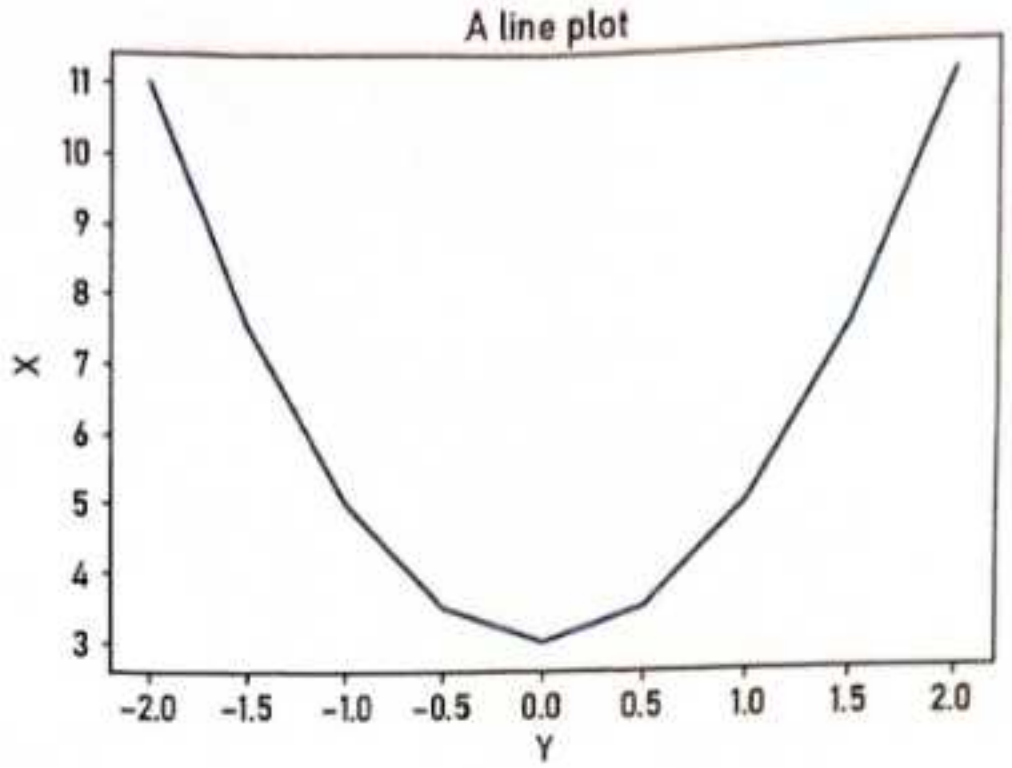
In [2]:
```
X = np.linspace(-2, 2, 9)
Y = 2*X*X + 3
```

In [3]:
```
plt.scatter(X, Y)
plt.ylabel('X')
plt.xlabel('Y')
plt.title('A scatter plot')

plt.show()
```



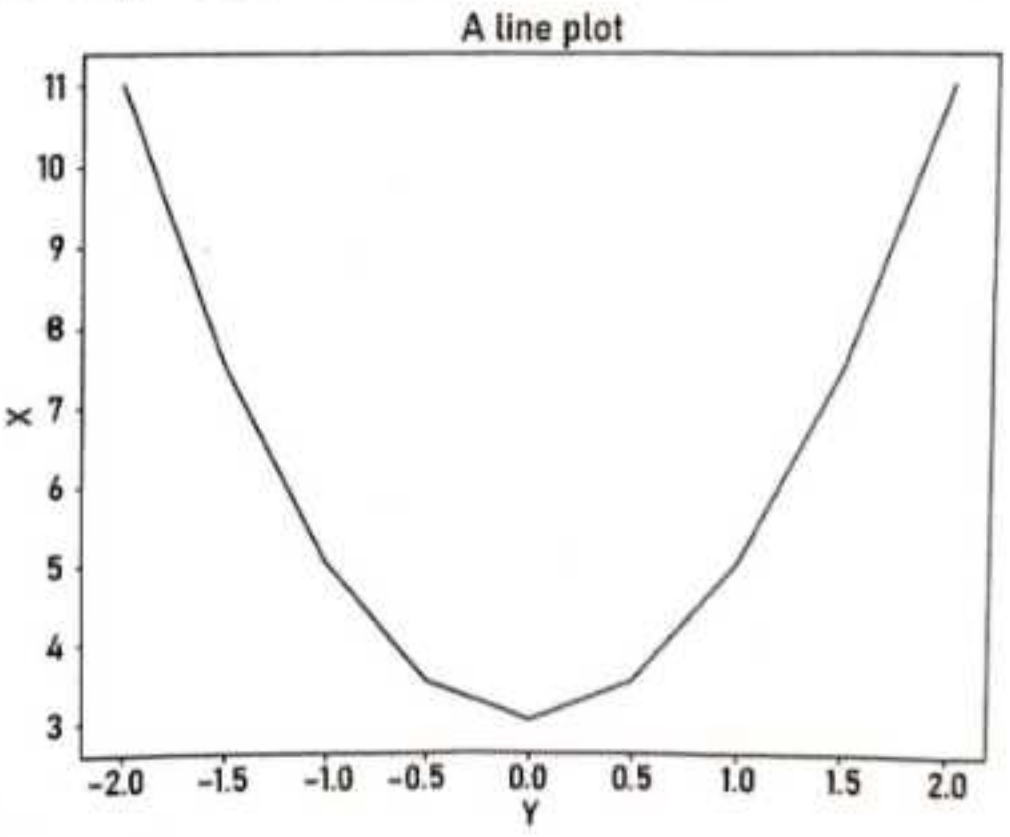
The corresponding line plot, connecting all the points can be created as shown here. *It should be noted that the points in the X array should be sorted from the minimum to the maximum value or vice versa*, else the graph will not plot as expected. In our example, X was generated in a sorted fashion, from –2 to 2. Hence, we did not require sorting the X values (and likewise the Y values)

In [4]:
```
plt.plot(X, Y)
plt.ylabel('X')
plt.xlabel('Y')
plt.title('A scatter plot')

plt.show()
```



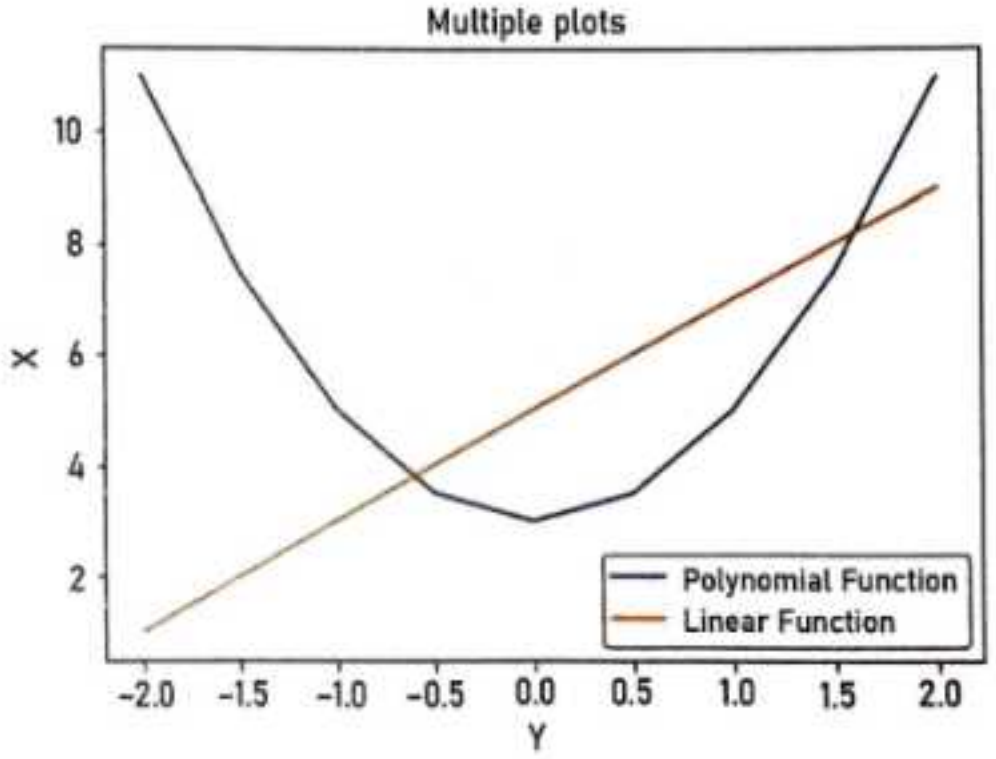
The size of the plot can be changed by specifying the figure size as shown here.

```
In [5]: ▸ plt.figure(figsize=(8, 6))
          plt.plot(X, Y)
          plt.ylabel('X')
          plt.xlabel('Y')
          plt.title('A line plot')

          plt.show()
```



Multiple plots can be made on the same area as shown in this section. Each plot can be given its own label. To display the labels, a call to the legend function has to be added.

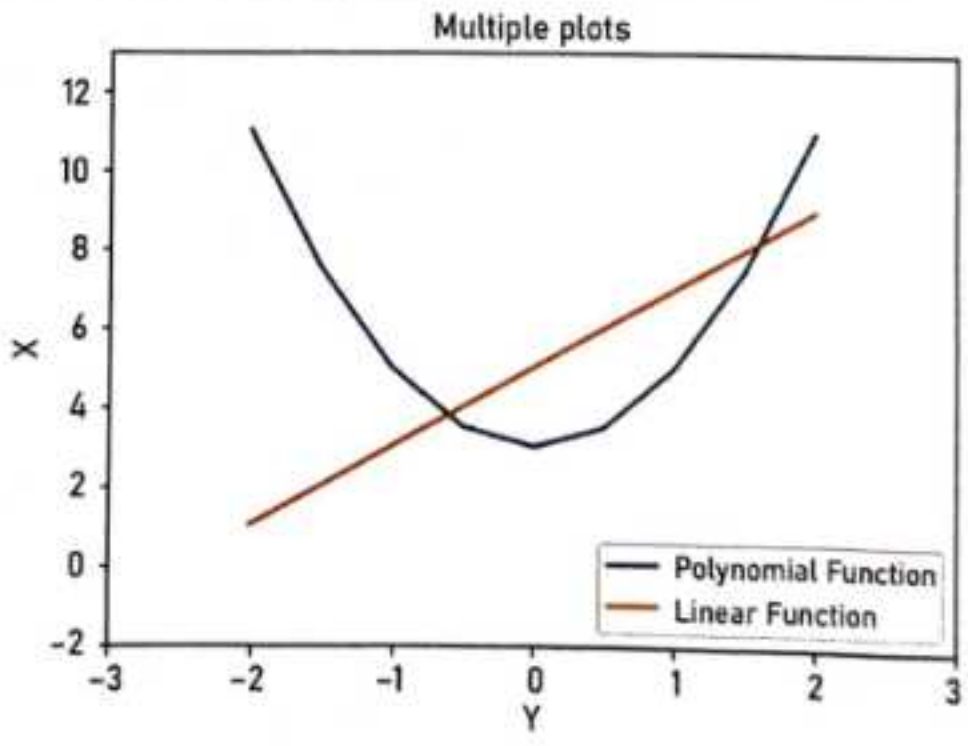
```
In [6]: X = np.linespace(-2, 2, 9)
        Y1 = 2*X*X + 3
        Y2 = 2*X + 5

        plt.plot(X, Y1, label="Polynomial Function")
        plt.plot(X, Y2, label="Linear Function")

        plt.ylabel('X')
        plt.xlabel('Y')
        plt.title('Multiple plots')

        plt.legend()
        plt.show()
```



By default, pyplot takes the minimum and maximum values of the *x* and *y* values. and creates the axes between the so derived limits. If the limits of the axes have to be changed, code similar to the one below can be used.

