**SUPERVISED MACHINE LEARNING**



BY

DR MYOTHIDA

*This book is dedicated to my students over the years and all the youths.*

# Contents

[Contents](#_bookmark0) 3

1. [Introduction to Machine Learning](#_bookmark1) 14
   1. [What is Machine Learning?](#_bookmark2) 15
   2. [Types of Machine Learning](#_bookmark3) 17
      1. [Supervised Machine Learning](#_bookmark4) 17
      2. [Unsupervised Machine](#_bookmark5) [Learning](#_bookmark5) 18
      3. [Reinforcement Learning](#_bookmark6) 20
   3. [Introduction to Python Environment](#_bookmark7) 21
      1. [Getting started with Google](#_bookmark8) [Colab](#_bookmark8) 22
   4. [Knowledge Test](#_bookmark13) 26
      1. [Supervised or Unsupervised?](#_bookmark14) 26
      2. [Regression, Classification or](#_bookmark15) [Clustering](#_bookmark15) 27
2. [Regression](#_bookmark16) 28
   1. [Linear Regression](#_bookmark17) 31
      1. [Simple Linear Regression](#_bookmark18) 31
      2. [Multiple Linear Regression](#_bookmark22) . 35
      3. [Feature Scaling](#_bookmark23) 36
      4. [Assumptions](#_bookmark24) 38
      5. [Knowledge Test](#_bookmark25) 40
   2. [Gradient Descent Method](#_bookmark26) 42
      1. [Knowledge Test](#_bookmark28) 45
   3. [Polynomial Regression](#_bookmark29) 46
      1. [Multiple Features](#_bookmark31) 48
      2. [Implementation](#_bookmark32) 49
      3. [Hyper-parameter](#_bookmark34) 50
      4. [Knowledge Test](#_bookmark35) 51
   4. [Model Implementation](#_bookmark37) 52
      1. [Data Preparation](#_bookmark38) 52
      2. [Data Splitting: Train-Test Split](#_bookmark39) 53
      3. [Data Modelling](#_bookmark40) 56
   5. [Performance Evaluation](#_bookmark42) 60
      1. [Mean Absolute Error](#_bookmark43) 60
      2. [Mean Squared Error](#_bookmark44) 60
      3. [R2-score](#_bookmark45) 61
      4. [Implementation using Python](#_bookmark46) 64
      5. [Cross-Validation](#_bookmark47) 69
      6. [Bias and Variance Trade-off](#_bookmark48) . 71
   6. [Regularization](#_bookmark50) 74
      1. [Lasso Regression](#_bookmark51) 74
      2. [Ridge Regression](#_bookmark52) 75
      3. [Lasso vs. Ridge](#_bookmark53) 76
   7. [Hyper-Parameter Tuning](#_bookmark55) 78
   8. [Project: Sale Amount Prediction](#_bookmark56) 80
      1. [Data Importing](#_bookmark57) 80
      2. [Train-Test-Split](#_bookmark58) 81
      3. [Modeling and Hyper-](#_bookmark59) [Parameter Tuning](#_bookmark59) 82
      4. [Model Evaluation](#_bookmark60) 83
3. [Classification](#_bookmark61) 85
   1. [Performance Evaluation](#_bookmark62) 88
      1. [Classification Performance](#_bookmark63) . 88
      2. [Confusion Matrix](#_bookmark64) 88
      3. [Accuracy](#_bookmark67) 92
      4. [Precision](#_bookmark69) 94
      5. [Recall (Sensitivity)](#_bookmark70) 95
      6. [True Negative Rate (Specificity)](#_bookmark71) 95
      7. [F-score](#_bookmark72) 96
      8. [Area under the ROC (AUROC](#_bookmark73)

[or AUC)](#_bookmark73) 96

* 1. [Logistic Regression](#_bookmark76) 99
     1. [Logistic Regression in Python](#_bookmark80) 102
     2. [Hyper-parameters](#_bookmark81) 104
  2. [K Nearest Neighbors or k-NN](#_bookmark82) 105
     1. [Distance Metrics in k-NN](#_bookmark84) 107
     2. [Values of k in k-NN](#_bookmark85) 108
     3. [K-NN in Python](#_bookmark86) 108
  3. [Support Vector Machine](#_bookmark87) 110
     1. [Hyperplane](#_bookmark88) 110
     2. [Decision boundary](#_bookmark89) 110
     3. [Support Vectors](#_bookmark91) 112
     4. [Margin](#_bookmark93) 113
     5. [SVM Kernels](#_bookmark94) 113
     6. [Hyper-parameters](#_bookmark97) 118
     7. [Linear SVM in Python](#_bookmark98) 120
     8. [Polynomial Kernel SVM in](#_bookmark99) [Python](#_bookmark99) 122
     9. [RBF Kernel SVM in Python](#_bookmark100) . 124
  4. [Project: Fraud Detection](#_bookmark101) 126
     1. [Data Importing](#_bookmark102) 127
     2. [Splitting the data](#_bookmark103) 127
     3. [Modelling](#_bookmark104) 128
     4. [Model Evaluation](#_bookmark106) 131
     5. [Discussion on Performance](#_bookmark107) . 133

1. [Further Reading](#_bookmark109) 136
   1. [Mathematic Topics recommended by](#_bookmark110) [ChatGPT](#_bookmark110) 138
   2. [Courses recommended by ChatGPT](#_bookmark111) 140
   3. [Books recommended by ChatGPT](#_bookmark112) . 141

[Bibliography](#_bookmark113) 144

*စာရေးသူ၏ အမှာစာ*

*ယခု စာအုပ်သည် ချင်းမိုင် တက္ကသိုလ်တွင် လက်ရှိ သင်ကြားလျက် ရှိသော* **‘Introduction to Supervised Machine Learning’ ဘာသာရပ်ကို အခြေခံ၍ ပြုစုထားသော စာအုပ်တစ်အုပ် ဖြစ်ပါသည်။**

**စာရေးသူမှာ စင်ကာပူနိုင်ငံ နန်ယန်း တက္ကသိုလ်တွင် အင်ဂျင်နီယာ ဘွဲ့ကြိုသင်တန်းအား ၂၀၀၁ မှ ၂၀၀၅ ခုနှစ်ထိ တတ်ရောက်ခဲ့ပါသည်။ ထိုဘွဲ့ကြိုသင်တန်းကာလအတွင်း ပြုလုပ်ခဲ့သည့် "Literature Review on Face Recognition Methods" ဟုခေါ်ဆိုသည့် စာတမ်းငယ် မှာ စာရေးသူ၏ ပထမဦးဆုံး မှတ်ဥာဏ်အတု (Artificial Intelligence or AI) နှင့် ပတ်သတ်သည့် သုတေသန လုပ်ငန်းစဥ် တစ်ခု ဖြစ်ခဲ့ပါသည်။ ထိုသုတေသနမှ အစပြု၍ computer vision and machine learning ကို ပိုမို စိတ်၀င်စားလာကာ မာစတာနှင့် ဒေါက်တာ ဘွဲ့များကို ဆက်လက် ရယူခဲ့ပါသည်။ ၂၀၀၃ ခုနှစ် မှ ၂၀၁၃ ခုနှစ် ကာလ များတွင်လည်း သုတေသန စာတမ်းငယ်များစွာကို ထုတ်၀ေခဲ့ပြီး ဒေါက်တာ ဘွဲ့အတွက် တင်သွင်းခဲ့သည့် သုတေသန စာတမ်းကို ပြင်ဆင်၍ "Contextual Analysis of Videos". ခေါင်းစဥ်ဖြင့် စာအုပ်တအုပ်ကိုလည်း ရေးသား ထုတ်၀ေခဲ့ပါ‌ သေးသည်။အဆိုပါ စာအုပ်မှာ ယခုတိုင် ရောင်းချနေရဆဲ ဖြစ်ပြီး amazon website တွင် ၀ယ်ယူဖတ်ရှု့နိုင်ပါသည်။**

အင်္ဂလိပ်ဘာသာဖြင့် နည်းပညာစာအုပ်များ စာစောင်များစွာ ရေးသား ထုတ်၀ေခဲ့သော်လည်း နည်းပညာဆိုင်ရာ စာအုပ်များကို မြန်မာ ဘာသာ အသုံးအနှုန်းဖြင့်ရေးသားရန် ယုံကြည်မှုအားနည်းခဲ့ပါသည်။ သို့သော် မြန်မာနိုင်ငံသို့ ပြန်ရောက်ပြီးနောက် လူငယ်များကို computer vision and machine learning ဆိုင်ရာ ဘာသာရပ်များကို သင်ကြားဖြစ်ရင်းမှ လိုအပ်ချက်များကို တွေ့မြင်လာရသည့်အတွက် ၂၀၁၇ /၂၀၁၈ ကာလတွင် "Introduction to MATLAB: Learning by Doing” ခေါင်းစဥ်ဖြင့် ပထမ ဦးဆုံး စာအုပ်ကို ထုတ်၀ေဖြစ်ခဲ့ပါသည်။ မြန်မာစာလုံးပေါင်း သတ်ပုံများ အမှား အယွင်းများစွာ ပါရှိခဲ့သော်လည်း နည်းပညာ စာအုပ်ဖြစ်သည့်အတွက် ဖတ်ရှု့သူ များပြားခဲ့ပါသည်။ စာအုပ် စောင်ရေ ၁၀၀၀ နီးပါး ရောင်းချခဲ့ရပြီး စာဖတ်သူ အများစုမှ အကျိုး ရှိသည်ဟု ပြန်လည် ပြောကြားမှုများလက်ခံရရှိခဲ့ရာမှ Introduction to Machine Learning စာအုပ်ကို ရေးသားရန် ခွန်အားများ ရရှိခဲ့ပါသည်။ ၂၀၁၉ ကာလတွင် စတင်ရေးသားခဲ့သော်လည်း အခြားလုပ်ငန်းများဖြင့် အချိန်‌ပေးနိုင်ရန်ခက်ခဲခဲ့ရ‌သောကြောင့် အဆုံးသတ် နိုင်ခဲ့ခြင်း မရှိပါ။

ယခုအခါ AI, machine learning ဆိုင်ရာ နည်းပညာ စာအုပ်များ၊ သင်တန်းများ အလွယ်တကူ ရရှိနိုင်ပြီ ဖြစ်သော်လည်း အင်တာနက် ခက်ခဲသော ဒေသမှ လူငယ်များ၊ အင်္ဂလိပ်စာ ဘာသာစကား အားနည်းသော လူငယ်များ အတွက် လေ့လာရန် ခက်ခဲနေဆဲ ဖြစ်သည်ကို တွေ့မြင်‌နေရသည့်အတွက် အဆိုပါ လူငယ်များအတွက် ရည်ရွယ်၍ ‘Introduction to Supervised Machine Learning’ (Supervised Machine Learning မိတ်ဆက်) စာအုပ်ကို အင်္ဂလိပ်-မြန်မာ နှစ်ဘာသာဖြင့် ရေးသားပြုစုထားပါသည်။

Supervised Machine Learning မိတ်ဆက် စာအုပ်တွင် အခန်း ပေါင်း ၄ ခန်း ပါ၀င်ပြီး ပထမ အခန်းတွင် Machine Learning နှင့် ပတ်သတ် သည့် ယေဘူယျ သိသင့်သည်များကို မိတ်ဆက်ထားပါသည်။ ဒုတိယ အခန်းတွင် regression method နှင့် ပတ်သတ်၍ ရှင်းလင်းထားပြီး တတိယ အခန်းတွင် classification method အချို့ကို ရှင်းလင်း တင်ပြထားပါသည်။ စတ္ထုတ္ထ အခန်းတွင်မူ ဆက်လက် လေ့လာသင့်သည့် သင်တန်းများ၊ စာအုပ်များကို ညွှန်းဆိုထားပါသည်။

မြန်မာ စကား အားနည်းချက်ကြောင့် စာလုံးပေါင်း သတ်ပုံ အမှားအယွင်းများ ရှိပါက သည်းခံ နားလည်ပေးပါရန် ကြိုတင် မေတ္တာရပ်ခံအပ်ပါသည်။

solely on "*Supervised Machine Learning Methods*" and to write the book in both Myanmar and English languages, so that it could be useful for my students from Bhutan and Thailand.