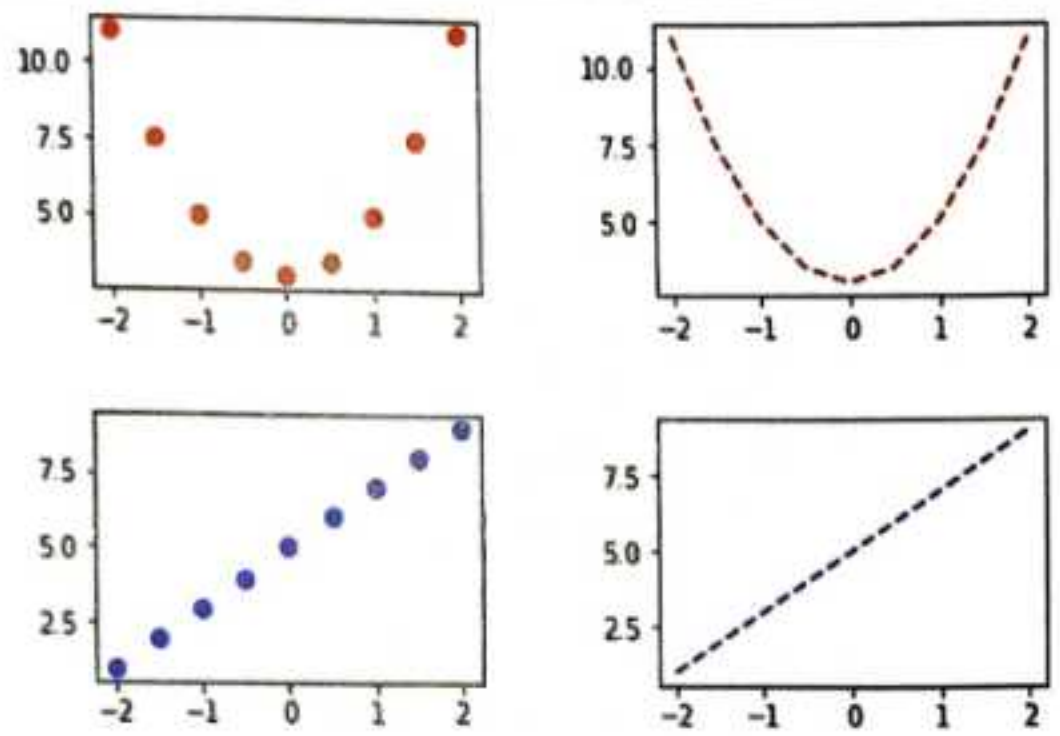
In [7]:

```
X = np.linespace(-2, 2, 9)
Y1 = 2*X*X + 3
Y2 = 2*X + 5

axes = plt.axes()
# Setting the bounds for the axes
axes.set_xlim([-3.0, 3.0])
axes.set_ylim([-2.0, 13.0])

plt.plot(X, Y1, label="Polynomial Function")
plt.plot(X, Y2, label="Linear Function")

plt.ylabel('X')
plt.xlabel('Y')
plt.title('Multiple plots')

plt.legend(loc='lower right')
plt.show()
```



Separate plots on the same display area can be plotted in a MxN grid. Below, we show four plots (called subplots) in a 2x2 grid.

```
In [8]: X = np.linespace(-2, 2, 9)
        Y1 = 2*X*X + 3
        Y2 = 2*X + 5

        plt.subplot(2, 2, 1)
        plt.scatter(X, Y1, c='red')

        plt.subplot(2, 2, 2)
        plt.plot(X, Y1, 'r - - ')

        plt.subplot(2, 2, 3)
        plt.scatter(X, Y2, c='blue')

        plt.subplot(2, 2, 4)
        plt.plot(X, Y2, 'b - - ')

        plt.show()
```

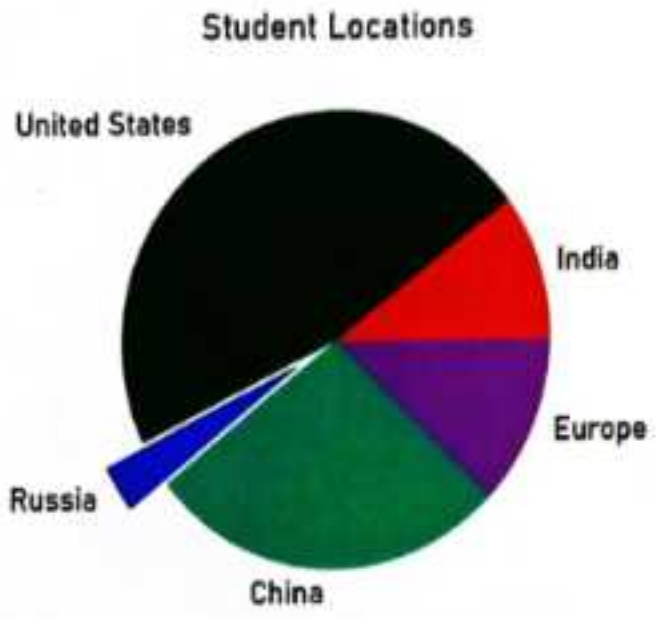


APPENDIX C.2 PIE CHARTS, BAR GRAPHS, HISTOGRAMS

Pie charts can be plotted as shown below.

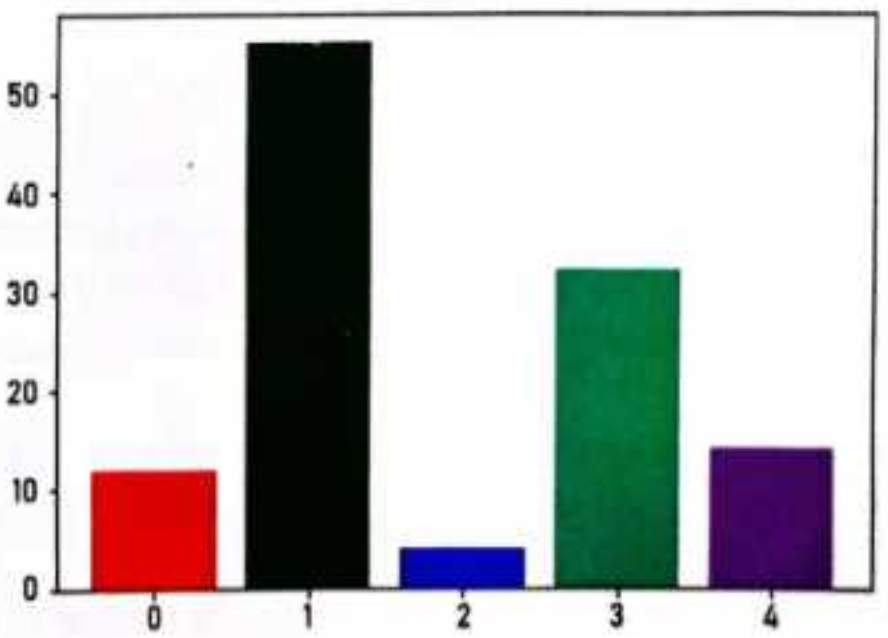
```
In [9]: values = [12, 55, 4, 32, 14]
        colors = ['r', 'g', 'b', 'c', 'm']
        |
        explode = [0, 0, 0.2, 0, 0.0]
```

```
labels = ['India', 'United States', 'Russia', 'China', 'Europe']
plt.pie(values, colors= colors, labels=labels, explode = explode)
plt.title('Student Locations')
plt.show()
```



A bar graph is shown below.

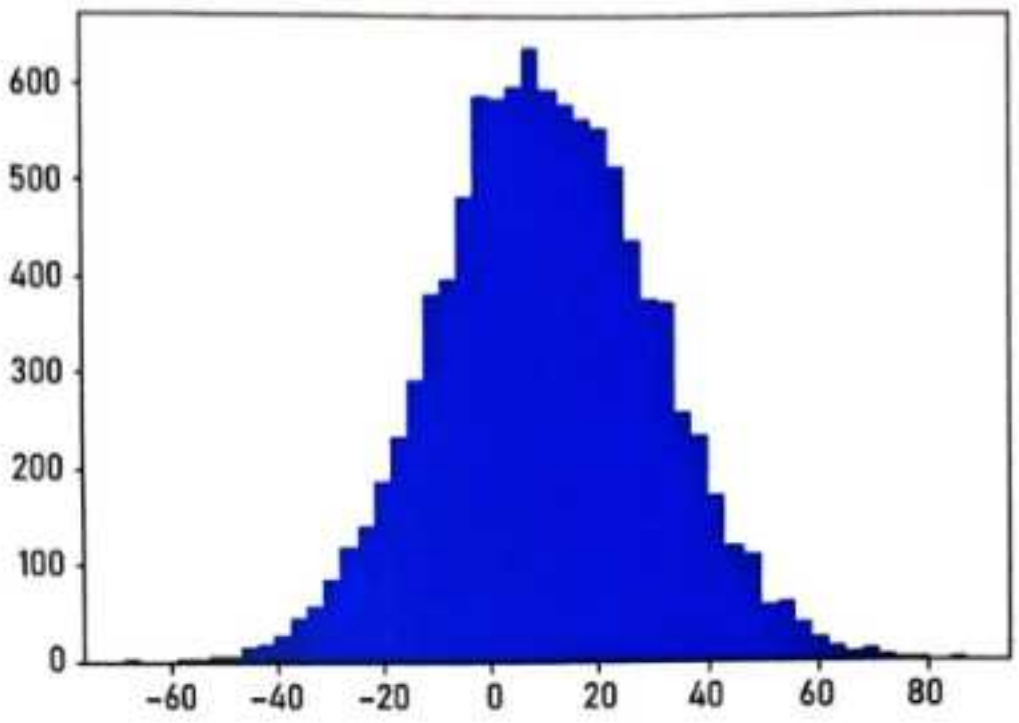
```
In [10]: ▸ values = [12, 55, 4, 32, 14]
           colors = ['r', 'g', 'b', 'c', 'm']
           plt.bar(range(0, 5), values, color= colors)
           plt.show()
```



A histogram is shown here. In this case, the data for the histogram has 10000 points and the histogram is plotted with 50 bins. The data itself corresponds to a normal distribution with a mean of 10 and standard deviation of 20. Details of a

normal distribution are discussed in Appendix E.7. The same code can be used to plot the histogram for any distribution.

In [11]:
```
x = np.random.normal(10, 20, 10000)
plt.hist(x, bins=50)
plt.show()
```



APPENDIX C EXERCISES

Review Exercises

1. Which Matplotlib package is referred to as plt?
2. Give the syntax for plotting points (x_i, y_i) using Matplotlib. Your code should include adding labels for the axes and a title for the plot.
3. Give an example of code which can be used to change the plot size of a Matplotlib plot.
4. What is the Matplotlib function for plotting a set of points connected by a line? What do you need to ensure about the sequence of the x_i's to ensure that the plot is correct?
5. What is the syntax for adding legends to a plot? What do you need to do to make the legends show on the plotted figure?
6. Write code which will plot 6 graphs, 2 in a row, for a total of 3 rows.
7. Give example code for plotting a pie chart.
8. Give example code for plotting a bar graph.
9. Give example code for plotting a histogram.

APPENDIX D

Programming Prerequisites : Pandas

Pandas is a powerful Python library for analyzing and processing structured data. Data in database tables or in Excel worksheets are examples of structured data. Structured data is usually though not necessarily, specified as rows and columns where the rows correspond to records (of employees for example) while columns correspond to features (*e.g.* name, age, salary, etc.). Data in its raw form is almost never ready to be fed into an analytic. In fact, estimates from various sources state that a data scientist spends about 80% of the time in analyzing and processing the data while the remaining 20% of the time is spent in building the analytic. So Pandas and the processing it allows is incredibly important in all data science projects.

Pandas is imported with the short form 'pd' as below.

```
In [1]: ▸ import pandas as pd
```

Pandas provides two data structures, a **series** and a **DataFrame**. We will focus only on the DataFrame structure in this brief tutorial.

APPENDIX D.1 PANDAS DATAFRAME

A DataFrame is like a table and stores tabular data. The DataFrame object comes with numerous functions which makes it easy to analyze the data and process it.

The data can be in a database or in an Excel document. We will work with data stored in Excel files or data stored in text files as comma-separated values (csv). In the code below, data from the game of cricket (pertaining to bowlers), is loaded into a DataFrame variable 'df'.

```
In [2]: df = pd.read_csv('bowlers.csv')
```

The methods head, tail, and sample can be used to examine the data in the DataFrame. Head prints the first few rows, 5 by default, tail prints the last few rows, while sample prints a random sample of rows. Sample prints a single row by default. Non-default behavior for all three functions can be specified by using the 'n' argument to specify the number of rows to print.

```
In [3]: df.head()
```

Out[3]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
0	A Ashish Reddy	270	386	19	20	143.0	95
1	A Chandila	234	242	11	12	103.0	92
2	A Choudhary	108	137	5	5	127.0	100
3	A Flintoff	66	105	2	3	NaN	67
4	A Kumble	983	1027	49	42	104.0	117

It should be noted that the numbers printed on the extreme left are the row number or more generally referred to as **row indices**.

```
In [4]: df.tail(n=6)
```

Out[4]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
350	Y Venugopal Rao	222	329	7	20	148.0	35
351	YA Abdulla	222	294	15	11	132.0	136
352	YK Pathan	1166	1350	45	80	116.0	56
353	YS Chahal	1219	1532	72	55	126.0	131
354	Yuvraj Singh	869	1042	39	71	120.0	55
355	Z Khan	2276	2691	119	99	118.0	120

```
In [5]: df.sample(n=5)
```

Out[5]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
138	JH Kallis	1799	2222	74	89	124.0	83
241	R Sharma	935	1079	42	44	115.0	95
172	LJ Wright	75	120	2	6	160.0	33
181	M Muralitharan	1581	1642	67	66	104.0	102
343	VY Mahesh	359	478	22	17	133.0	129

APPENDIX D.2 DATAFRAME SUMMARY

The number of rows and columns in the DataFrame and a summary description of each column can be retrieved as shown here.

```
In [6]: ▸ nrows, ncols = df.shape
          print('rows = ' , nrows)
          print('cols = ' , ncols)
          rows = 356
          cols = 7
```

The describe() method returns statistics for only numerical columns. The attribute 'count' refers to the count of the number of values in the particular column and in the absence of missing values, it should be the same as the number of rows returned by the shape attribute of the DataFrame. 'std' refers to the standard deviation of the values from the mean, while 25%, 50%, and 75% refer to the 1st, 2nd, and 3rd quartile values respectively.

```
In [7]: ▸ df.describe()
```