The purpose of this book is to document my teach- ings at Chiang Mai University in a physical form and make it accessible for students with limited re- sources to learn from. The book aims to provide a comprehensive and easy-to-follow introduction to the fundamental concepts of machine learning methods. It is divided into four parts.

The first chapter provides an overview of the ba- sic questions of machine learning and introduces the Python development environment. The second chapter covers various regression methods and the third chapter discusses different classification meth- ods. The last chapter provides recommendations for continuing the journey of learning machine learn- ing. I believe that hands-on learning is crucial for un- derstanding and thus, the explanations in the book are accompanied by detailed ’Python code’ snippets throughout the text. The readers can follow the in-

structions and run the code on their own computer or an online platform such as Google Colab.

I hope that this book provies a valuable resource for underprivileged youths.

# Website

The drmyothida website contains much ad- ditional material, will be available soon at [www.drmyothida.org/courses.](http://www.drmyothida.org/courses) As students read through **Introduction to Supervised Machine Learning**, they can go online to take self-grading quizzes and access learning materials such as Pow- erPoint slides and recorded videos. The complete codes for all the projects are also available at the public [**GitHub Repo**](https://github.com/myothida/Intro-To-Supervised-Machine-Learning.git).

*Chapter 1*

# Introduction to Machine Learning

Over a decade, machine learning has become part of our daily routines. We often use search engines like Google, which utilize machine learning to pro- vide us with the information we request, and similar technology is used in map search engines to suggest the most efficient routes. However, terms such as Machine Learning, Deep Learning, Neural Network,

and Artificial Intelligence are frequently used inter- changeably, causing confusion for those new to the field. This chapter aims to clarify these terms before delving deeper into the topic of machine learning.

## What is Machine Learning?

There are many definitions of "Machine learning", some of which are complex and use technical jargon, while others are more straightforward. Essentially, machine learning methods are created by humans to train computers or machines to respond or make decisions based on data, or to extract important in- formation from data. For example, ChatGPT [[2](#_bookmark115)], a widely used AI invention as of January 2023, is able to answer questions and even write code for developers, it was trained on 570 gigabytes of text according to Stanford University. Does this infor- mation make you feel that understanding Machine Learning might be challenging for you?

Consider the way you first learned to speak. Our mothers would repeatedly show us their face and

say the word "mama" until we connected her face with the word "mama" and memorized it. This is a simple example of machine learning, specifically supervised machine learning. Similarly, a computer must be trained to perform a task, such as saying "mama", by using collected data, such as a face with or without makeup and the word "mama", until it knows how to perform the task when presented with new data, such as a different face.

## Types of Machine Learning

There are three main types of learning problems in machine learning: supervised, unsupervised, and reinforcement learning.

### Supervised Machine Learning

The example of "teaching a child to speak their first word" illustrates a supervised machine learning problem. This type of machine learning requires a pre-existing data-set with known information and desired outcomes. The machine learning model must learn from this dataset in order to predict the desired outcome.

Supervised machine learning is widely used in today’s technologies across various fields such as bio-security, weather forecasting, text prediction, and spam detection, etc. These methods use pre- existing data that includes both input features and the desired output (or target variable) to understand the underlying relationship. The model learns from the data, and the resulting parameters can be used

to predict outcomes for new data.

An example of supervised learning is a biometric authentication system that uses facial recognition or fingerprints for logging into computers or mobile phones. The system is trained to recognize an indi- vidual’s unique bio-data through registration. Once registered, the system compares presented bio-data to the registered information, and grants ("accept") or denies access ("reject") accordingly. Another ex- ample is a weather forecasting system that provides numerical data as output.

Depending on the type of output, supervised learning methods can be divided into two categories: **classification**, which assigns a class label to new data (such as "accept" or "reject") and **regression**, which predicts a numerical value (such as weather data).

* + 1. *Unsupervised Machine Learning* Unsupervised machine learning refers to the pro- cess of analyzing data without a pre-existing labeled dataset. In this case, the machine learning method

extracts information from the data on its own.

For example, imagine you are an active user of a social media page and your team has been upload- ing photos regularly for over a decade. One day, you decide to download all the photos and sort them based on similar activities. Manually checking and labeling each photo would be time-consuming as there are more than 100,000 photos. Unsupervised machine learning can be used to analyze and group the photos into different clusters based on similarity measures. Photos in the same group will be more alike than those in different groups.

To sum up, unsupervised machine learning meth- ods focus on analyzing and grouping a large dataset into different clusters, based on user-defined rules of association. Another common use of unsuper- vised machine learning is anomaly detection, where the data is divided into normal (majority) and ab- normal (outlier) groups.

### Reinforcement Learning

Reinforcement learning is a field of machine learn- ing, in which an agent learns to perform tasks by trial-and-error, while receiving feedback in form of reward signals. Referring to the example of learning our first word, we start saying "mama" after learning from our parents, but sometimes we make mistakes and our parents help correct us by giving rewards for saying the right word or scolding us when we are wrong.

Similarly, reinforcement learning methods learn by interacting with their environment and receiving positive or negative feedback. Reinforcement learn- ing is often used in training robots for navigation.

## Introduction to Python Environment

In this course, we will use Python [[6](#_bookmark119)] to develop ma- chine learning methods. Python is an open-source programming language, it is easy to learn and there are a plethora of libraries and frameworks specifi- cally designed for machine learning. There are sev- eral Integrated Development Environments (IDEs) to write Python code and Jupyter Notebook [[3](#_bookmark116)] is one of the most popular choices among machine learning practitioners. Additionally, Google Colab

[[4](#_bookmark117)] is a free cloud service offered by Google, which allows you to run Jupyter Notebooks online with- out installing it on your computer and access the computing resources provided by Google.

For the exercises discussed in this book, I use both Jupyter Notebook on the Visual Studio Code (VS code) editor [[1](#_bookmark114)] and Google Colab [[4](#_bookmark117)]. You can ac- quire the codes from online resources or write them yourself to learn. The most effective way to learn any subject is through hands-on practice.

### Getting started with Google Colab

This section will provide an overview on how to use the Google Colab platform. If you prefer a visual demonstration, you can refer to the YouTube video provided on the [**link**](https://youtu.be/9Dt8inY5lhY)[[5](#_bookmark118)] where I have explained each line of code for using the Google Colab platform.

#### Open a new Colab notebook

* + - * Goto [**Google drive**](https://drive.google.com/)and log-in using your Google Account. If you do not have one, regis- ter with Google and it is free.
      * Click on *new* > *more* > *Google Colaboratory* as shown in Figure [1.1](#_bookmark9).
      * If you do not find the *Google Colaboratory* in the list, you can click *’Connect more apps’* and search ’Google Colab’ in the search bar and add Google Colab to your list as shown in Fig- ure [1.2.](#_bookmark10)

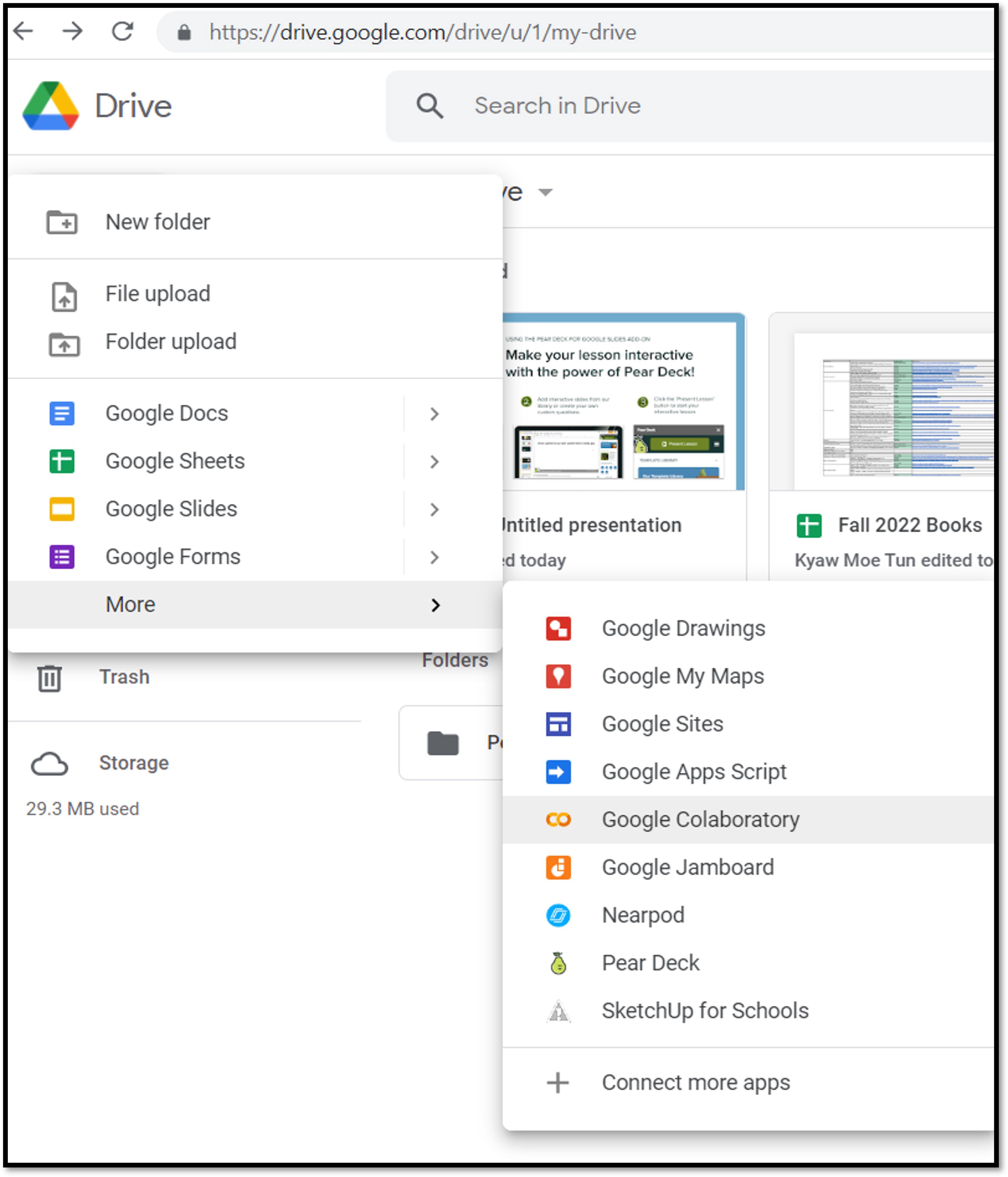


Figure 1.1: *Creating a new Colab notebook*

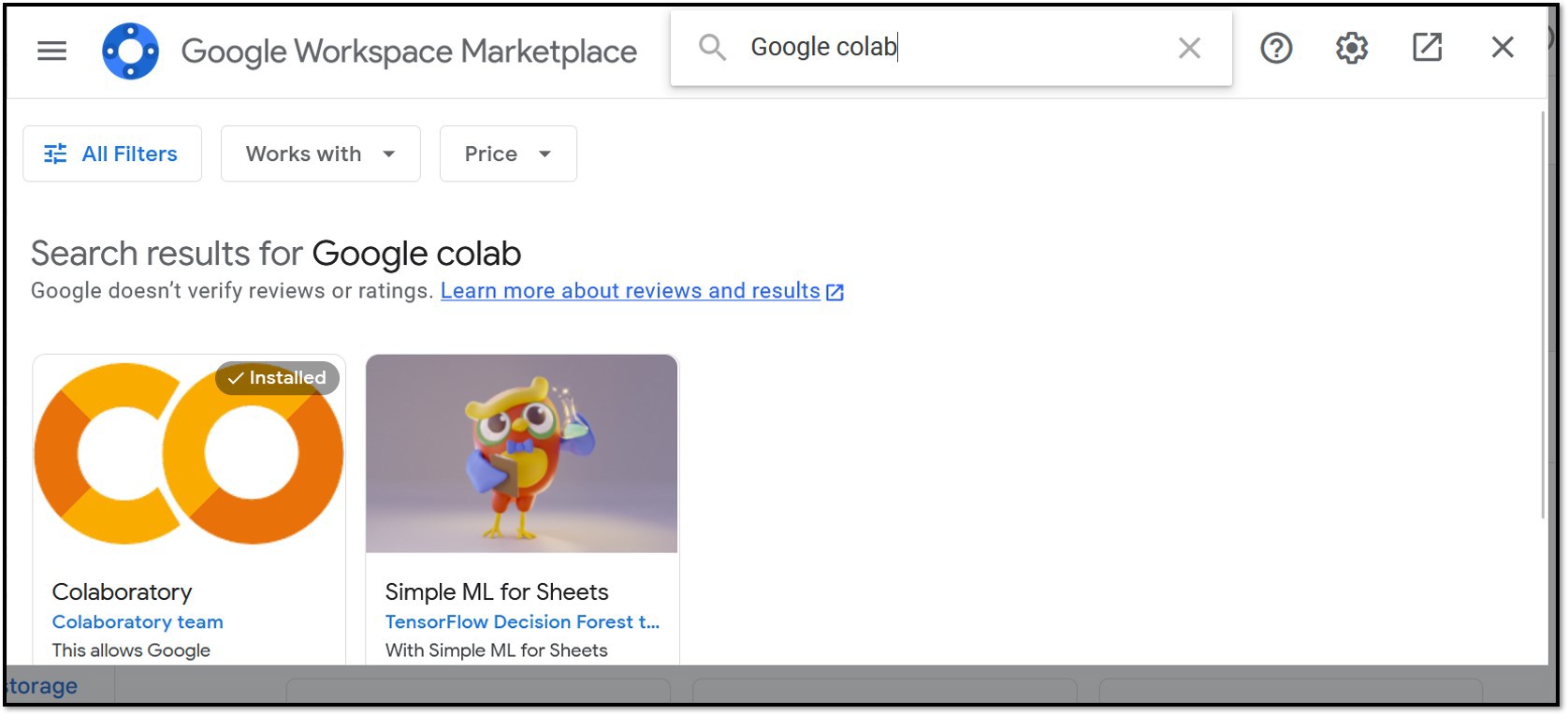


Figure 1.2: *Connecting to Google Colab App*

#### Running a Cell

* + - * Make sure the runtime is connected. The notebook shows a green check on the top right corner if it is connected.
      * Enter your code at the cursor as shown in Fig- ure [1.3.](#_bookmark11)
      * To Run cell: press Ctrl + Enter
      * To Run cell and add new cell below: Alt + Enter
      * To Run cell and goto cell below: Shift + Enter



Figure 1.3: *Running a cell*

#### Example Code

Figure [1.4](#_bookmark12) shows how to read a data file in the Google Colab. Make sure that the files are uploaded into the runtime environment as shown in Figure.

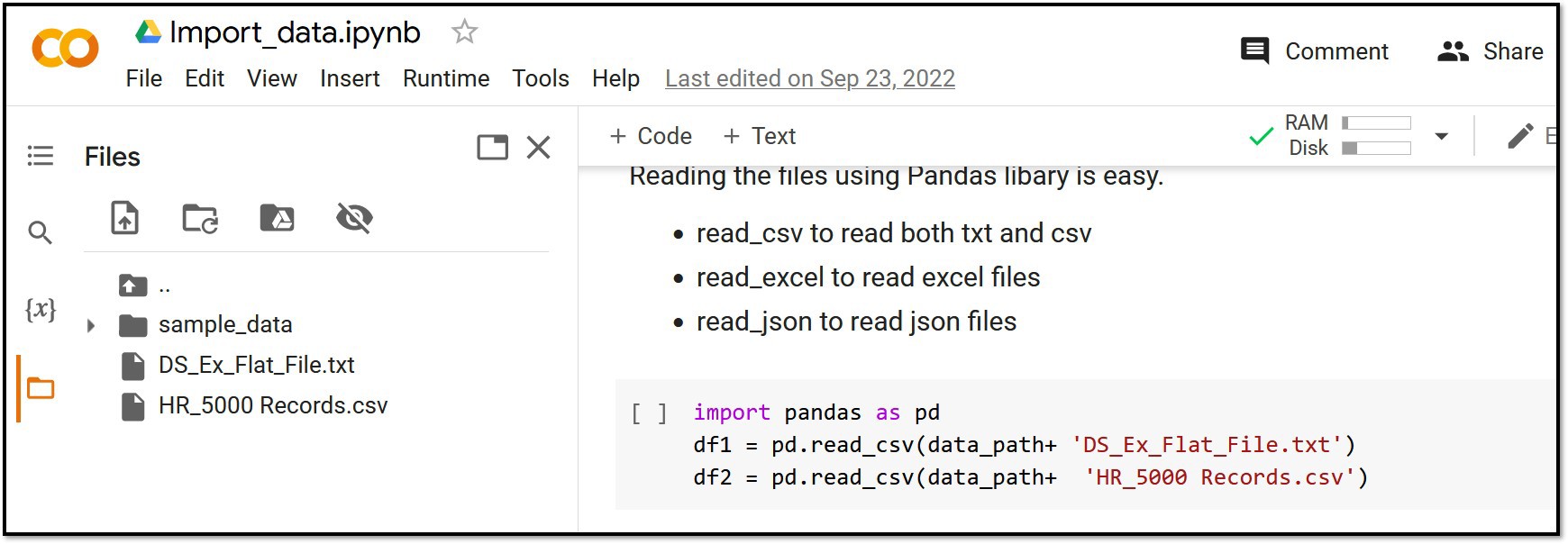


Figure 1.4: *Importing Files*

## Knowledge Test

### Supervised or Unsupervised?

1. Method to classify if a new patient is having dia- betes or not.
   1. Supervised B. Unsupervised
2. Method to cluster a set of articles into different groups based on similar stories.
   1. Supervised B. Unsupervised
3. Method to Filter if an email is spam.
   1. Supervised B. Unsupervised
4. Speech recognition and facial recognition soft- ware.
   1. Supervised B. Unsupervised
5. Segment customer data into groups in marketing environments.
   1. Supervised B. Unsupervised
6. Detect a fighting event from the CCTV camera.
   1. Supervised B. Unsupervised

### Regression, Classification or Clustering

1. Perform initial exploratory analysis on a raw data- set to understand the grouping of data points.
   1. Regression B. Classification C. Clustering
2. Method to predict the score of a basketball game.
   1. Regression B. Classification C. Clustering
3. You run an online business and you want to pre- dict how many of soft toys will sell over the next month.
   1. Regression B. Classification C. Clustering
4. You want to develop an algorithm to check the type of coins (10 baht, 5 baht , 2 or 1 baht).
   1. Regression B. Classification C. Clustering
5. Determining whether or not someone will be a defaulter of the loan.
   1. Regression B. Classification C. Clustering
6. You run an online shop. You want to group the items sold the most in different seasons.
   1. Regression B. Classification C. Clustering

*Chapter 2*

# Regression

As seen in the previous examples, regression analysis uses pre-existing data to understand the relationship between input features and the desired output. In regression anal- ysis, there are five key terms that it’s important to be familiar with.

* The term **"Training data"** refers to the pre-existing data that includes the values of the desired output and the input features that affect the output.
* The term **"Target Variable"** refers to the feature of interest. In weather prediction, for example, the

temperature would be the target variable. This variable is also sometimes referred to as the "de- pendent variable" because it is dependent on other variables.

* **"Independent Variables"** are those that are related to the target variable. The target variable often de- pends on multiple features, but each feature can have a different impact on the target variable. For example, in the case of temperature prediction, factors such as season, time, and regional geogra- phy all play a role, but each feature has a different level of influence on the temperature.
* **"Parameters**" refers to the coefficients that deter- mine the relationship between the target variable and independent variables. Training a machine learning model involves identifying these parame- ters. The number of parameters depends on the complexity of the model.
* **"Residuals"**, also known as errors, refer to the dis- crepancy between the predicted and actual values.

Consider a retail company that utilizes Facebook ads to promote their products. The company wants to de- termine the effect of increasing their ad spending on

their sales volume. In this example, the sales volume is the target variable (*y* ) and the spending amount (*x*) is an independent feature. To examine the correlation be- tween sales volume and spending amount, we first need to gather training data on past spending and sales. The relationship between spending and sales can be mod-

eled using a linear equation *y* = *ω*1*x* + *b* or a polynomial equation *y* = *ω*1 *xk* + *ω*2 *xk*−1 + *ω*3 *xk*−2 + .. + *b*. The goal of the machine learning model is to find the coefficients

(parameters) that minimize the residual values (the dif- ference between the actual and predicted values).

## Linear Regression

Linear Regression is a type of machine learning model that uses a linear equation to predict a target variable based on one or more independent variables. The goal is to find the best parameters that minimize the difference between the predicted and actual values.

It is a simple and widely used method, and can be further divided into simple and multiple linear regression based on the number of variables used.

### Simple Linear Regression

Simple Linear Regression is a type of Linear Regression which models the relationship between one independent variable (*x*) and the target variable (*y* ) using a linear equa- tion as follow:

*y* = *ω*1*x* + *b*, (2.1)

where the coefficient *ω*1 and cut off point *b* are parame- ters that determine the linear equation used to model the relationship between the target and independent vari- able. Changing these parameters *ω*1 and *b* results in dif- ferent models that can describe the relationship between the target *y* and independent variable *x*. For instance, using the *x* and *y* values in Table [2.1,](#_bookmark19) various linear equa-

tions can be formulated as below:

|  |  |  |  |
| --- | --- | --- | --- |
| *y* | = | 100*x* | (2.2) |
| *y* | = | 100*x* + 800 | (2.3) |
| *y* | = | 150*x* + 800 | (2.4) |
| *y* | = | 125*x* + 800 | (2.5) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *x* | 0 | 1 | 2 | 3 | 4 |
| *y* | 800 | 900 | 100 | 1150 | 1300 |

Table 2.1: *Example values of the target and the independent variable.*

The goal of a Simple Linear Regression model is to find the optimal parameters (*ω*1 and *b*) that can accurately predict the target variable (*y* ) for a new independent vari- able (*x*).