Out[7]:

	balls	runs	wkts	innings	eco	wktspermatch
count	356.000000	356.000000	356.000000	356.000000	354.000000	356.000000
mean	422.640449	516.654494	20.893258	21.266854	135.429379	82.834270
std	582.234748	686.102174	30.244354	27.148089	38.819358	48.724209
min	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000
25%	50.750000	71.750000	2.000000	4.000000	117.250000	50.000000
50%	192.000000	240.000000	9.000000	10.000000	129.000000	89.000000
75%	518.250000	603.750000	25.000000	26.000000	143.000000	112.250000
max	2989.000000	3295.000000	170.000000	134.000000	383.000000	300.000000

APPENDIX D.3 ACCESSING DATAFRAME ROWS AND COLUMNS

There are many ways of selective access of rows and columns of a DataFrame. Some of the most common methods are listed in this section. *Most of these methods create a new DataFrame.* For example, the code given here creates a new DataFrame with the three specified columns. The columns to be included are passed as a list enclosed within square brackets. From the shape of the new DataFrame, it can be seen the new DataFrame has the same number of rows as the original, but has only three columns.

```
In [8]: someCols = df['bowler', 'runs', 'wkts']]
        print(someCols.shape)
        someCols.head()
        (356, 3)
```

Out[8]:

	bowler	runs	wkts
0	A Ashish Reddy	386	19
1	A Chandila	242	11
2	A Choudhary	137	5
3	A Flintoff	105	2
4	A Kumble	1027	49

A set of contiguous rows can be accessed as shown below by specifying the row indices, where the row corresponding to the end index is not returned as part of the subset of rows.

```
In [9]: df [0:3]
```

Out[9]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
0	A Ashish Reddy	270	386	19	20	143.0	95
1	A Chandila	234	242	11	12	103.0	92
2	A Choudhary	108	137	5	5	127.0	100

A subset of rows and columns can be accessed as shown below. Both pieces of code are equivalent.

```
In [10]: df [0:3][['bowler' , 'innings' , 'wkts' ]]
```

Out[10]:

	bowler	innings	wkts
0	A Ashish Reddy	20	19
1	A Chandila	12	11
2	A Choudhary	5	5

```
In [11]: df[['bowler' , 'innings' , 'wkts' ]] [0:3]
```

Out[11]:

	bowler	innings	wkts
0	A Ashish Reddy	20	19
1	A Chandila	12	11
2	A Choudhary	5	5

Columns can also be accessed by using the column index. This becomes important when there are many columns. However, the syntax for accessing columns by index is different than that for accessing them by names. We illustrate the syntax on a synthetic dataset as shown here.

```
In [12]: ▸ cdf = pd.read_excel('multi.xlsx')
         print(cdf.shape)
         cdf.head()
         (10, 10)
```

Out[12]:

	col0	col1	col2	col3	col4	col5	col6	col7	col8	col9
0	0	1	2	3	4	5	6	7	8	9
1	1	11	12	13	14	15	16	17	18	19
2	2	12	22	23	24	25	26	27	28	29
3	3	13	32	33	34	35	36	37	38	39
4	4	14	42	43	44	45	46	47	48	49

The 'iloc' operator accesses rows and columns by their integer location (hence the name iloc) instead of by name. The 'iloc' operator takes a start and stop value for both the row as well as column locations. If no values are specified, as in the command below, the minimum and maximum locations are used by default. The command below prints the all the rows and columns 2 through 8.

```
In [13]: ▸ cdf.iloc[:,2:9]
```

Out[13]:

	col2	col3	col4	col5	col6	col7	col8
0	2	3	4	5	6	7	8
1	12	13	14	15	16	17	18
2	22	23	24	25	26	27	28
3	32	33	34	35	36	37	38
4	42	43	44	45	46	47	48
5	52	53	54	55	56	57	58
6	62	63	64	65	66	67	68
7	72	73	74	75	76	77	78
8	82	83	84	85	86	87	88
9	92	93	94	95	96	97	98

A list of integer locations can also be specified to the operator as below.

```
In [14]: ▸ cdf.iloc[:3, [5,2,3]]
```

Out[14]:

	col5	col2	col3
0	5	2	3
1	15	12	13
2	25	22	23

A single column can be returned by the iloc operator by specifying the column integer location as shown here. It should be noticed that in this case, the object returned is technically a Pandas Series object and not a DataFrame.

```
In [15]: col4 = cdf.iloc[4]
         print(type(df))
         print(type(col4)
         print(col4)
         <class 'pandas.core.frame.DataFrame'>
         <class 'pandas.core.series.Series'>
         col0     4
         col1    14
         col2    42
         col3    43
         col4    44
         col5    45
         col6    46
         col7    47
         col8    48
         col9    49
         Name: 4, dtype: int64
```

Finally, using two integer indices as below returns the element corresponding to the row and column specified by the indices.

```
In [16]: x = cdf.iloc[4, 5]
         print(x)
         45
```

APPENDIX D.4 CONDITIONAL ACCESS OF DATA

Oftentimes, we want to analyze data which satisfies certain conditions. Few examples of conditional access are given below. For example, if we want to analyze only those bowlers who have taken a single wicket, the code below can be used. Note that 'df.wkts' is a short form notation for accessing elements of the column 'wkts'.

```
In [17]: df[df.wkts==1].sample(n=5)
```

Out[17]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
210	NB Singh	25	14	1	2	56.0	50
305	SS Mundhe	7	5	1	1	71.0	100
22	AC Voges	56	74	1	7	132.0	14
321	T Henderson	36	40	1	2	111.0	50
62	BMAJ Mendis	30	36	1	2	120.0	50

Multiple conditions can be applied as given in this section. The code displayed here returns a DataFrame of the bowlers who have taken less than 10 wickets and played in 10 innings. Note that for the second condition, we use the long form syntax, df ["innings"] for accessing column values instead of the short form notation df.innings. Though not required in this particular case, when the column name has blank spaces, the long form notation has to be used.

```
In [18]: df[(df.wkts<10) & (df["innings"] <10)].sample(n=5)
```

Out[18]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
75	CK Kapugedera	17	49	0	3	288.0	0
92	DJ Muthuswami	84	101	4	6	120.0	67
58	BCJ Cutting	171	230	9	9	135.0	100
277	S Sriram	18	49	0	2	272.0	0
174	LPC Silva	6	21	0	1	350.0	0

APPENDIX D.5 ADDING AND DELETING COLUMNS

Adding a new column and deleting entire columns is easy. The code below creates a new column which contains the average wickets taken by each bowler. Note that dividing the 'wts' column by the 'innings' column divides each element in the 'wkts' column by each element of the 'innings' column. Using the head() command, we see that a new column 'AvgWkts' has been added to the DataFrame.

```
In [19]: df['AvgWkts']=df.wkts/df.innings
```

```
In [20]: df.head()
```

Out[20]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch	AvgWkts
0	A Ashish Reddy	270	386	19	20	143.0	95	0.950000
1	A Chandila	234	242	11	12	103.0	92	0.916667
2	A Choudhary	108	137	5	5	127.0	100	1.000000
3	A Flintoff	66	105	2	3	NaN	67	0.666667
4	A Kumble	983	1027	49	42	104.0	117	1.166667

A column can be removed using the drop() command with the value of the "axis" parameter set to 1. If "axis" is set to 0, then it would imply dropping a row. The drop command creates returns a new DataFrame with the indicated columns (or rows) dropped. The column is not dropped from the DataFrame to which the command

is applied. If the requirement is to remove the column from the DataFrame in question, then the "inplace" parameter needs to be specified as "True" as below.

```
In [21]: ▸ df.drop(['AvgWkts'], axis=1, inplace=True)
           df.head()
```

Out[21]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
0	A Ashish Reddy	270	386	19	20	143.0	95
1	A Chandila	234	242	11	12	103.0	92
2	A Choudhary	108	137	5	5	127.0	100
3	A Flintoff	66	105	2	3	NaN	67
4	A Kumble	983	1027	49	42	104.0	117

APPENDIX D.6 DEALING WITH NULL VALUES

Oftentimes, the value for a particular cell in the table may be missing. When this happens, we say that we have a **null** value in that cell. Many functions require all cell values in the DataFrame to be non-null. Hence, part of the data pre-processing step is to check for null values. If null values are found, then either the null values have to be replaced by non-null values, or sometimes, the row corresponding to the cell with the null value is removed from the analysis. There are several strategies for replacing null values with non-null values. These strategies are not discussed here.

The isnull() command checks for null values in columns of the DataFrame. When the isnull() command is followed by the sum command, the number of null values per column is reported. From the result of the code below, we can see that the 'eco' column has two null values while the rest of the columns do not have any.

```
In [22]: ▸ df.isnull().sum()
Out[22]: bowler        0
         balls         0
         runs          0
         wkts          0
         innings       0
         eco           2
         wktspermatch  0
         dtype: int64
```

The presence of nulls can also be detected by observing the output of the describe command as below. The describe command shows that every column has 356 values while the 'eco' column has only 354. This indicates that the eco column has two more missing values than the other columns. Note that we say that the 'eco' column has two more missing values – this is because the other columns may have missing values too.

```
In [23]: ▸ df.describe()
```

Out[23]:

	balls	runs	wkts	innings	eco	wktspermatch
count	356.000000	356.000000	356.000000	356.000000	354.000000	356.000000
mean	422.640449	516.654494	20.893258	21.266854	135.429379	82.834270
std	582.234748	686.102174	30.244354	27.148089	38.819358	48.724209
min	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000
25%	50.750000	71.750000	2.000000	4.000000	117.250000	50.000000
50%	192.000000	240.000000	9.000000	10.000000	129.000000	89.000000
75%	518.250000	603.750000	25.000000	26.000000	143.000000	112.250000
max.	2989.000000	3295.000000	170.000000	134.000000	383.000000	300.000000

If we wanted to find out which rows have null values, we can use the code given here.

```
In [24]: ▸ df[df.isnull().any(axis=1)]
```

Out[24]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
3	A Flintoff	66	105	2	3	NaN	67
70	CH Gayle	579	675	19	37	NaN	51

To remove the rows which contain missing data, the command below can be applied. It can be seen that the resultant DataFrame has 354 rows while our original DataFrame had 356. If the modification is desired in the original DataFrame, then the "inplace" argument to dropna() method has to be specified as True.

```
In [25]: ▸ dropdf=df.dropna()
          print(dropdf.shape)
          (354, 7)
```

We have seen above that the 3rd row of the original DataFrame for the bowler Andrew Flintoff had a missing value in the 'eco' column. Printing the first rows of the resultant DataFrame confirms that the 3rd row has indeed been dropped.

In [26]:
```
dropdf.head()
```

Out[26]:

	bowler	balls	runs	wkts	innings	eco	wktspermatch
0	A Ashish Reddy	270	386	19	20	143.0	95
1	A Chandila	234	242	11	12	103.0	92
2	A Choudhary	108	137	5	5	127.0	100
4	A Kumble	983	1027	49	42	104.0	117
5	A Mishra	2703	3191	142	126	118.0	113

APPENDIX D.7 SOME USEFUL DATAFRAME OPERATIONS

Pandas has numerous functions for performing many types of data processing operations including plotting the data. A select set of the operations are described in this section. We illustrate the operations on a new dataset of batting data of cricketers. In the dataset given here, the column 'batsman' contains the name of a player. Matches consist of two innings. Hence, the 'innings' column may have values 1 and 2 for the same value in the 'match' column. 'balls' column has values for the number of balls the batsman played in the given innings, and the 'runs' column contains the number of runs scored.

In [27]:
```
teamdf = pd.read_csv('bat.csv')
teamdf.head()
```

Out[27]:

	batsman	match	inning	balls	runs
0	A Rahane	1	1	45	47
1	A Rahane	1	2	70	77
2	A Rahane	2	1	33	39
3	A Rahane	2	2	2	1
4	A Rahane	3	1	102	87

The unique() command can be used to find out the unique values in a particular column. For example, if we wanted to find the batsmen in our dataset, we could run the command on the 'batsman' column as below.

In [28]:
```
batsmen = teamdf['batsman'].unique()
print (len(batsmen), batsmen)
4 ['A Rahane' 'R Pant' 'R Sharma' 'V Kohli']
```

To find the number of occurrences of each unique value, the value_counts function can be used.

```
In [29]: split = teamdf.batsman.value_counts()
         split
Out[29]: R Sharma    6
         A Rahane    5
         R Pant      5
         V Kohli     5
         Name: batsman, dtype: int64
```

The occurrences as a fraction of the total can be reported as below by setting the normalize parameter to 'True'.

```
In [30]: split = teamdf.batsman.value_counts(normalize=True)
         split
Out[30]: R Sharma    0.285714
         A Rahane    0.238095
         R Pant      0.238095
         V Kohli     0.238095
         Name: batsman, dtype: float64
```

Some commands work only on columns with numeric values. For example, if we want to find the total runs in the 'runs' column or the average values, then the code below can be applied.

```
In [31]: total_runs = teamdf['runs'].sum()
         total_runs
Out[31]: 776
```

The statistics can be calculated on multiple columns simultaneously. To do so, the names of the columns have to be specified as a list as below.