Figure [2.1](#_bookmark20) illustrates the data and the lines generated by the four different equations (2.2 to 2.5). The black stars represent the training data points listed in Table

2.1. It’s clear that the red line does not fit the data points well and is not a good model. The yellow and blue lines pass through some of the data points. The goal of Simple Linear Regression is to find the parameters that produce

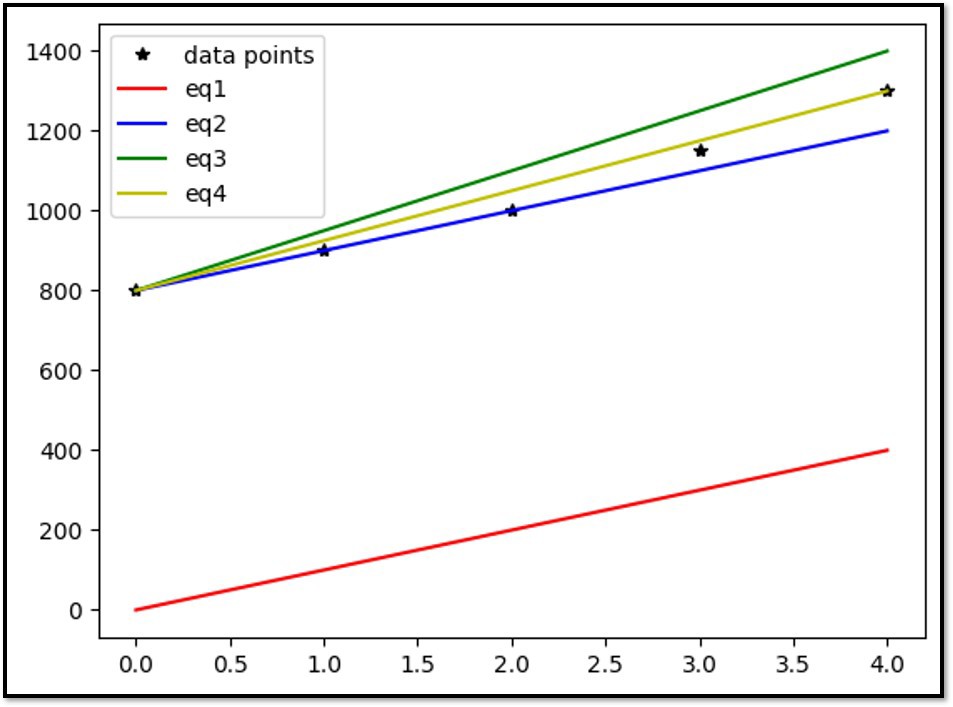


Figure 2.1: *An example of a Linear Regression Model*

a line that best describes the relationship between the target variable *y* and independent variable *x*. The best model is the one that results in the least difference be- tween the actual and predicted values.

##### Cost Function

The cost function is a measure of the average squared er- ror (or residual) between the actual and predicted values. Mathematically,

*N*

1 *i* *i*

*ε* = (*y*˜ − *y* )2, (2.6)

*N i* =1

where *N* represents the number of data points (or obser- vations) and *y*˜*i* represents the predicted value using the model, which can be described as:

*y*˜*i* = *ω*1*xi* + *b*, (2.7) where *ω*1 and *b* are model parameters.

### Multiple Linear Regression

Multiple Linear Regression is an extension of Simple Lin- ear Regression that takes into account more than one independent variable to predict the target value. This is achieved by using a linear equation, and it can be mathe- matically represented as:

*y* = *ω*1 *x*1 + *ω*2 *x*2 + *ω*3 *x*3 + ... + *b*, (2.8)

where *ωi* , *i* = 0, 1, 2, ... are parameters of the model.

Like in Simple Linear Regression, the objective of Mul- tiple Linear Regression is to find the parameters that min- imize the cost function in equation [2.6.](#_bookmark21) However, since it uses more than one feature, the different ranges of val- ues of the features can lead to one feature having more influence than the other in predicting the target value. Therefore, it is essential to scale the features to have the same or similar ranges of values.

### Feature Scaling

Feature scaling is a technique used to adjust the inde- pendent variables to a similar range of values. There are several methods for feature scaling, including min-max scaling and z-score standardization, which are among the most widely used. Min-max scaling is often used in conjunction with neural networks, while z-score stan- dardization is more frequently applied to linear regres- sion models.

##### Min-Max Scaling

Min-max scaling, also referred to as normalization, is a method for scaling data into a range of 0 to 1. It is mathematically defined as:

*xscaled*

*x* − *xmin*

= , (2.9)

*x* − *x*

*max min*

where *xmin* and *xmax* represent the minimum and maxi- mum values of the variable *x*. The minimum value of *x* will be mapped to 0 and the maximum value to 1, and all other values of *x* will be scaled accordingly.

##### Z-score Standardization

Z-score standardization, also known as zero-mean scal- ing, is a method that scales the values of each indepen- dent variable to have a mean of zero and a standard de- viation of one. It can be calculated using the following formula:

*xscaled*

*x* − *µ*

= , (2.10)

*σ*

where *µ* and *σ* are mean and standard deviation of the variable *x*.

### Assumptions

There are four underlying assumptions in linear regres- sion methods:

* + - 1. **Linear relationship**: There should be a linear re- lationship between the target *y* and independent variables *xi* . This means that a change in *y* due to a unit change in *x* should be constant. If the relationship is not linear, the linear regression may not predict the target values correctly.
      2. **No multi-collinearity**: The independent variables should not be correlated, otherwise it becomes difficult to identify how they individually influence the target value and it can inflate the standard errors of some or all of the regression coefficients.
      3. **Consistent variance of residuals**: The variance of the residuals should be consistent across all predicted values, otherwise the estimates for the model coefficients may become unreliable.
      4. **Normally distributed residuals**: The residuals should be normally distributed, meaning most of the data points should be close to one straight line and the points farther away should fall off

smoothly and symmetrically. If this is not the case, linear regression may not be a good choice for the problem.

It is important to check if these assumptions are met when considering linear regression for modeling and de- ploying.

### Knowledge Test

* + - 1. Table [2.1](#_bookmark19) shows the income earned by Mr ’Arm’ based on the number of overtime hours he worked. Represent the relationship between the overtime hours and income using the equation *y* = *ax* + *b* where *x* represents overtime hours and *y* repre- sents income. Find the values of ‘a‘ and ‘b‘ and chose the best equation. Explain your reasoning for the chosen equation.
         * *y* = 100*x* + 800
         * *y* = 125*x* + 800
         * *y* = 150*x* + 800
      2. Why is feature scaling important in multiple linear regression?
      3. What is the assumption of linearity in linear re- gression?
      4. What is multi-correlation and why is it important to avoid in linear regression?
      5. Can you explain the normality assumption in lin- ear regression?
      6. The multiple linear regression model for predict- ing sale volume is represented as *y* = 3.92*x*1 +

2.79*x*2 + 0.01*x*3 + 14 where *y* is the sale volume,

*x*1 is the amount spent on TV advertising, *x*2 is the amount spent on radio advertising, and *x*3 is the amount spent on newspaper advertising.

1. Which advertisement program has the least impact on sale volume? Can you explain your reasoning?
2. If we increase the amount spent on TV adver- tising by one dollar while keeping the other two programs constant, what will be the re- sulting sale volume? Can you explain your reasoning?

## Gradient Descent Method

The Gradient Descent method is an optimization algo- rithm that finds the values of parameters (coefficients) that minimize a cost function. In the context of simple linear regression, the cost function depends on two pa- rameters (*ω*1 and *b*) and can be represented in terms of the parameter *θ* as:

*N*

*J* (*θ*) = 1 ((*ω x* + *b*) − *y* ) , (2.11)

2

1 *i* *i*

*N*

*i* =1

where *θ* ∈ (*ω*1, *b*). To find the minimum of this func- tion, we first compute the gradient, which measures the change in all weights in relation to the change in error. The changes of *J* (*θ*) with respect to *ω*1 and *b* are given below as:

*∂J* (*θ*)

= 2*xi* ((*ω*1 *xi* + *b*) − *yi* ) (2.12)

*∂ω*1

*N i* =1

*N*

*N*

*N* 1 *i* *i*

*∂J* (*θ*)

*∂b*

= 2 ((*ω x* + *b*) − *y* ) (2.13)

*i* =1

The implementation of the Gradient Descent method can be broken down into four steps:

**Step 1** Initialize the parameters *θk* .

**Step 2** Compute the cost function value *Jk* (*θ*) using the parameters *θk* .

**Step 3** Update the parameters :

*θk*+1 = *θk* − *α*

*d J* (*θ*) (2.14)

*dθ*

where the parameter *α* is a learning rate.

**Step 4** Repeat steps 2 and 3 until the changes in cost function values are very small, or for a pre-defined number of iterations.

The performance of gradient descent is affected by the initial values of the parameters and the size of the learn- ing rate. A high learning rate may prevent the algorithm from finding the local minimum and reaching the best solution, while a low learning rate may extend the time needed for the algorithm to reach convergence.

Figure [2.2](#_bookmark27) illustrates the operation of the gradient de- scent method. The parameter, *ω*1 and *b* are initially set to

0.24 and 0.9 respectively. The gradient descent method

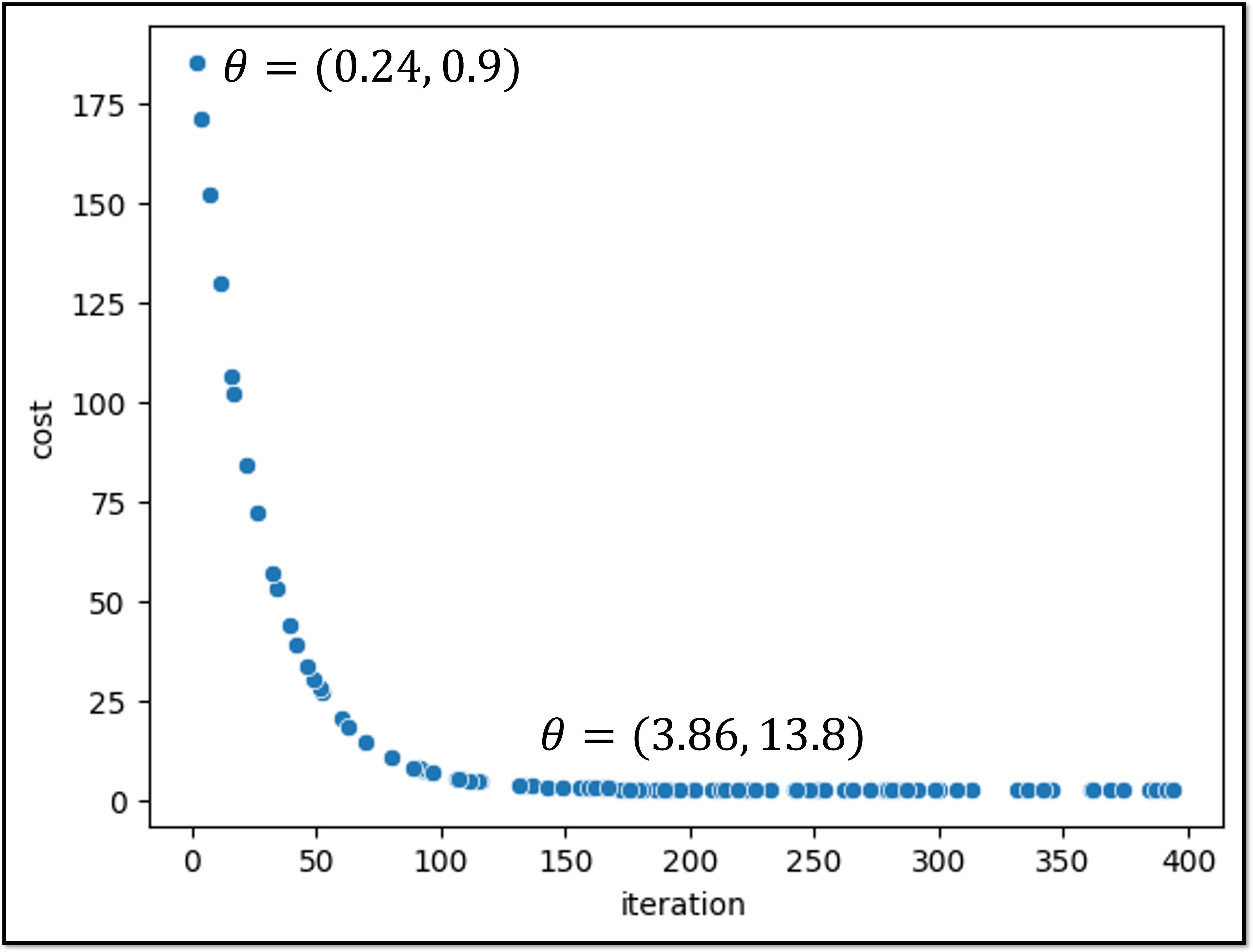


Figure 2.2: *How does the Gradient Decent Method work?*

then repeatedly modifies the parameters until the cost function reaches convergence. In this example, the cost function reaches a minimum value of 2.85 at iteration 200, when the values of the parameters are *ω*1 = 3.86 and *b* = 13.8. Beyond iteration 200, the cost function does not change significantly, indicating that these values of the parameters provide the optimal solution with the small- est difference between the actual and predicted values.

### Knowledge Test

* + - 1. Can you explain the concept of the learning rate in the gradient descent method?
      2. What is the effect of feature scaling on the gradient descent method?
      3. Figure [2.2](#_bookmark27) shows the values of cost function re- sulted by running gradient descent for 400 itera- tions with *α* = 0.01. The graph shows that the cost function, *J* (*θ*) decreases rapidly at first and then levels off.
         * what do you think will happen to the cost function if the learning rate is increased to 1 (*α* = 1)?
         * what do you think will happen to the cost function if we decrease the value of the learn- ing rate to 0.001 (*α* = 0.001)?

## Polynomial Regression

A polynomial regression model represents the associa- tion between a target variable and one or more indepen- dent variables by means of an equation of the *nth* degree polynomial equation. When there is only one indepen- dent variable, *x*, the model can be formulated as

*y* = *ω*1 *xn* + *ω*2 *xn*−1 + *ω*3 *xn*−2 + ... + *ωnx* + *b* (2.15)

where *b* is the cut off point, and *ωi* where *i* ranges from 1 to *n* are coefficients of the independent variable.

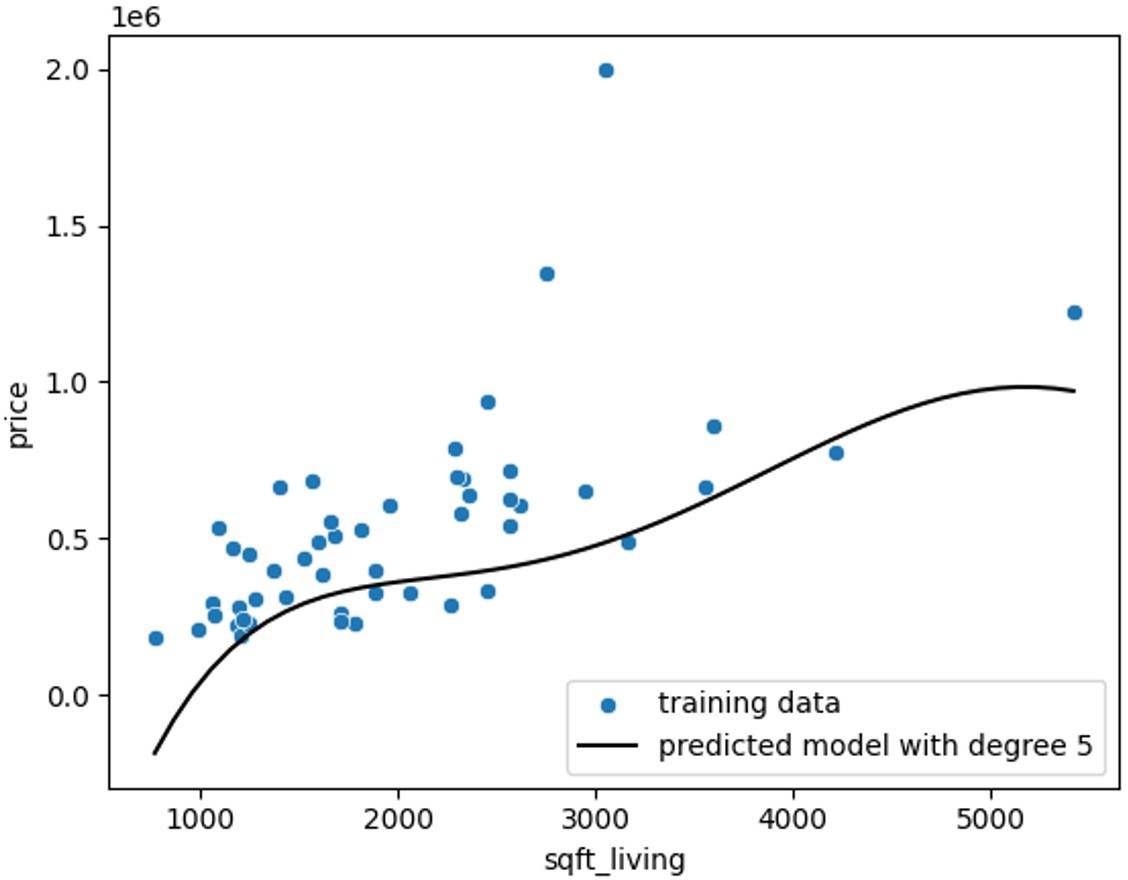
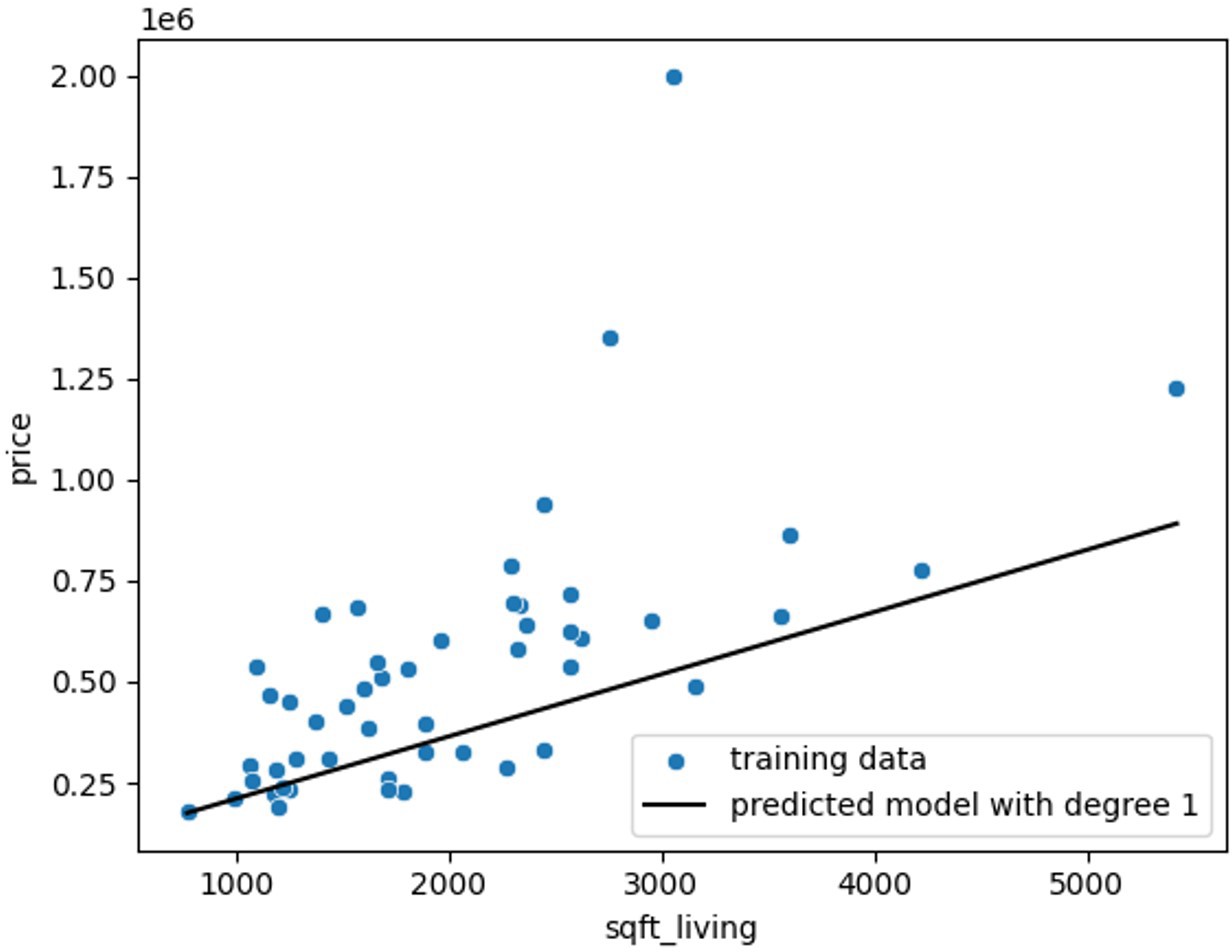


Figure 2.3: *Linear Regression vs. Polynomial Regression*

Figure [2.3](#_bookmark30) illustrates the difference between linear and polynomial regression when using a single feature. Poly-

nomial regression uses a curve to fit the data, while linear regression uses a straight line. As the degree of the poly- nomial increases, the fit of the curve to the data improves and the difference between actual and predicted values decreases. In general, when the relationship between the target and independent variables is non-linear and can- not be represented by a straight line, polynomial regres- sion tends to provide a better fit than linear regression.

However, using a higher degree polynomial also in- creases the number of parameters and makes the com- putation more complex. For a single feature, the number of parameters for an *nth* degree polynomial is *n* + 1, and it would be even higher for multiple features.

### Multiple Features

The mathematical equation for a 2nd-order polynomial regression model with 2 independent variables is as fol- lows:

*y* = *ω*1 *x*2 + *ω*2 *x*1 + *ω*3 *x*2 + *ω*4 *x*2

1 2

+ *ω*5*x*1*x*2 + *b* (2.16)

A 2*nd* -order polynomial regression model with 4 features has 5 coefficients and one association and one cut-off point. As the polynomial degree and number of features increase, the number of coefficients also increases.

### Implementation

In contrast to linear regression methods, using polyno- mial regression requires an additional step, as illustrated in Figure [2.4.](#_bookmark33) The data must be transformed to a higher dimensional space in order to model the relationship be- tween the target and independent variables using a linear equation. Then, the parameters can be determined using the Gradient descent method, as previously described in section [2.2.](#_bookmark26)



Figure 2.4: *Polynomial Regression Implementation*

### Hyper-parameter

In polynomial regression, the degree of order is a hyper- parameter that must be selected and provided as input to the model. Typically, a higher degree of order will provide a better fit to the data and result in a lower difference between actual and predicted values. However, as the order increases, the model may attempt to fit every single data point in the data-set and lose its ability to generalize to new data, known as over-fitting. The topic of how to prevent over-fitting and choose appropriate hyper- parameters will be discussed in section [2.7.](#_bookmark55)

### Knowledge Test

* + - 1. How does polynomial regression differ from linear regression?