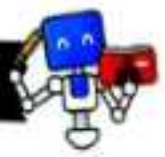
```
In [32]: teamdf[['runs','balls']].mean()
Out[32]: runs     36.952381
         balls    33.952381
         dtype: float64
```

The functions can be applied to all the columns of the DataFrame also as below.

```
In [33]: teamdf.mean()
Out[33]: match    1.857143
         inning   1.428571
         balls    33.952381
         runs     36.952381
         dtype: float64
```

Finally, if the new DataFrame that may have been derived from data processing has to be saved on disk, it can be saved in csv format using the to_csv() function.

```
In [34]: ▸ df.to_csv('savedData.csv')
```

APPENDIX D  EXERCISES

Review Exercises

1. Explain the different components of a Pandas DataFrame. What is a DataFrame commonly used for?
2. How can you see some random set of elements of a DataFrame?
3. How can you see a summary of a DataFrame? What information does the summary contain?
4. How can you find the number of rows and columns in a DataFrame?
5. Provide the syntax for accessing rows 4 to 10 and columns c1, c2, and c3 of a DataFrame.
6. How can you access all the data for all columns numbered 5 and above of a DataFrame?
7. What is the type of a single column of a DataFrame df when accessed as df["c1"]? What is the type when accessed as df[["c1"]]?
8. A DataFrame has a column 'age'. How can you find all entries for ages between 40 and 60?
9. A DataFrame has columns 'age' and 'dependents'. How can you create a new column 'avgDep' whose entries are the value obtained by dividing the corresponding value in the age column by the corresponding value in the dependent column?
10. What is the parameter "axis" used for when doing operations on a DataFrame?
11. Provide the code for reporting the number of null values per column of a DataFrame.
12. Provide the code for finding the rows which have null values in a DataFrame.
13. How can you find the unique values in a column of a DataFrame?

Investigative Exercises

1. Write code for adding a new column to a DataFrame such that the value of an entry equals the maximum value in the corresponding row of the original DataFrame.
2. Two DataFrames have columns with the same name. How can you perform a join of the two DataFrames on the columns with the same name?
3. A DataFrame has rows corresponding to various items and the columns correspond to the months of the year. The entries in the DataFrame correspond to the amount of the item sold in the month given by the column. How can you plot a graph of sales for every product in the DataFrame on the same graph?

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#pandas

https://youtu.be/vOsUloF5gO0

https://youtu.be/D7OMuQl6-qw

https://youtu.be/h556iPmjW6c

APPENDIX E

Statistics and Probability

In machine learning and data science in general, we work with data. The user needs to understand the data thoroughly for doing meaningful work with it. In this chapter we will discuss some concepts in Statistics and Probability which are frequently used in data science and machine learning and which are required to understand the discussions in the Machine Learning part of this book.

APPENDIX E.1 PROBABILITY

We first define some basic terminology used in probability.

Experiment

An experiment is any procedure that can be infinitely repeated and has a well-defined set of possible outcomes or results. Rolling a die is an experiment. Outcomes are getting a 1, 2, 3, 4, 5 or 6.

Outcome

An outcome is a possible result of an experiment. Each possible outcome of a particular experiment is unique, and different outcomes are mutually exclusive, that is only one outcome will occur on each trial of the experiment. On rolling a die, we can get a 1 or 2, we cannot get both 1 and 2.

Sample space

The set of all outcomes of an experiment is its sample space 'S'. The sample space for rolling a die is $S = \{1, 2, 3, 4, 5, 6\}$.

Event

An event consists of set of outcomes for a given experiment. Probabilities are usually assigned to events. For example, an event for rolling a die could be getting an even number. In this case, event consists of the outcomes {2, 4, 6}. Two events may not be mutually exclusive. If we define event E_1 as numbers less than or equal to 4, *i.e.*, {1, 2, 3, 4} and event E_2 as numbers equal to and greater than 4, *i.e.*, {4, 5, 6}, then E_1 and E_2 have an outcome in common, 4. E_1 and E_2 are not mutually exclusive.

Probability

Probability is a measure quantifying the likelihood that events will occur. Consider the example of rolling a die. If we assume all outcomes are equally likely, then we can define the probability of an event as below:

$$P(E) = \frac{\textit{Number of ways event can occur}}{\textit{Number of possible outcomes}} \qquad \textit{Equation } \textbf{E.1.1}$$

In the case of rolling a die, number of possible outcomes is 6. If the event we are interested in is that we get an even number, then E = {2, 4, 6}. Hence, $P(E) = \frac{3}{6}$ or 0.5.

Properties

Some properties of probabilities are stated without proof. There are some other properties which are not discussed here, and the interested reader should refer to other detailed sources on probability theory for further details.

- The probability of an event will always be between 0.0 and 1.0.
- If two events E_1 and E_2 are **mutually exclusive**, then the probability that either will happen is a sum of their individual probabilities, *i.e.*, (U stands for union, ∩ for intersection)

$$P(E_1 \cup E_2) = P(E_1) + P(E_2),\ E_1 \cap E_2 = \emptyset \qquad \textit{Equation } \textbf{E.1.2}$$

For example, if $E_1 = \{1, 3\}$ and $E_2 = \{2, 4, 5\}$, then

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) = \frac{2}{6} + \frac{3}{6} = \frac{5}{6} = 0.83.$$

- Probability of complementary events sum to 1. Two events are complementary if they are mutually exclusive and together, they consist of all outcomes in the sample space.

$$P(E_1) + P(E_2) = 1.0, \quad E_1 \cup E_2 = 5, \quad E_1 \cap E_2 = \emptyset$$

$$P(E_1) = 1.0 - P(E_2) \qquad \textit{Equation } \textbf{E.1.3}$$

On tossing a die, the event of getting a square number Esq and the event of getting a number which is not a square, *i.e.*, Ensq are complemetary. Esq is {1, 4} and $P(Esq) = \frac{2}{6}$. Hence, $P(Ensq) = 1 - P(Esq) = 1 - \frac{2}{6} = \frac{4}{6}$.

If there are more than 2 mutually exclusive events and their union is the complete sample space, then also, their individual probabilities sum up to 1.0.

APPENDIX E.2 VARIABLE TYPES

Understanding variable types is crucial in deciding upon the type of charts to use while doing exploratory data analysis or while deciding upon a suitable machine learning algorithm to be applied on our data. Variables of three types appear in data science.

- **Discrete**: where measurements are restricted to the integers 0, 1, 2 ... Some examples are the number of runs scored by a batsman or the number of children in a family. These are also called count variables.
- **Continuous:** where measurements can be made with precision and can theoretically have infinite values with a range. For example, weight of a person, distance travelled, etc.
- **Categorical:** have a finite number of values and may be numeric or non-numeric. For example, gender is usually one of ('Male', 'Female', 'Transgender') and is categorical. Grades in an examination could be one of ('A', 'B', 'C', 'D'), and is also categorical.

APPENDIX E.3 HISTOGRAMS

In data science, and in machine learning, we are provided data on various variables such as costs, temperature, height, scores, etc. The first step in studying the variables is to study the distributions of their values.

Histogram

Histogram is a diagram consisting of rectangles where each rectangle corresponds to an interval of values, and the area of the rectangle is proportional to the frequency of the variable in the given interval.

Consider the following values of the 'age' variable for a given set of customers:

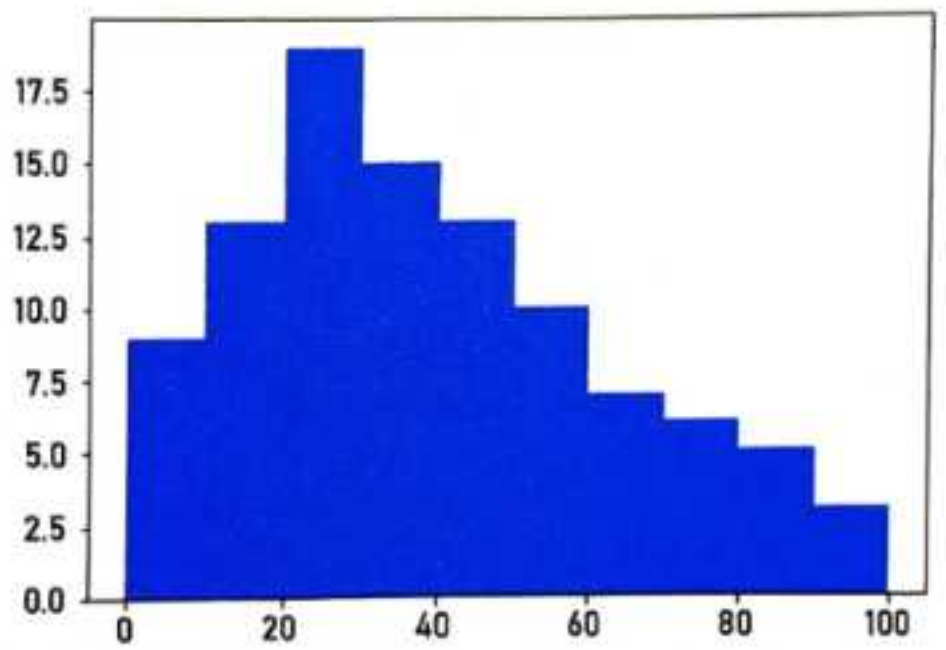
1, 1, 2, 3, 3, 5, 7, 8, 9, 10, 10, 11, 11, 13, 13, 15, 16, 17, 18, 18, 18, 19, 20, 21, 21, 23, 24, 24, 25, 25, 25, 25, 26, 26, 26, 27, 27, 27, 27, 27, 29, 30, 30, 31, 33, 34, 34, 34, 35, 36, 36, 37, 37, 38, 38, 39, 40, 41, 41, 42, 43, 44, 45, 45, 46, 47, 48, 48, 49, 50, 51, 52, 53, 54, 55, 55, 56, 57, 58, 60, 61, 63, 64, 65, 66, 68, 70, 71, 72, 74, 75, 77, 81, 83, 84, 87, 89, 90, 90, 91.

To study the distribution, we typically divide the entire range of values into a small set of intervals, and then plot the frequency of occurrence of the values in each interval. Assuming intervals of size 10, the values can be assigned into the intervals or bins as given here (all except the last bin is considered open at the right boundary, *e.g.* 20 belongs to the third bin and not the second bin in the example):

[0, 10): 1, 1, 2, 3, 3, 5, 7, 8, 9
[10, 20): 10, 10, 11, 11, 13, 13, 15, 16, 17, 18, 18, 18, 19
[20, 30): 20, 21, 21, 23, 24, 24, 25, 25, 25, 25, 26, 26, 26, 27, 27, 27, 27, 27, 29
[30, 40): 30, 30, 31, 33, 34, 34, 34, 35, 36, 36, 37, 37, 38, 38, 39
[40, 50): 40, 41, 41, 42, 43, 44, 45, 45, 46, 47, 48, 48, 49
[50, 60): 50, 51, 52, 53, 54, 55, 55, 56, 57, 58
[60, 70): 60, 61, 63, 64, 65, 66, 68
[70, 80): 70, 71, 72, 74, 75, 77
[80, 90): 81, 83, 84, 87, 89
[90, 100]: 90, 90, 91

The code for dividing the distribution and then plotting it as a histogram in Python is provided here:

```
In [1]: ▸ import matplotlib.pyplot as plt
          %matplotlib inline
          sampleData = [1, 1, 2, 3, 3, 5, 7, 8, 9,
                        10, 10, 11, 11, 13, 13, 15, 16, 17, 18, 18, 18, 19,
                        20, 21, 21, 23, 24, 24, 25, 25, 25, 25, 26, 26, 26, 27, 27, 27, 27, 27, 29,
                        30, 30, 31, 33, 34, 34, 34, 35, 36, 36, 37, 37, 38, 38, 39,
                        40, 41, 41, 42, 43, 44, 45, 45, 46, 47, 48, 48, 49,
                        50, 51, 52, 53, 54, 55, 55, 56, 57, 58,
                        60, 61, 63, 64, 65, 66, 68,
                        70, 71, 72, 74, 75, 77, 81, 83, 84, 87, 89,
                        90, 90, 91 ]

          plt.hist(sampleData, bins =[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
          plt.show()
```



It should be pointed out that an easier way of plotting such histograms is by simply specifying the number of bins to use instead of explicitly specifying the boundaries of the bins.

The visualization tells us, for example, that most of the customers are in the age group [20, 30]. It also tells us that there are more younger people (age < 30) than older people (age > 60).

APPENDIX E.4 MEASURES OF CENTRAL TENDENCY (AVERAGES)

When a set of counts or continuous observations are reduced to a single summary statistic describing the distribution, we have a measure of central tendency or an average. We often refer to the average of a distribution, for example average salary, average marks in an exam, and average rainfall, etc.

Mean, Median, and Mode are typical measures of the average value of a distribution.

Mean

Mean is the sum of the values divided by the size of the distribution. For example, if there are M values in a distribution, then the mean is defined as:

$$\bar{x} = \frac{x_0 + x_1 + \ldots + x_{M-1}}{M} = \frac{1}{M}\sum_{i=0}^{i=M-1} x_i \qquad \textit{Equation } \textbf{E.4.1}$$

Median

Median is the middle value when the numbers are sorted and arranged in order. For example, if the values in the distribution are 2, 5, 6, 8, 9, 10, 13, then there are 7 numbers in the distribution. The middle value is the 4th value which is 8 for the given distribution. If the distribution has an even number of elements, then the median is the mean of the two middle values.