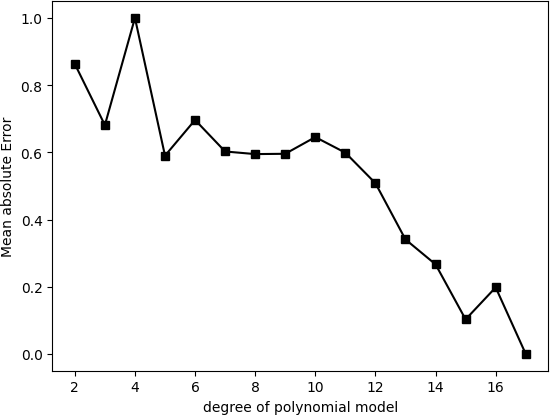


Figure 2.5: *Mean squared error vs. order of the Polynomial re- gression Model*

* + - 1. Figure [2.5](#_bookmark36) shows the plot of the mean squared er- ror obtained for different order of the polynomial regression model. Which order of the polynomial regression model has the lowest mean squared er- ror? Can you explain why this may or may not be the best choice?

## Model Implementation

The implementation of machine learning models has become much simpler and more efficient with the ad- vancement of various python libraries. This section will cover the step-by-step process of implementing a ma- chine learning model using Python.

### Data Preparation

Data preparation, also known as data pre-processing, is a necessary step before building any machine learning model. This stage includes:

* cleaning the data, such as removing outliers and identifying incorrect or missing information,
* analyzing the data to understand the relationship between the target and different features, and de- tect any correlation among features,
* creating new features from the existing features through feature engineering.

In summary, this stage makes the data suitable for use in the machine learning modeling process.

### Data Splitting: Train-Test Split

The objective of a supervised regression method is to predict the values of the unseen data using the model trained on the labelled data set. If we use the whole avail- able data set for training and estimate the parameters of the model, the model might provide a perfect score for the trained data but perform poorly for the unseen data. To evaluate how the model will perform on the unseen data, it is important to hold part of the data as the testing data.

Train-Test Split means dividing the labelled data into two subsets: training data set that is used to train the model and the testing data set that is used to evaluate the model. Please take note that your training data-set should be sufficiently large to capture the underlying correlation in the data-set. In general, two-third of the data is used as training data-set and the remaining one- third is used as the testing data.

Another important point to take note is to split the data-set using a random selection. This is to ensure that the train and test data sets are representative of the origi- nal data set.

##### Train-Test Split in Python

Data splitting can be easily implemented in python using the train test split function under the model selection module in the scikit-learn library. The function takes a loaded data set as input and returns the data set split into two subsets. It is a good practice to first extract the independent variables *X* and the target variable *y* from the labelled data set and then give *X* and *y* as input to the ’train test split’ function. The function randomly split both *X* and *y* into training and testing sets.

The below python codes read the *‘Advertising’* data set from the data folder and split them into *X* that contains the independent variables and *y* , the target variable. In this example, the sale amount is the target value and its value depends on the other three features which are the cost for the *‘TV’, ‘radio’, and ‘newspaper’.* programme.

*# ==================================================#*

import pandas as pd

from sklearn.model\_selection import train\_test\_split df=pd.read\_csv(’..\\data\Advertising.csv’)

X=df[[’TV’, ’radio’, ’newspaper’]] y=df[’sales’]

*# ==================================================#*

Then, the data set is randomly split into 67 percent as training and 33 percent as testing data.

*# ==================================================#*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,

test\_size = 0.33, random\_state=1)

*# ==================================================#*

The random state is set at 1 so that the same subset is provided by Python every time we run the code.

### Data Modelling

This stage involves modelling a machine learning method using the training data set. The python codes for both lin- ear regression and polynomial regression are discussed in this section.

##### Linear Regression in Python

Below codes show how to model a linear regression using Python sklearn library. Please take note that the variable *Xtrain* includes the multiple features. The codes for the simple and multiple linear regression model are the same except the dimensions of the variable *Xtrain* .

Fit function under the Linear Regression module finds

the best parameters that provides the minimum differ- ence between the actual and predicted values of the tar- get variable, sale amount.

*# ==================================================#*

from sklearn.linear\_model import LinearRegression

lr = LinearRegression() lr.fit(X\_train, y\_train)

print(lr.coef\_, lr.intercept\_)

*# ==================================================#*

Feature scaling is an important step in developing a multiple linear regression model and this can be per- formed using pre-processing module in sklearn library. The below python codes show how to perform the feature scaling and then use the scaled features to train a linear regression model.

*# ==================================================#*

*## feature scaling*

from sklearn.preprocessing import StandardScaler scale = StandardScaler()

X\_scaled = scale.fit\_transform(X\_train)

*## linear regression model*

from sklearn.linear\_model import LinearRegression

lr = LinearRegression() lr.fit(X\_scaled, y\_train) print(lr.coef\_, lr.intercept\_)

*# ==================================================#*

##### Polynomial Regression in Python

Polynomial regression is the extension of the linear re- gression with an additional step of the feature transfor-

mation and the below codes give how to perform polyno- mial regression using sklearn library.

*# ==================================================#*

*## Poly regression model*

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=5, include\_bias=False) X\_poly = poly.fit\_transform(X\_scaled)

from sklearn.linear\_model import LinearRegression lr\_poly = LinearRegression()

lr\_poly.fit(X\_poly, y\_train)

*# ==================================================#*

Polynomial Features module is used to transform the features in 5*th* order polynomial equation to a higher dimensional linear space before performing the linear regression on the transformed features.

##### Pipeline

Figure [2.6](#_bookmark41) summaries the processes involved in imple- menting a regression model. The feature scaling, polyno- mial transform and modelling can be performed in one step using the pipeline module in sklearn library.

Below codes show the implementing of a poly regres- sion model with degree 5 using the pipeline function.

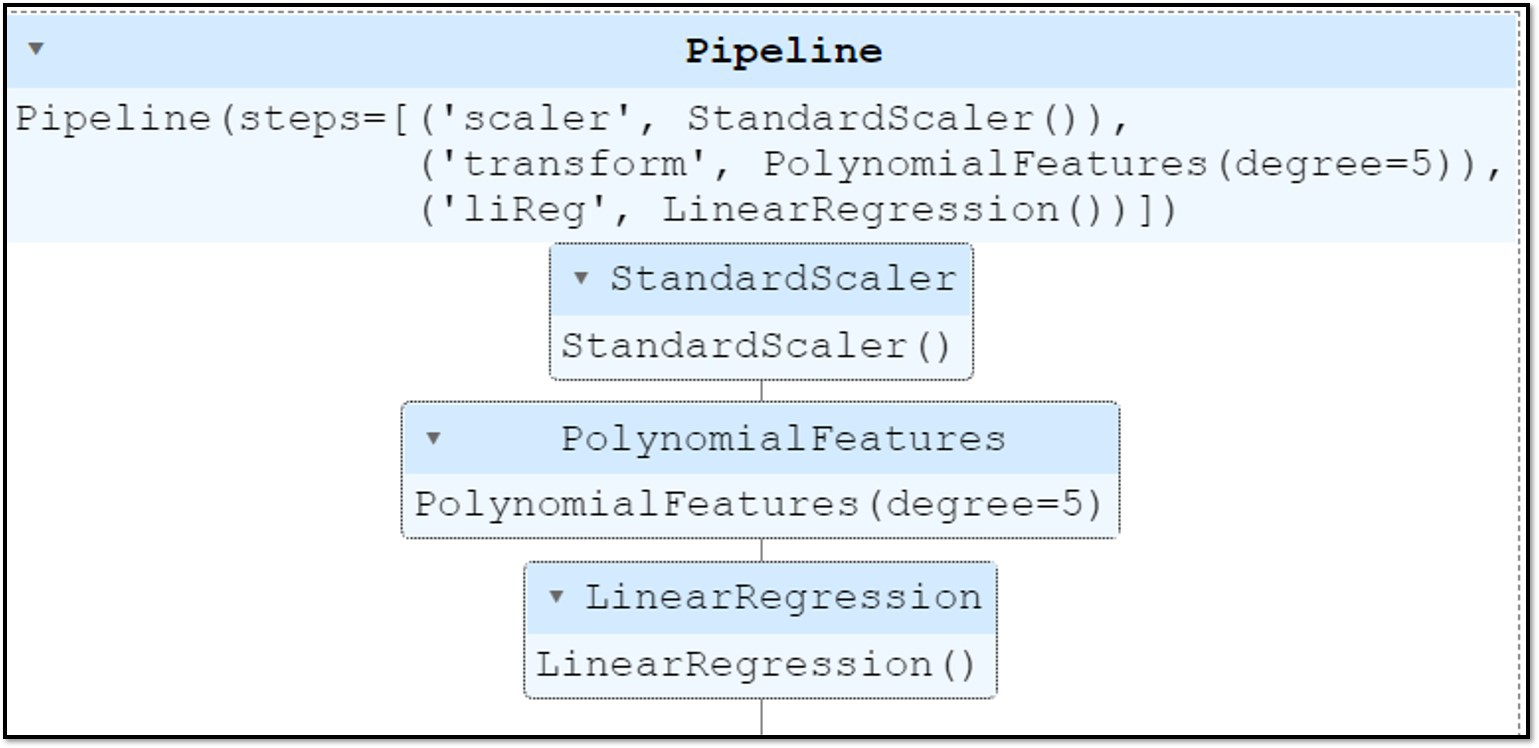


Figure 2.6: *Pipeline for Modeling a Regression Model*

*# ==================================================#*

from sklearn.pipeline import Pipeline

steps = [(’scaler’, StandardScaler()),

(’poly’, PolynomialFeatures(degree = 5)), (’liReg’, LinearRegression())]

pipeline = Pipeline(steps) pipeline.fit(X\_train, y\_train)

*# ==================================================#*

## Performance Evaluation

This stage evaluates the predictive performance of the model using the test data set. There are different metrics to evaluate the model and the most common evaluation metrics are mean absolute error, mean squared error, and r2-score.

### Mean Absolute Error

The mean absolute error (MAE) calculates how close the prediction is to the actual value on average and defines as:

*mae* = 1 |*y*˜ − *y* |, (2.17)

*N*

*i* *i*

*N*

*i* =1

### Mean Squared Error

The mean squared error or root mean squared error mea- sures the average squared residuals (difference between the actual and the predicted values).

*N*

*mse* = 1 (*y*˜ − *y* ) , (2.18)

2

*i* *i*

*N*

*i* =1

*r mse* =

1 *N*

*N i* =1

1I

(*y*˜*i* − *yi* )2, (2.19)

### R2-score

The R2 score (pronounced R-Squared Score) measures how well the data fit the regression model on a scale of zero to one and is defined as:

*R*2 = 1 − *RSS* , (2.20)

*T SS*

where *RSS* is the sum of the squared of residuals and *T SS* is the total sum of squared of difference between each data point and average values. In other words, *T SS* measures the variations of the data. *RSS* and *T SS* are defined as below:

*RSS* =

*T SS* =

*N*

(*yi* − *y*˜*i* )2 (2.21)

*i* =1

*N*

(*yi* − *y*¯*i* )2 (2.22)

*i* =1

where *y*˜*i* is the predicted value and *y*¯*i* is the mean value of the variable. The below table shows how the *RSS*, *T SS* and *R*2 values are computed for two extreme cases of *R*2 = 1 and *R*2 = 0.

***Case 1:*** *R*2 = 1

The first table shows for the case of *R*2 = 1. There is no difference between the predicted and the actual values. This results the RSS value to be zero and gives r2-score of one.

|  |  |  |  |
| --- | --- | --- | --- |
| *yi* | *y*˜*i* | (*yi* − *y*˜*i* )2 | (*yi* − *y*¯*i* )2 |
| 10 | 10 | 0 | 100 |
| 20 | 20 | 0 | 0 |
| 30 | 30 | 0 | 100 |
| *y*¯*i* = 20 |  | RSS = 0 | TSS = 200 |

Table 2.2: ***Case 1****: Predicted and Actual Values are the same.*

***Case 2:*** *R*2 = 0

The second table shows the case for *R*2 = 0. The model always return the average values of the actual data as pre- dicted values and this results *RSS* and *T SS* values to be the same and gives r2-score of zero. It can be seen from two extreme examples that *R*2 score is between 0 and 1 and the higher value indicates the better performance.

However, *R*2 score can be negative when the trained model fits worse than the baseline model which is re- turning the average values. Please note that there is a

|  |  |  |  |
| --- | --- | --- | --- |
| *yi* | *y*˜*i* | (*yi* − *y*˜*i* )2 | (*yi* − *y*¯*i* )2 |
| 10 | 20 | 100 | 100 |
| 20 | 20 | 0 | 0 |
| 30 | 20 | 100 | 100 |
| *y*¯*i* = 20 |  | RSS = 200 | TSS = 200 |

Table 2.3: ***Case 2****: Model always returns the average as predicted value.*

modified evaluation metric of ’r2 score’ and known as ’adjusted R2-score’. The adjusted r2-score considers the number of independent variables to detect the irrele- vant feature. The detailed calculation of the ‘adjusted R2-score’ is not in the scope of this book.

### Implementation using Python

Python *sklearn metrics* module can be used to compute the evaluation metrics. All function requires two parame- ters, the actual and the predicted values. The predicted values can be computed using the ’predict’ function of the trained model.

We must perform pre-processing steps such as feature scale or polynomial transformation on the testing data set or data during the deployment if we have used them when training a model. Otherwise, the feature space will be different and the model will not be able to perform effectively.

However, it is important that the transformation pa- rameters are learnt only from the training data but not from the testing data. If we use the information from the testing data set while building the model, the model may report a good performance during the testing but it will behave poorly during the actual deployment. This problem is known as the ‘**data leakage**’ problem and it is important to avoid using any information from the test- ing data set while performing the pre-processing steps.

##### Evaluating the Linear Model

The below codes show how to evaluate the multiple lin- ear model we have previously trained. During the train- ing,we have scaled the features using the standard scaler and hence, we need to scale the testing data as the way we did for the training. We have to use the same mean and variance values to transform the testing data, not by learning new parameters from the testing data set.

In Python sklearn model, this is achieved by calling the ‘*transform*’ function during the testing while we use ‘*fit transform*’ during the training.

*# ==================================================#*

from sklearn.metrics import mean\_absolute\_error from sklearn.metrics import mean\_squared\_error from sklearn.metrics import r2\_score

X\_test\_scaled = scale.transform(X\_test) ytest\_pred = lr.predict(X\_test\_scaled)

mae = mean\_absolute\_error(y\_test, ytest\_pred) print(’Mean absolute error: ’, mae)

mse = mean\_squared\_error(y\_test, ytest\_pred,

squared= True) print(’Mean squared error: ’, mse)

r2 = r2\_score(y\_test, ytest\_pred) print(’R2 score is: ’, r2)

*# ==================================================#*

The root mean squared error (rmse) can be computed by setting the parameter **’squared’** to False.

##### Evaluating the Polynomial Model

In the polynomial regression model, we have scaled the features using the standard scaler and also transformed them using 5*th* order polynomial transform. The same pre-processing steps have to be performed before pre- dicting the target values.

*# ==================================================#*

from sklearn.metrics import mean\_absolute\_error from sklearn.metrics import mean\_squared\_error from sklearn.metrics import r2\_score

X\_test\_scaled = scale.transform(X\_test) X\_test\_poly = poly.transform(X\_test\_scaled)

ytest\_pred = lr\_poly.predict(X\_test\_poly)

mae = mean\_absolute\_error(y\_test, ytest\_pred) print(’Mean absolute error: ’, mae)

mse = mean\_squared\_error(y\_test, ytest\_pred,

squared= True) print(’Mean squared error: ’, mse)

r2 = r2\_score(y\_test, ytest\_pred) print(’R2 score is: ’, r2)

*# ==================================================#*

##### Evaluating the Pipeline

The sklearn pipeline function provides a way to pre- vent the data leakage as it ensures that the appropriate method is performed on the correct data subset. We do not need to perform the pre-processing steps separately.

*# ==================================================#*

from sklearn.metrics import mean\_absolute\_error from sklearn.metrics import mean\_squared\_error from sklearn.metrics import r2\_score

ytest\_pred = pipeline.predict(X\_test)

mae = mean\_absolute\_error(y\_test, ytest\_pred) print(’Mean absolute error: ’, mae)

mse = mean\_squared\_error(y\_test, ytest\_pred,

squared= True) print(’Mean squared error: ’, mse)

r2 = r2\_score(y\_test, ytest\_pred) print(’R2 score is: ’, r2)

*# ==================================================#*

### Cross-Validation

Cross-validation uses multiple different testing data sets to evaluate the model performance and hence, this tech- nique gives a better idea of how the model will perform with the new data.

This is achieved by first splitting the initial training data set into k subsets (known as folds). Then, the model is repetitively trained using the k-1 subsets and tested on the remaining fold. This process is repeated till all the folds are used for testing. The model performance is then evaluated for all the folds and then reported the average performance.

Below codes show how to use the python sklearn ‘model selection’ module to train a cross-validated poly regression model. We use the pipeline function to avoid data leakage problem.

*# ==================================================#*

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

kfold = KFold(n\_splits=5,shuffle=False) steps = [(’scaler’, StandardScaler()),

(’poly’, PolynomialFeatures(degree = 4,

include\_bias=False)),

(’liReg’, LinearRegression())] pipeline = Pipeline(steps)

mse = -1\*cross\_val\_score(pipeline, X\_train, y\_train,

scoring = ’neg\_mean\_squared\_error’, cv=kfold)

print(’average mean squared error is’, np.mean(mse))

*# ==================================================#*

The parameter ‘n-splits’ in the ‘KFold’ function defines the number of subsets whereas the ‘cross val score’ func- tion provides the score (defined by the scoring parame- ter) for each fold.

In this example, the number of splits is 5 and scoring parameter is defined as negative mean squared error. The cross validation method is used to find the hyper- parameters using a Grid search method. Optimizing the hyper-parameters will be discussed in section [2.7.](#_bookmark55)

### Bias and Variance Trade-off

The bias-variance trade-off is a foundational concept in evaluating the supervised machine learning methods.

##### Bias

Bias is the difference between the average prediction and the actual value. A model that over-simplifies and gener- alizes the relationship between the target and indepen- dent variables fails to capture the actual trend in the data and tend to have a high bias. In general, a simple lin- ear regression are more likely to produce high bias. The model under-fits the data and its produce the high error for both training and testing data set.

##### Variance

Variance refers to what extent the accuracy of the ma- chine learning model changes depending on the data set. It is not uncommon to see that a machine learning model performs pretty well for the training data set but performs poorly in the new data set or testing data set. This is the results of training a machine learning model on a small data set or when the model tries to over-fit the training data set and loss generalization.

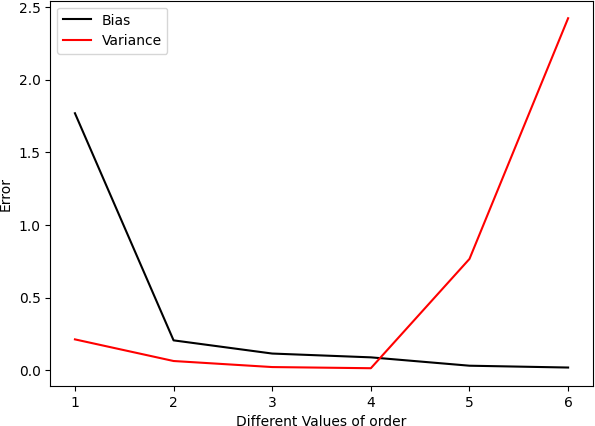


Figure 2.7: *Error vs. Order of regression model*

##### Bias Variance Trade-off

Figure [2.7](#_bookmark49) shows how the model in the previous example of the sale amount prediction problem performs against the order of the polynomial regression model. We can see that the error for the training data set (bias) becomes smaller and smaller when the order of the polynomial regression model increases (in other words, the model

becomes more complex). On the other hand, the vari- ance (the performance difference between the training and testing data set) increases as the model complexity increases.

A machine learning model should have a good balance between the bias and variance so that it does not over-fit or under-fit the problem. In the above example, the best result will be obtained at the order 4.

## Regularization

Regularization is a technique that uses to minimize the over-fitting of the data, especially when there is a large variance between the train and test set performances. One of the common methods for regularization is to mod- ify the loss function by adding a regularization term. This section discusses two common methods: lasso and ridge regression.

### Lasso Regression

Lasso regression is also known as *L*1 regularization as it adds a *L*1 penalty term based on to the absolute value of the magnitude of the coefficient.

*N Np*

*ε* = 1 (*y*˜ − *y* )2 + *α* ∥ *ω* ∥, (2.23)

*i i j*

*N i* =1

*j* =1

where *N* is the number of data points and *y*˜*i* is the pre- dicted value using the model, *Np* is the number of param- eters, *ωj* , *j* = 1, 2,..., *Np* are parameters of the model and the parameter *α* controls the amount of regularization.

The parameter *α* reduce the value of coefficients and when it is sufficiently large, some of the coefficients *ωi* will be eliminated. Hence, lasso regression is espe-

cially useful when the model is suffering from the multi- collinearity or when you have a high dimensional data set (number of features is more than the number of obser- vations (data)) and want to automatically remove some of the less important features.

### Ridge Regression

Ridge regression is similar to lass regression, excepts that the ridge regression adds *L*2 norm as the penalty term.

*N Np*

*ε* = 1 (*y*˜ − *y* )2 + *α* *ω*2, (2.24)

*i* *i*

*N i* =1

*j*

*j* =1

where *N* is the number of data points and *y*˜*i* is the pre- dicted value using the model, *Np* is the number of param- eters, *ωj* , *j* = 1, 2,..., *Np* are parameters of the model and the parameter *α* controls the amount of regularization.

Compared to lasso, the ridge regression will never elim- inate the coefficient and hence, ridge regression can not be used for automatic feature selection.

### Lasso vs. Ridge

In the previous example of ‘*sale amount prediction*’, the amount of sale is predicted using three advertisement programme and the multiple linear regression model is given as:

*y* = 3.893*x*1 + 3.420*x*2 + 2.943*x*3 + 13.788, (2.25)

where *xi* are independent features. We can see that the variable *x*1 is the most important feature and has the largest coefficient.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *α* = 0 | *α* = 0.5 | *α* = 1 | *α* = 10 |
| *ω*1 (L) | 3.893 | 3.420 | 2.943 | 0 |
| *ω*2 (L) | 2.761 | 2.310 | 1.832 | 0 |
| *ω*3 (L) | 0.067 | 0 | 0 | 0 |
| *ω*1 (R) | 3.893 | 3.879 | 3.865 | 3.625 |
| *ω*2 (R) | 2.761 | 2.749 | 2.738 | 2.546 |
| *ω*3 (R) | 0.067 | 0.072 | 0.078 | 0.163 |

Table 2.4: *Effects of the regularized parameter α on the Parame- ters*

In this section, we re-implement the above model us- ing the ridge and lasso regression with different regu- larized weight and Table [2.4](#_bookmark54) shows how the regularized parameter *α* in the Lasso and Ridge model effects the magnitude of the coefficients. Please note that the imple- mentation is done just to demonstrate the effect of the regularized parameter *α* and it is not necessary to do the regularization in this problem.

We can see that the lasso regression method eliminates the coefficients for the least important variable *x*3 when the parameter *α* becomes sufficiently large *α* = 10 but

the ridge regression only reduces the magnitude of the

high coefficients but keeps the lower coefficient value.