Mode

Mode is the most commonly occurring value in the distribution. If the distribution has values 0, 1, 1, 1, 2, 3, 4, and 4, then the mode is 1 since it occurs more often than any other value. If more than one value occurs equally often, then the mode is the average of these values.

Mean can be quite misleading at times. In a village of 10,000 people, even if one person has lost a leg in an accident, the village will be reported to have 1.9999 legs per person, and almost everyone will have higher than average legs! Or, when we calculate the earnings in a small group of relatively poor people, the presence of a billionaire will make everyone look super-rich. Therefore, one data point can easily skew the average.

The median which refers to the mid-point of all the values if we arrange in a descending or ascending order is often a better representation of the representation of the average (our villagers on an average have two legs surely, the median value)

In machine learning problems, when we have outliers (a few very high or low values at extreme ends of the range), it may be safer to use median as a representation of the average.

APPENDIX E.5 MEASURES OF DISPERSION

A histogram shows the distribution of the data from which one can get a sense of the middle value as well as a sense of how spread out the data is. Measure of central tendency or averages provide just one number, the 'middle' value, to describe the distribution. Measures of dispersion provide a number to quantify the spread of the values of the distribution.

Consider the two distributions in Figure E.5.1 (*Source:* Wikipedia article on Standard deviation). Both have the same mean value. But the values of the red distribution are very close to its mean value, while the values of the blue distribution have a significant spread. We can say that the red distribution has low dispersion while the blue has high dispersion. We discuss a few mathematical measures of dispersion next.

Range

Range is the difference between the lowest and highest value of the data and is very sensitive to any outliers. For the age distribution discussed earlier, the range is [1, 91]

Percentile

Percentile is the value below which a given percentage of observations in a group of observations falls. For example, the 25th percentile is the value below which 25% of the observations lie. The 25th percentile is called the 1st quartile, the 50th percentile is called the 2nd quartile, and the 75th percentile is called the 3rd quartile. The 50th percentile is the same as the median.

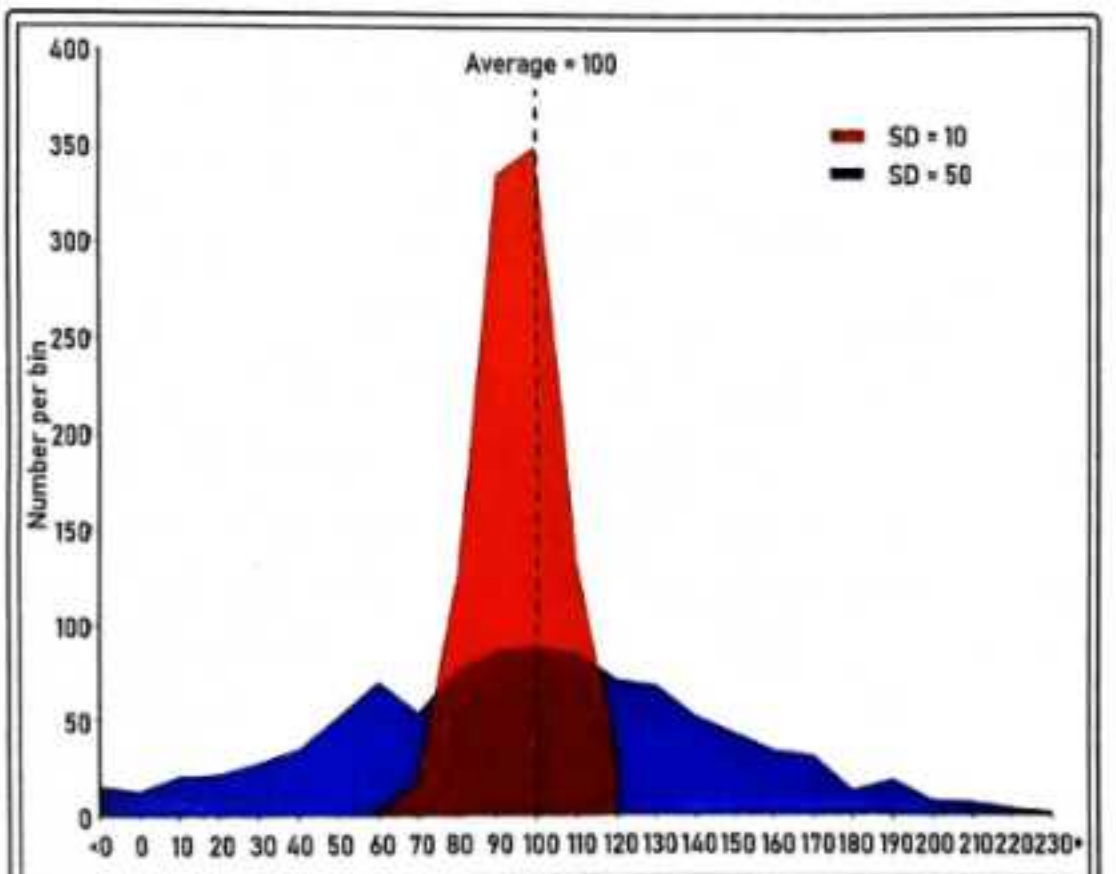
Interquartile Range (IQR)

IQR is the distance between the 25th and 75th percentiles and thus represents the 'central half' of the data. It is unaffected by the outliers or extremes.

Standard Deviation

The standard deviation is defined in Equation E.5.1. Note that the symbol σ is used for standard deviation. Standard deviation is a widely used measure of the spread or dispersion of a distribution. It is appropriate for 'well-behaved' symmetric data and is also influenced by outliers.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{i=N}(x_i - \bar{x})^2} \qquad \textit{Equation } \textbf{E.5.1}$$



Example of samples from two populations with the same mean but different standard deviations. Red population has mean 100 and SD 10; blue population has mean 100 and SD 50.

Figure E.5.1 Two different distributions. Both have the same mean. The red distribution has a smaller spread while the blue has a larger spread.

Variance

Variance is simply the square of the standard deviation. The measures of central tendency as well as dispersion can easily be calculated by using functions provided by various Python libraries. In this section, we show the

code for calculating the measures using NumPy. "sampleData" is the age data we used in the discussions earlier in the chapter.

```
In [2]: ▸ import numpy as np
          from scipy import stats

          sampleMean = np.mean(sampleData)
          sampleMedian = np.median(sampleData)
          sampleMode = stats.mode(sampleData)
          print("Mean:", sampleMean)
          print("Median:", sampleMedian)
          print("Mode:", sampleMode[0][0])
          Mean: 39.26
          Median: 36.0
          Mode: 27
```

```
In [3]: ▸ sampleSD = np.std(sampleData, ddof=1)
          print("Sample SD:", sampleSD)
          Sample SD: 23.719156474006066
```

APPENDIX E.6 EFFECT OF OUTLIERS

As an illustration of how the measures discussed earlier behave in the presence of outliers, let us try to apply the measures on M.S. Dhoni's batting performance over the years in one day matches.

Year	Mat	Inns	Runs	Avg	SR
2004	3	3	19	9.5	135.71
2005	27	24	895	49.72	103.11
2006	29	26	821	41.05	92.98
2007	37	33	1103	44.12	89.6
2008	29	26	1097	57.74	82.3
2009	29	24	1198	70.47	85.57
2010	18	17	600	46.15	78.95
2011	24	22	764	58.77	89.88
2012	16	14	524	65.5	87.63
2013	26	20	753	62.75	96.05

2014	12	10	418	52.25	92.07
2015	20	17	640	45.71	86.84
2016	13	10	278	27.8	80.12
2017	29	22	788	60.62	84.73
2018	20	13	275	25	71.43
2019	18	16	600	60	82.3

Over the years, how would you describe Dhoni's yearly performance? Is the median or mean better as a measure? Is the standard deviation important?

If the above data were loaded in a DataFrame object using Pandas, a describe() method called on the DataFrame would yield the following statistics:

```
df = pd.read_csv ('/ml/Dhoni.csv')
df.describe ()
```

	Year	Matches	Inns	Total_Runs	Avg	SR
count	16.000000	16.000	16.000000	16.000000	16.000000	16.000000
mean	2011.500000	21.875	18.562500	673.312500	48.571875	89.954375
std	4.760952	8.500	7.553972	324.603495	16.431555	14.276618
min	2004.000000	3.000	3.000000	19.000000	9.500000	71.430000
25%	2007.750000	17.500	13.750000	497.500000	43.352500	82.300000
50%	2011.500000	22.000	18.500000	696.500000	50.985000	87.235000
75%	2015.250000	29.000	24.000000	839.500000	60.155000	92.297500
max.	2019.000000	37.000	33.000000	1198.000000	70.470000	135.710000

The dataset has 16 observations, one per year from 2004 to 2019. Dhoni's year-wise batting statistics are as follows:

Summary Statistic	2004–2019
Mean	48.5
Median (50%)	50.98
Range	9.5–70.47
Interquartile Range	43.35–60.15
Standard Deviation	16.43

If you observe closely, in the year 2004, Dhoni played only 3 matches and averaged 9.5 runs. In the context of his performance, this is an outlier. The table below shows the same calculation done by ignoring the year 2004.

Summary Statistic	2004–2019	2005–2019
Mean	48.5	51.17
Median (50%)	50.98	52.25
Range	9.5–70.47	25–70.47
Interquartile Range	43.35–60.15	44.91–60.31
Standard Deviation	16.43	13.15

It can be seen that the Interquartile and Median are more insulated from the effect of outliers than the mean and the standard deviation.

Boxplots. Boxplots are an effective way of visualizing the interquartile range, median, as well as get a sense of the outliers. We show an example of boxplot below.

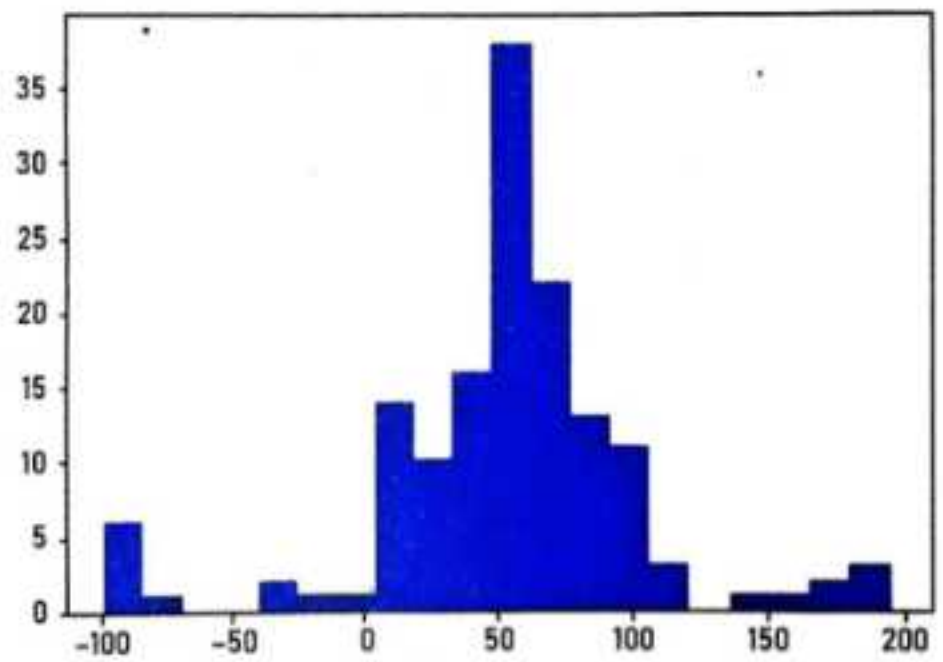
We create a synthetic dataset below. To understand the dataset, note that np.random.rand(100) generates an array of 100 random number between 0 and 1. Since we multiply the numbers by 100, *spread* is an array of 100 numbers between 0 and 100. *center* is simply an array of 25 numbers, each having value of 50 which is a number in the middle of the range [0, 100]. *flier_high* consists of 10 numbers which are between [100, 200]. *flier_low* consists of 10 numbers between [–100,0].

In [4]:
```
# Fixing random state for reproducibility
np.random.seed(19680801)

# fake up some data
spread = np.random.rand (100) * 100
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
```

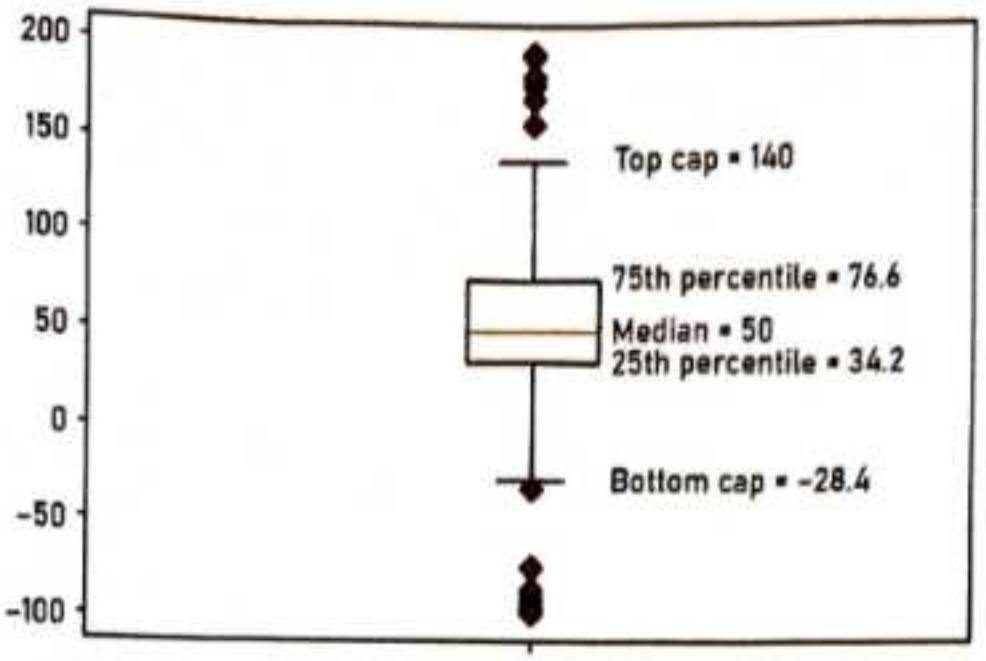
A histogram of the distribution is shown in this section. The flier_high and flier_low are clearly visible as outliers on the two ends of the distribution.

Random sampling is not appropriate in all situations. This is especially true for time series data. If we capture temperature data over the course of the day, we will see a pattern of the temperature rising as the day moves towards noon and declining during evening hours. This pattern repeats. If we picked a random sample of such time series data, the repeating pattern may be missing from the sample. In such cases, as an example, if we have data for say, 10 years, we can use the data from the last year as the sample.

In [8]:
```
data = np.concatenate((spread, center, flier_high, flier_low))
plt.hist(data, bins=20 )
plt.show()
```



A boxplot of the same data is shown below. The basic function is plt.boxplot(). We use some additional code to annotate the boxplot for our data. The annotated boxplot is shown below. The default for determining outliers are values which are greater than the top cap, (75th percentile + 1.5*IQR) and values which are less than the bottom cap (25th percentile - 1.5*IQR). In our example, the IQR is 42.4 (= 76.6 - 34.2). Hence, the top cap is 140.6 (=76.6 + 1.5*42.4) and the bottom cap is similarly -28.4.



APPENDIX E.7 THE NORMAL AND THE UNIFORM DISTRIBUTION

In this section, we will discuss two distributions which show up very often in data analysis.

The Normal Distribution

If we were doing a survey and measuring the heights of a large number of individuals, say, all men in India, the chances are that the data would follow a normal distribution, where the mean of the height would have the highest frequency of observations, with the frequencies falling away on either side of the mean. Two normal distributions are shown in Figure E.7.1. They have the same mean value but different standard deviations due to the difference in the variations.

Many distributions that appear in nature have been shown to follow a normal distribution.

In a normal distribution, 68.3% of the observations lie within one standard deviation from the mean, 95% are within two standard deviations, and 99.8% within three standard deviations of the mean. These properties of normal distributions are used in many mathematical processes.

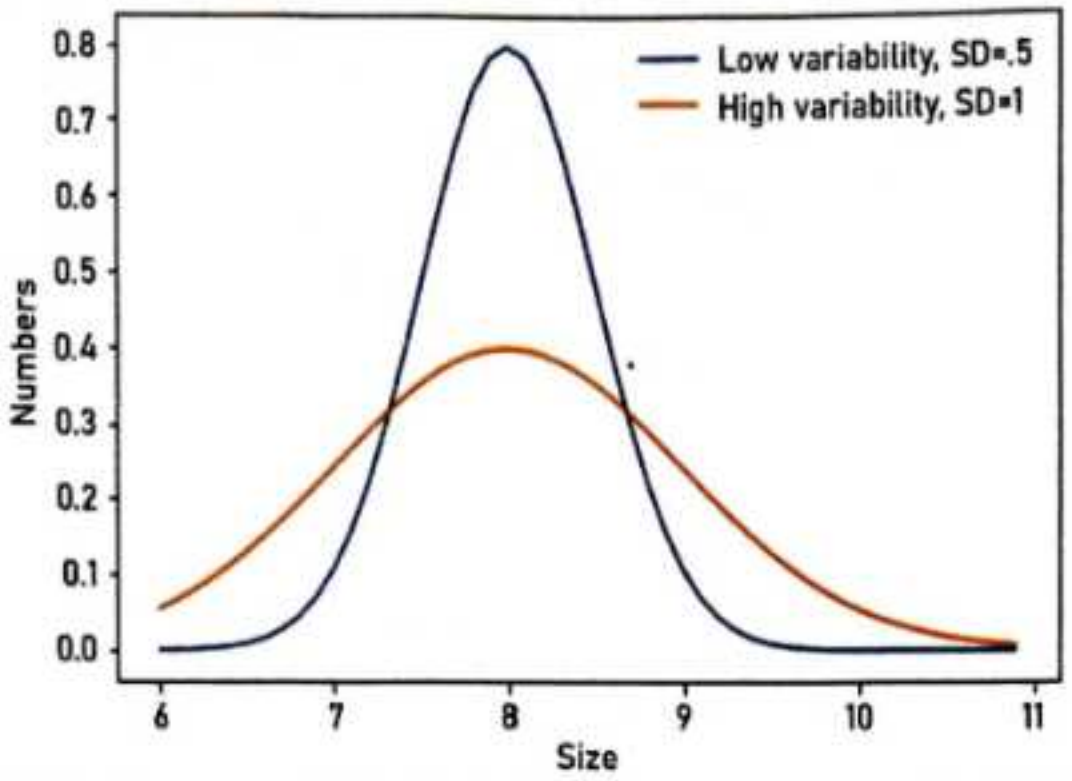
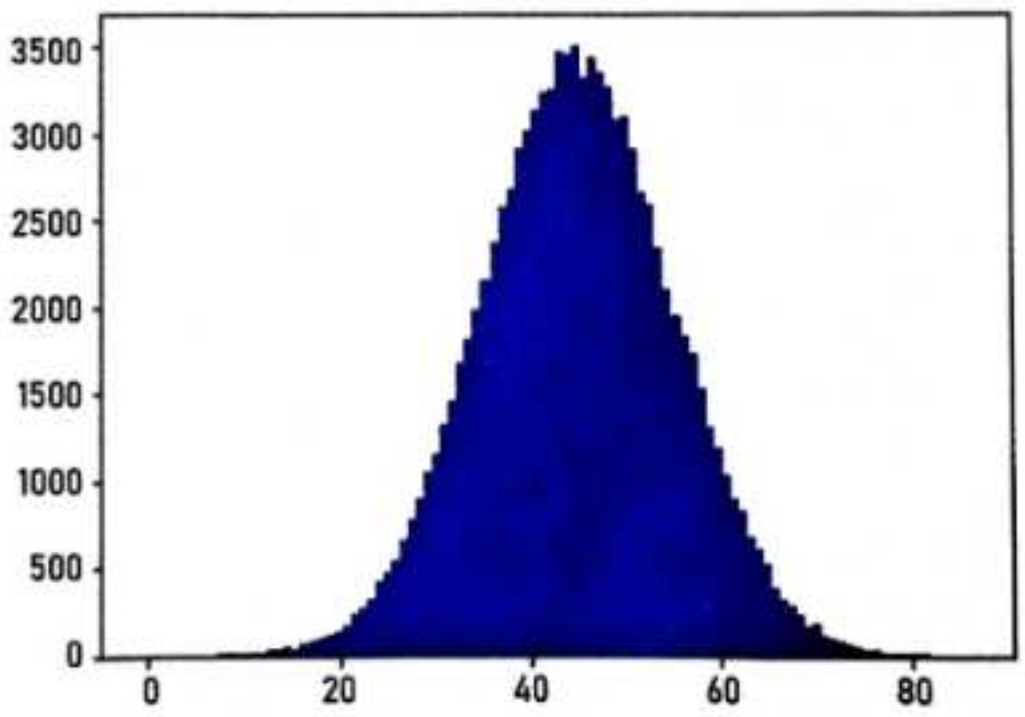


Figure E.7.1. Two normal distributions

A random distribution generated using the NumPy function is shown below. The distribution has a mean of 45 and a standard deviation of 10.

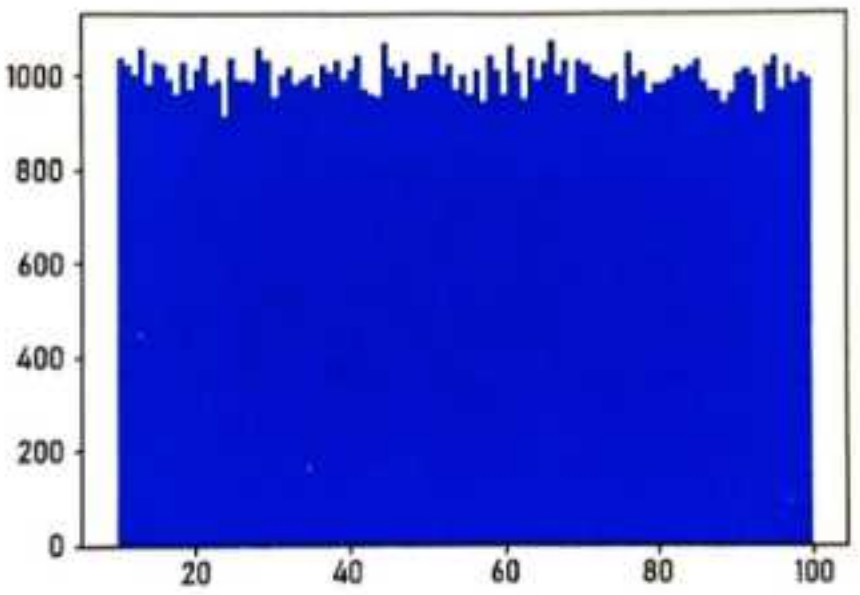
```
In [11]: ▸ xn = np.random.normal(45, 10, 100000)
           plt.hist(xn, bins=100)
           plt.show()
```



Uniform distribution

A uniform distribution has a finite range. The values of the distribution are equally distributed within the range. A uniform distribution generated using NumPy is shown in this section. The range of the distribution is [10, 100]. The mean and median of a uniform distribution is the value which is at the middle of the range. In our case, it would be 55 (or close to it). Pseudo-random numbers, which we will study in a subsequent section, follow the uniform distribution.

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

np.random.seed(10)
xu = np.random.uniform(10, 100, 100000)

plt.hist(xu, bins=100)
plt.show()
```



APPENDIX E.8 SAMPLING

In statistics, sampling refers to identifying a subset of data points which are representative of the whole population. It is often not feasible to collect data of an entire population, possibly because the exercise could take too long or be too expensive. That is when sampling becomes important.

Sampling is used in machine learning for a slightly different purpose. Given a dataset from which to build a predictive model, we need to split the data into a

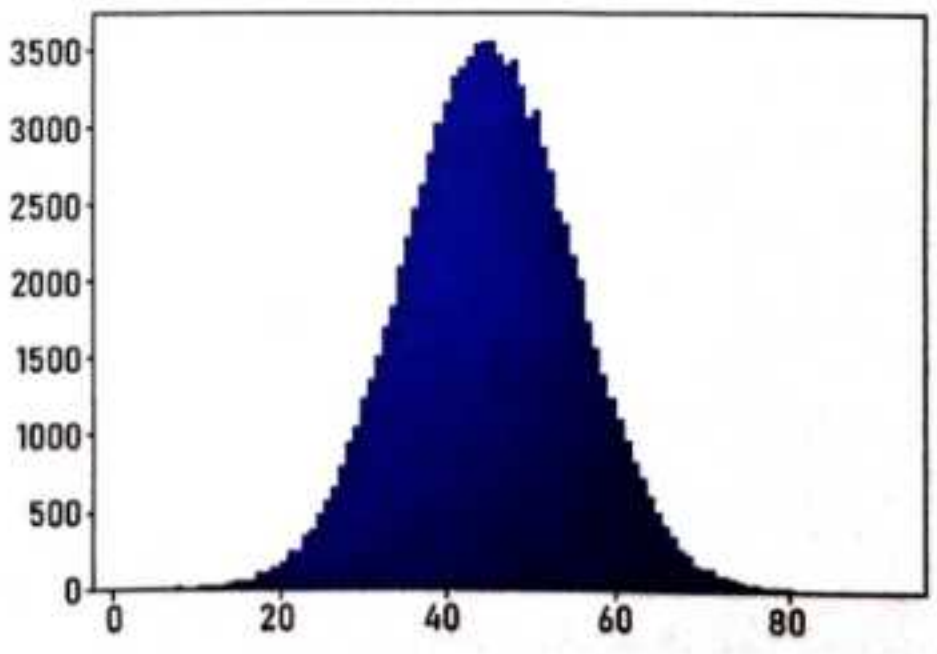
training and *test* set. The training set is used to build the predictive model, and the test set is used to get an unbiased estimate of the accuracy of the model. For more details, refer to Chapter 6. Creating these two subsets require sampling to be used.

The most important part of generating a sample is to ensure that the sample is representative of the population which it is supposed to represent. If it is not representative, then the conclusions we will draw from the sample will not be applicable to the population.

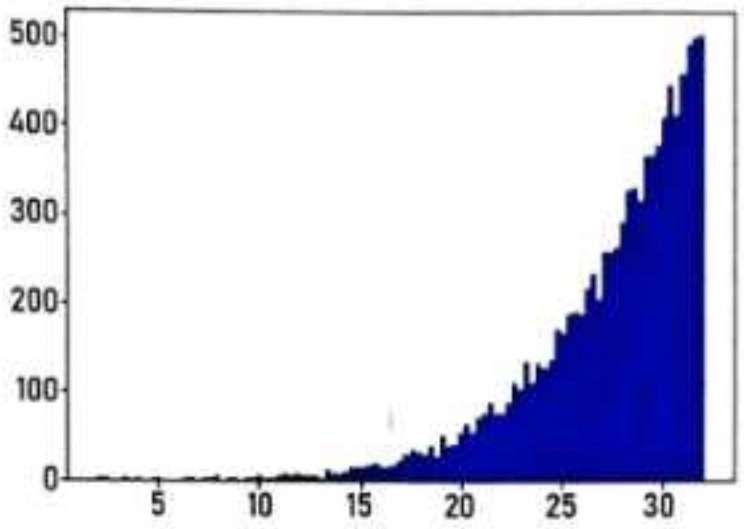
Suppose we have a population which follows the normal distribution as shown in this section. To derive a sample which has 10% the size of the population, we sort the values and then create a sample from the first 10% of the values. The distribution of the sample, printed as the following output can be seen to be very different from the distribution of the source set. It should be clear that if we studied the properties of this sample, those properties will not hold for the population.

We may be tempted to think that if we had not sorted the dataset and then selected the first 10% as our sample, the sample would be representative. But what if our dataset was given to us sorted? What if we did not know whether the source dataset was sorted or partially sorted?

```
In [1]: ▸ import matplotlib.pyplot as plt
          %matplotlib inline
          import numpy as np

          xn = np.random.normal(45, 10, 100000)
          plt.hist(xn, bins=100)
          plt.show()
```



```
In [11]: ▸ xnsrtd = np.sort(xn)
          xnsrtds = xns[ 0:10000]
          plt.hist(xnsrtds, bins=100)
          plt.show()
```



APPENDIX E.8.1 RANDOM SAMPLING

To generate a representative sample, a technique called random sampling is commonly used.

Random Number

A random number is a number generated by a process, such that it is not possible to predict what the next generated number would be. If the random numbers are in a specified range, say [0 to 10^{10}], then another way to view the random number generation process is that as it generates numbers, the next number is equally likely to be any one of the numbers in the given range.

Random Sampling

In random sampling, each element of the population is equally likely to be selected. If we select a sufficiently large number of elements using random sampling from a given population, the subset of these elements is very likely (but not guaranteed) to be representative of the initial population. There are two aspects that need to be understood in this context. One is the importance of random sampling. Second is the point that for the sample to be representative, it needs to be sufficiently large.

Why is the size of the sample important?

Let us say that we want to study the weights of an adult population and which are distributed as in Figure E.8.1(a). The sample in Figure E.8.1(b) is not a representative sample since it does not contain all the unique values in the population. When there are a large set of unique values, it may not be possible to ensure that all the values are included in the sample, but one should strive to include most of them in the sample. This will set a lower limit on the size of the sample. In our case, using this criterion, our sample size has to be at least 3.

The sample in Figure E.8.1(c) has all the unique values, but if we were to study the proportion of people with each weight from the sample, we get the impression that 5 out of 8 people weigh 40, that is 62.5% of people weigh 40 kilograms. However, if we look at our original population, then we see that 1000 out of 6300 people which is roughly 16% of the population weigh 40 kilograms. So our sample is misleading. For our sample to be perfectly representative, we need a minimum 10, 45, and 8 people in each of the weight categories (or a multiple thereof) as shown in Figure E.8.1(d). This requires our sample size to be a minimum of 63.

Weight	Population Count
40	1000
50	4500
60	800

(a)

Weight	Population Count
40	1
50	1
60	-

(b)

Weight	Population Count
40	5
50	2
60	1

(c)

Weight	Population Count
40	10
50	45
60	8

(d)

Weight	Population Count
40	5
50	23
60	3

(e)

Figure E.8.1 (a), (b), (c), (d), (e) Different samples of weights of an adult population - with different distributions

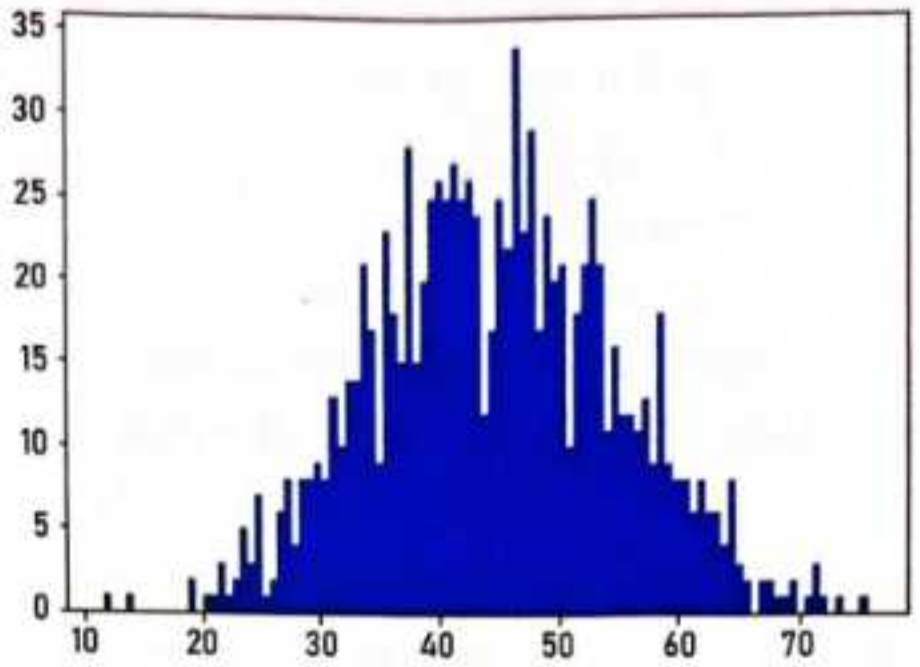
A sample which closely represents our population is shown in Figure E.8.1(e). Here, the proportion is roughly the same as in Figure E.8.1(a), though not exactly same.

So, we see that a sample which is too small may be inadequate to represent the population faithfully. It can be shown that as the size of the sample is increased; it begins to represent the population more closely. There are statistical methods to determine a good sample size. However, that discussion is beyond the scope of this book.

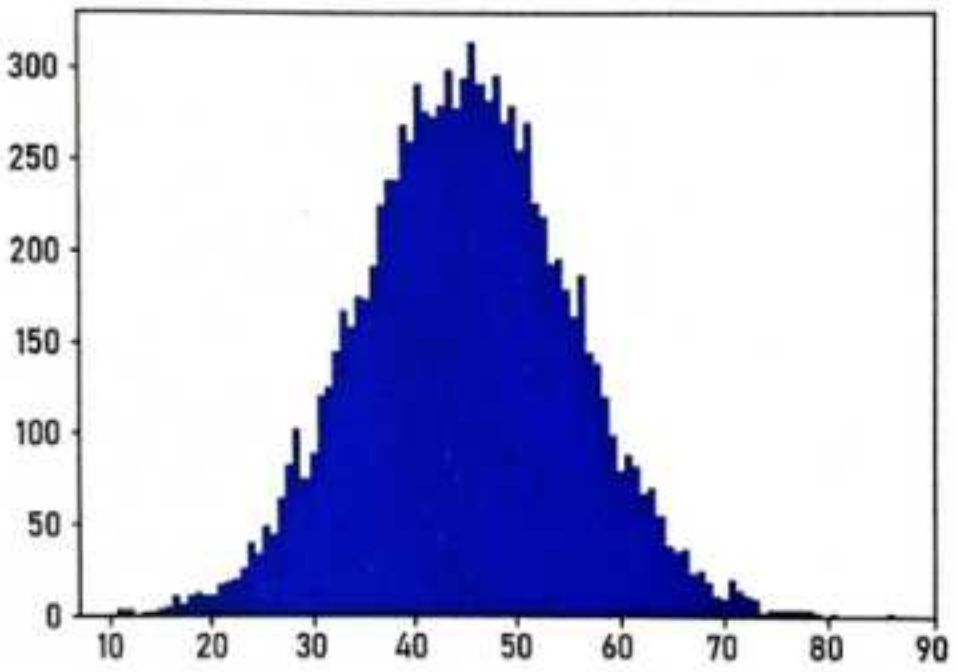
Why is random sampling important? As noted, in general we do not know how the data given to us is arranged. It could be sorted if not fully, possibly partially, or there could be other patterns in the way the data is arranged. Consider a hypothetical example of images of cows, cats, and dogs and there are an equal number of each. We want a 10% sample of the images. So we decide to select every 10th image, *i.e.*, the 1st one, the 11th one, the 21st one, and so on. But due to the way the data was collected and arranged, every 10th image is that of a cow. So, our sample may contain images of mostly cows and would be representative. Random sampling increases the chances, though it does not guarantee, that the sample will be representative by breaking any patterns that may exist in the arrangement of the data.

Random sampling together with a large sample size usually provides us with a representative sample. We see two random samples in this section, both generated from the sorted normal distribution of Section E.8. In the first sample, we collect only 1% of the points from the population. The sample does show a similar distribution as the population, but it does not have the smoothness, indicating that the count of values in different sub-ranges of the sample are not proportional to the counts in the original population. The second sample has 10% of the points in the population. The distribution appears very similar to the original distribution and is fairly smooth.

In [14]:
```
import random

nrandSamp = random.sample(list(xnsrtd), 1000)

plt.hist(nrandSamp, bins=100)
plt.show()
```




```
In [15]: ▸ nrandSamp = random.sample(list(xnsrtd), 10000)
           plt.hist(nrandSamp, bins=100)
           plt.show()
```



Pseudo-random number generator

Most computer-generated numbers are not really random. They appear random, as the sequence of numbers generated do not seem to have a predictable pattern when observed for short lengths. However, if they are observed for larger lengths, then the pattern starts repeating. Thus, these generators are called pseudo-random. Random sampling in computer programs is usually based on pseudo-random number generators. They work well for most practical purposes, as long as a sufficiently large sample size is selected.

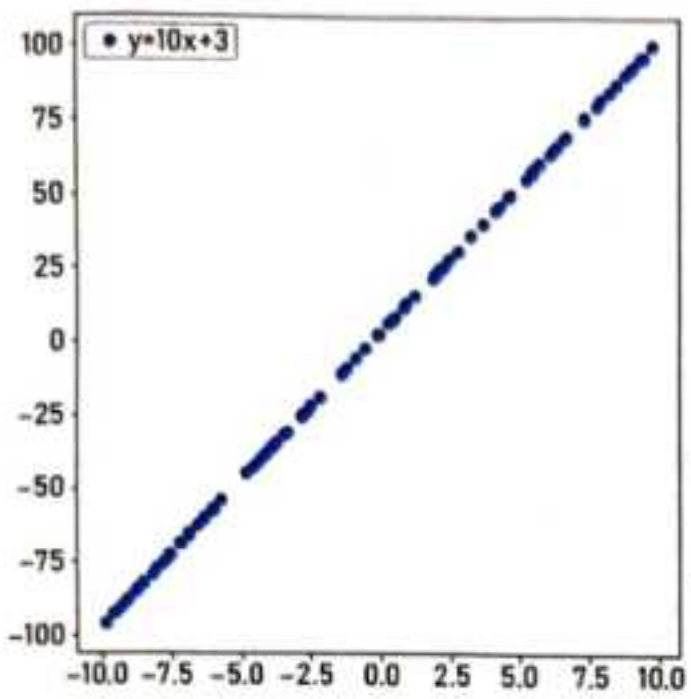
Random seed

Suppose we are using a function which is a random number generator in our program. These functions are written in such a way that if we invoke the program twice, the random number produced may not be the same between the invocations. This can change the result of the program, even though slightly, from one invocation to the next, all inputs remaining unchanged. Usually, this is not desirable since it makes it difficult to compare the results of the program from one invocation to the next. There is a way of specifying the initial or starting state of the pseudo-random number generator through a seed variable. If the seed variable is fixed, then the sequence of numbers generated, though pseudo-random, remain fixed between invocations. The same method is used for controlling any function which uses pseudo-random processes and not just random number generators.

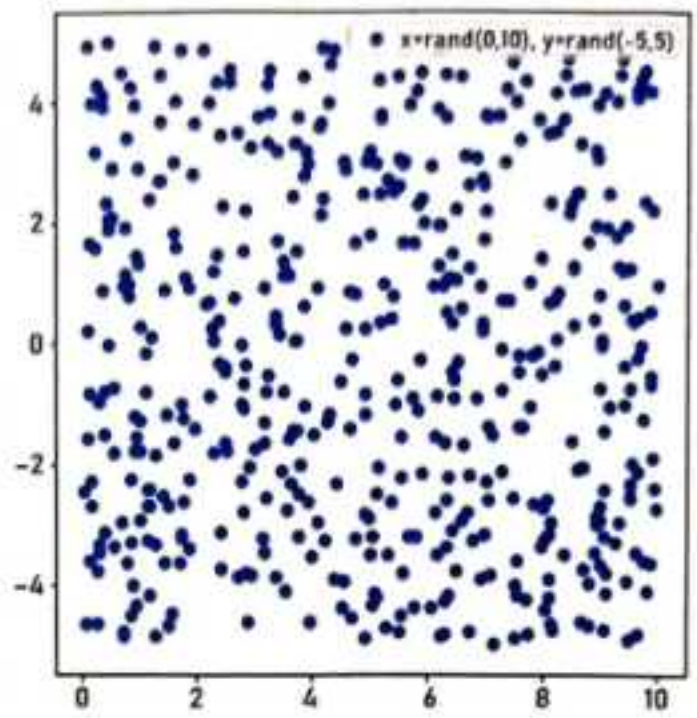
Random sampling is not appropriate in all situations. This is especially true for time series data. The reader is asked to exercise caution if the need for sampling time series data arises. The method used for sampling will be application dependent.

APPENDIX E.9 CORRELATION

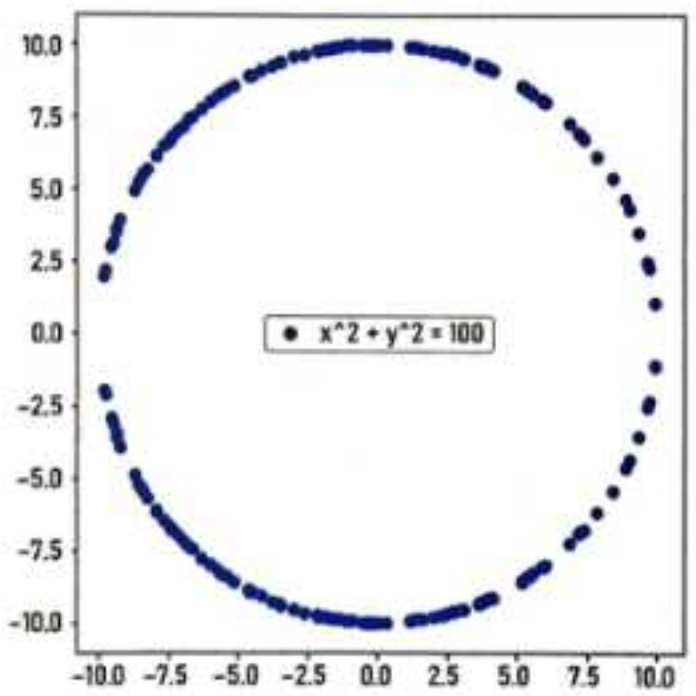
While solving problems of machine learning, one of the intrinsic goals is to find relationships between variables. Correlations are a means to measure the relationship between variables. Consider the graph shown in Figure E.9.1(a). It represents points from the straight line y = 10*x + 3. There is thus a relationship between x and y specified by the equation and we term such relationships as linear. Consider the graph shown in Figure E.9.1 (b). x and y have been generated randomly from the range (0, 10) and (–5, 5). There is thus, no relationship between them and the lack of relationship can be seen as a set of points with no apparent pattern in them. The third plot in Figure E.9.1(c) corresponds to the equation $x^2 + y^2 = 100$, the equation of a circle. Thus, there is a relationship between the x and y variables which is obvious from the plot, but the relationship is not *linear*.



(a)



(b)



(c)

Figure E.9.1 (*a*) A plot of a linear relationship. (*b*) A plot with no apparent relationship. (*c*) A plot where the relationship is defined by a circle.

Trying to visually ascertain relationships is not always feasible. Also, in most real-life situations, one will not find plots which are as perfect as in E.9.1.1(a) or (c). Hence, we often use numerical measures of the correlation between variables. The most common measure used is called the Pearson's correlation coefficient, given below in Equation E.9.1. An intuition behind the working of the equation is given in the Section 14.4.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \quad \textit{Equation } \textbf{E.9.1}$$

The PCC takes on values between –1 and 1 and measures the linearity of a relationship. The value of 1 corresponds to a perfect positive linear relationship, where when *x* increases, *y* also increases. The value of –1 corresponds to a perfect negative linear relationship, where when *x* increases, *y* decreases and vice versa.

A set of plots and their PCC values (*Source:* Wikipedia article on Pearson correlation coefficient) are shown below in Figure E.9.2. It can be seen that the plots in the last row show clear relationship between the *x* and *y* variables. However, the PCC value for them is 0. This is an illustration of the point that PCC is not good for measuring strength of relationships which are not collinear.

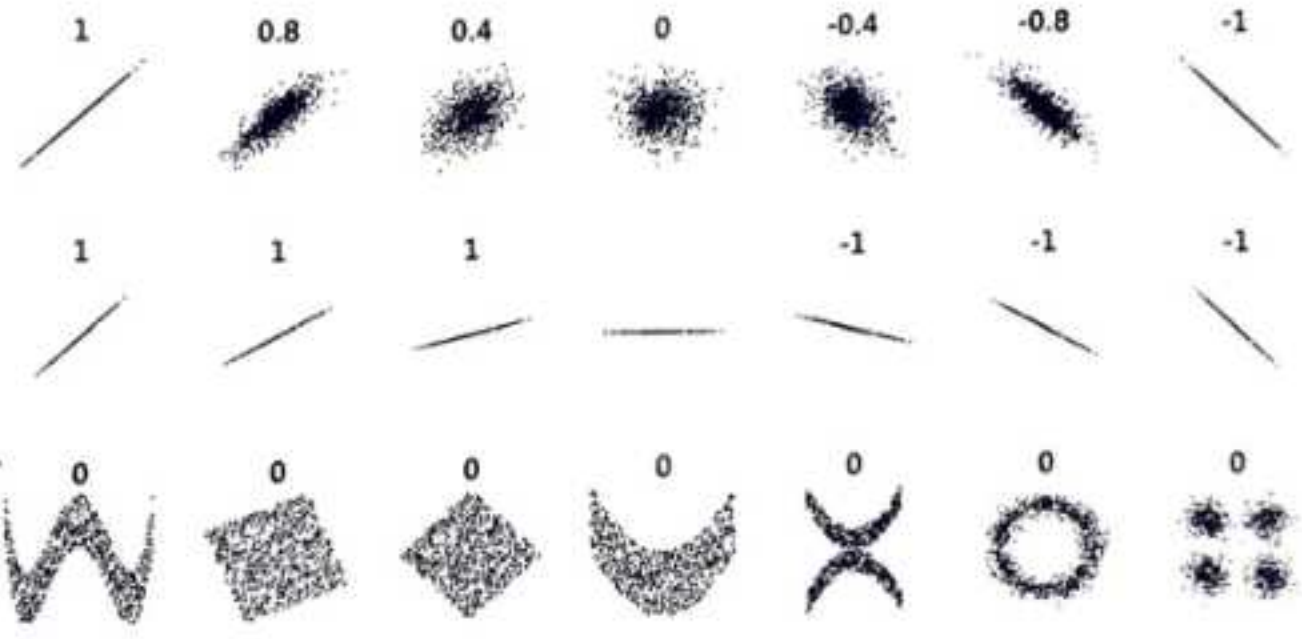


Figure E.9.2 Plots showing different relationships and their Pearson Correlation Coefficient

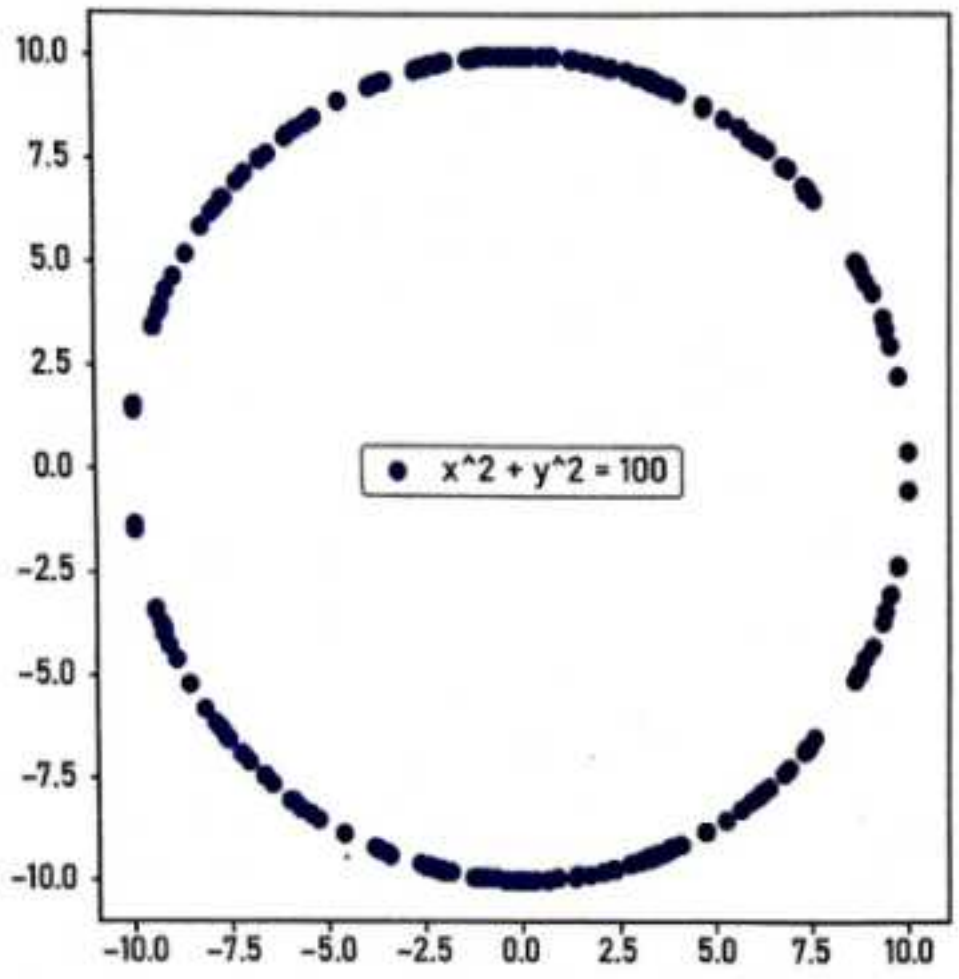
We provide the complete code for generating a set of points which are on a circle and then measuring their PCC below. The PCC below can be seen to be 0, indicating that there is no linear relationship between the two variables.

In [55]:
```
# We want a circle. x^2 + y^2 = 100.
x = np.random.uniform(-10, 10, 100)
yplus = np.sqrt( 100 – x*x)
yminus = -np.sqrt( 100 - x*x)
```

```
X = list(x) + list(x)
Y = list(yplus) + list(yminus)

plt.figure(figsize=(6, 6))
plt.scatter(X, Y, label='x^2 + y^2 = 100')
plt.legend(loc= 'center' )
plt.show()
```

In [56]:
```
(corr, _) = stats.pearsonr(X, Y)
print("PCC:", corr)
```

```
PCC: 0.0
```



APPENDIX E EXERCISES

Review Exercises

1. What is a sample space of an experiment?
2. Define probability in terms of events and outcomes of an experiment.

3. Suppose there are two events E1 and E2. When E1 happens, E2 cannot happen, and vice versa. Express the probability that either E1 or E2 will happen in terms of the probabilities of E1 and E2.
4. Suppose the probability of an event E1 is 1/3. What is the probability of the complementary event to E1?
5. What is the difference between a discrete and a categorical variable?
6. What is a histogram? What are histograms useful for?
7. Why is a median often a better measure of central tendency than a mean?
8. Which measure of central tendency is more robust in the presence of outliers? Why?
9. Give the formula for standard deviation of a distribution. What does standard deviation measure?
10. Define Interquartile range.
11. What are boxplots? How do they represent outliers?
12. Why is the normal distribution so important? How can a normal distribution be characterized?
13. What is the characteristic of a normal distribution with respect to its standard deviation?
14. Define a uniform distribution.
15. Why is random sampling an important technique for generating a sample of a distribution?
16. What can be done to increase the chances of a random sample to be more representative of its parent distribution?
17. What is a seed in the context of random number generator?
18. Why is correlation between variables important?
19. Give the formula for Pearson's correlation coefficient. What kind of correlation does it measure?
20. What is approximate value of Pearson's coefficient for points which lie on a circle? What does this value imply?

Investigative Exercises

1. Random sampling works well in generating a representative sample when the size of the sample is relatively large, as discussed in one of the preceding

sections. Suppose you have a dataset which has data belonging to two classes, a dog class and a cat class. There are 10,000 samples belonging to the dog class, but only 200 belonging to the cat class. There is a significant probability that a random sample will cause the cat class to be even more under-represented in the sample. What trick can be used to improve the chances of the cat class being properly represented in a random sample?

Supporting Material

https://www.aspiration.ai/machine-learning/Tests.jsp#statistics-and-probability