## Hyper-Parameter Tuning

Hyper-parameters are parameters that are pre-defined before modelling. The typical parameters for a regression model are the order of the polynomial regression model, *k* and the regularized parameter *α* in the Lasso and Ridge model. The cross-validation method discussed in section

[2.5.5](#_bookmark47) is used to find the hyper-parameter for a machine learning model and these parameters are later used to develop a model to be deployed and tested on the initial testing data set.

Grid search and random search are common methods to find the best hyper parameters for a model. These methods first define a parameter space that includes a set of possible hyper-parameter values that can be used to build the model. Grid search method uses every com- bination of hyper-parameter values to train the model and select the best hyper-parameter.

Random search randomly selects and tests a random combination of hyper-parameters. It is more efficient than grid search for the higher number of parameters. Below code shows how to find the hyper parameter *k* for the poly regression model using the Grid search method.

*# ==================================================#*

from sklearn.model\_selection import GridSearchCV

steps = [(’scaler’, StandardScaler()),

(’poly’, PolynomialFeatures(degree = order,

include\_bias=False)), (’liReg’, LinearRegression())]

parameters = {"poly degree":[1, 4, 6, 9]} pipeline = Pipeline(steps)

poly\_grid = GridSearchCV(pipeline, parameters,

cv=4, scoring=’neg\_mean\_squared\_error’)

poly\_grid.fit(X\_train, y\_train)

print (’best order is :’, poly\_grid.best\_params\_)

y\_pred\_test = poly\_grid.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred\_test)

*# ==================================================#*

## Project: Sale Amount Prediction

The complete example for predicting the sale amount based on the different advertisement programme is given below. This is a small data set with four features and 200 rows of entries. The target feature is the ‘sale amount’ and the other three features are cost for ‘TV’, ‘radio’ and ‘newspaper’ programme respectively. The data set can be downloaded from [here.](https://www.kaggle.com/sazid28/advertising.csv) The data is already clean and there is no missing data.

### Data Importing

We first import the necessary modules and read the data from the computer.

*# ==================================================#*

import pandas as pd

df=pd.read\_csv(’..\\data\Advertising.csv’)

X=df[[’TV’, ’radio’, ’newspaper’]].values y=df[’sales’].values

*# ==================================================#*

### Train-Test-Split

The data is then split into the training data set and testing data set. Two-third of the data is used for training and one-third is kept for testing.

*# ==================================================#*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,

test\_size = 0.33, random\_state=1)

*# ==================================================#*

### Modeling and Hyper-Parameter Tuning

The grid search method is used to find the hyper parame- ter *k*, order of the polynomial regression model. Pipeline function is used to avoid the potential data leakage issue.

*# ==================================================#*

from sklearn.model\_selection import GridSearchCV from sklearn.preprocessing import StandardScaler from sklearn.preprocessing import PolynomialFeatures from sklearn.linear\_model import LinearRegression from sklearn.pipeline import Pipeline

steps = [(’scaler’, StandardScaler()),

(’poly’, PolynomialFeatures(degree = 2,

include\_bias=False)), (’liReg’, LinearRegression())]

parameters = {"poly degree":[2, 3, 4, 5, 7, 9]} pipeline = Pipeline(steps)

poly\_grid = GridSearchCV(pipeline, parameters,

cv=5, scoring=’neg\_mean\_squared\_error’, verbose= True)

poly\_grid.fit(X\_train, y\_train)

print (’best order is :’, poly\_grid.best\_params\_)

### Model Evaluation

The best model is used to evaluate the performance on both the training and testing data set.

*# ==================================================#*

rom sklearn.metrics import mean\_absolute\_error from sklearn.metrics import mean\_squared\_error from sklearn.metrics import r2\_score

*# Evaluation on the Tesing data set*

ytest\_pred = poly\_grid.predict(X\_test)

mae = mean\_absolute\_error(y\_test, ytest\_pred) mse = mean\_squared\_error(y\_test, ytest\_pred,

squared= True) r2 = r2\_score(y\_test, ytest\_pred) *#Evaluation on the Training data set*

ytr\_pred = poly\_grid.predict(X\_train)

maeT = mean\_absolute\_error(y\_train, ytr\_pred) mseT = mean\_squared\_error(y\_train, ytr\_pred,

squared= True) r2T = r2\_score(y\_train, ytr\_pred) *#Keep all results in the tabular form*

result = pd.DataFrame({’mae’: [maeT, mae],

’mse’: [mseT, mse],

’r2’: [r2T, r2]})

result.index = [’Training’, ’Testing’]

The readers are encouraged to download the complete codes and data set from the public [**GitHub Repo**](https://github.com/myothida/Intro-To-Supervised-Machine-Learning.git)and try running using different data set.

*Chapter 3*

# Classification

Classification is the act or process of dividing things into groups according to their type.

*Cambridge dictionary*

In machine learning, classification methods assign an input data into a class label based on the pre-learnt infor- mation from the training data. In our previous example of bio-authentication system, a face is classified into two classes: ’authorized’ or ’reject’ based on the pre-trained

information. The terms we learnt in ’regression methods’ such as training data set, testing data set, residuals or errors, cost function and independent variables will also be used in the classification in the same content.

Refer back to the example of sale amount prediction in section [2.8,](#_bookmark56) the regression model predicts the sale amount (numerical value) based on the independent variables (advertisement cost from different programme). Consider a scenario where we want to check if the sale amount meets the target or not. Your output can be only two value : ‘meet the target’ (yes) or ‘do not meet the tar- get’ (no). This is called a **binary classification** problem as there are only two possible outputs ‘Yes : 1’ and ‘No: 0’.

Common problems of binary classification problems include

* predicting if an email is spam (1) or not (0) based on the previous examples.
* predicting if a tumor is malignant (1) or not (0) based on the similar tumours.
* predicting if a customer is default (1) or not (0) based on the customer’s behaviour.

**Multi-class classification** problem is a task with more than two possible outputs. For example, assume that we have a set of names from 135 different races of Myanmar. The objective of a multi-class classifier is to label each name as one of the races. We have 135 classes in this example.

There are other classification problems such as **multi- label classification** where an input is labelled to more than one class. This is a common problem in text cate- gorization where a text document belongs to more then one label (class).

## Performance Evaluation

### Classification Performance

There are different metrics for evaluating the perfor- mance of a classification model and this section dis- cusses the most common metrics before we start learning about any particular classification method. All the met- rics discussed in this section is based on the binary clas- sification problem and the formulation and concept can be easily extended to multi-class classification problem. It is important to note that there is no one-size-fit-all metric to measure the performance of a classification model. Different metrics are to be used depending on

the problem.

### Confusion Matrix

In binary classification problem, there are only two pos- sible outputs from the classification model: one and zero.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Predicted output | 0 | 1 | 0 | 0 | 1 | 1 |
| Actual Label | 0 | 0 | 1 | 0 | 0 | 1 |

Table 3.1: *Example labels of predicted and actual output.*

Table [3.1](#_bookmark65) lists the examples of predicted outputs and actual labels. There are only four possible combinations:

* + - * Case 1: Predicted as one and Actual value is one (11)
      * Case 2: Predicted as one but Actual value is zero (10)
      * Case 3: Predicted as zero and Actual value is one (01)
      * Case 4: Predicted as zero and Actual value is zero (00)

A good classification model should have higher num- bers of case 1 and case 4 and lower numbers of case 2 and case 3. From Table [3.1](#_bookmark65), we can see that there are

* + - * 1 number of case 1 (11)
      * 2 number of case 2 (10)
      * 1 number of case 3 (01)
      * 2 number of case 4 (00)

This information can be summarized in a square matrix form below:

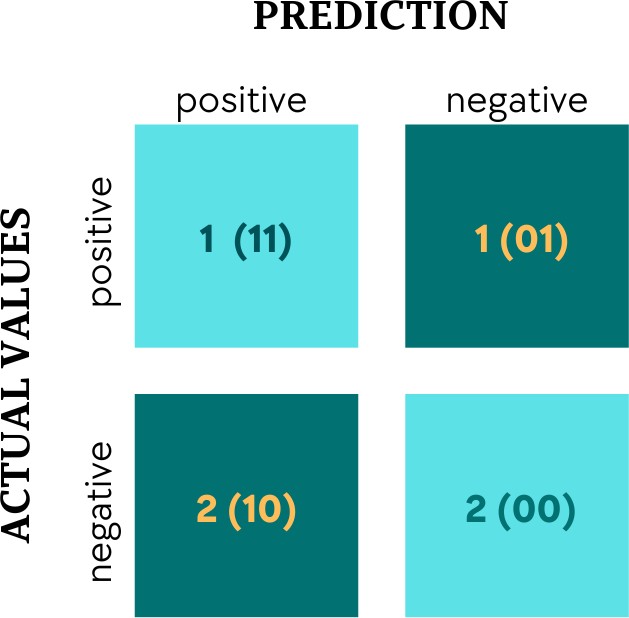


Figure 3.1: *Confusion Matrix for Table* [*3.1*](#_bookmark65)

The rows indicate the actual values and the columns indicate the predicted values. The first column refers to the cases where the model predicted as one (positive). Among 3 cases where the model predicted as one, 1 case is correct (true) as the actual value is also one. This value

is reflected in the first square box.

In general, **Confusion matrix** is a squared matrix (as shown in Figure [3.](#_bookmark66)2) that summarizes 4 different com- binations of predicted and actual values using the four terms defined as below:

* + - * True Positive (**TP**) refers to Case 1 where the model predicts as one (positive) and actual value is also one (positive)
      * False Positive (**FP**) refers to Case 2 where the model predicts as one (positive) but actual value is zero (negative)
      * False Negative (**FN**) refers to Case 3 where the model predicts as zero (negative) but actual value is one (positive)
      * True Negative (**TN**) refers toCase 4 where the model predicts as zero (negative) and actual value is also zero (negative)

In this book, we use the same conversion as the Python sklearn where the rows are actual values and columns indicate the predicted ones. Another common version of the confusion matrix is to define the rows as predicted values and the columns as actual ones.

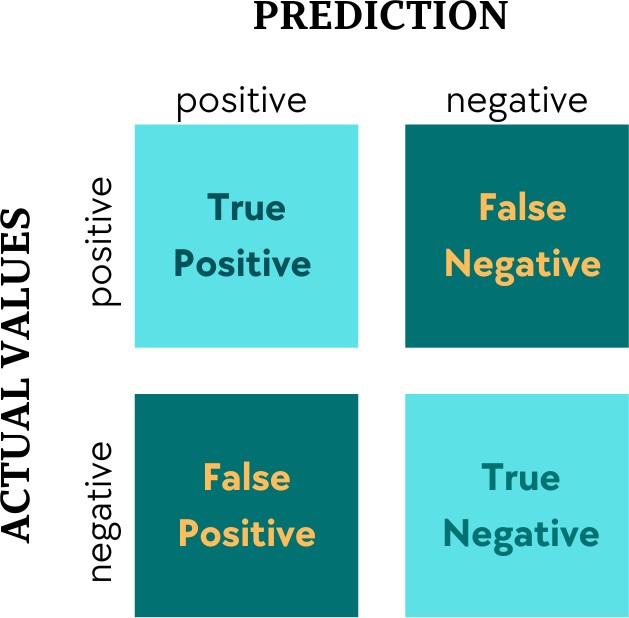


Figure 3.2: *Confusion Matrix: rows (Actual) and columns (Pre- diction)*

### Accuracy

Accuracy measures the ratio of the number of correct predictions to the total number of predictions. For exam- ple, in Table [3.1,](#_bookmark65) there are six predictions (samples) and 3 out of six predictions are correct. Hence, the accuracy value is 50% for the example in Table [3.1.](#_bookmark65) Generally, the

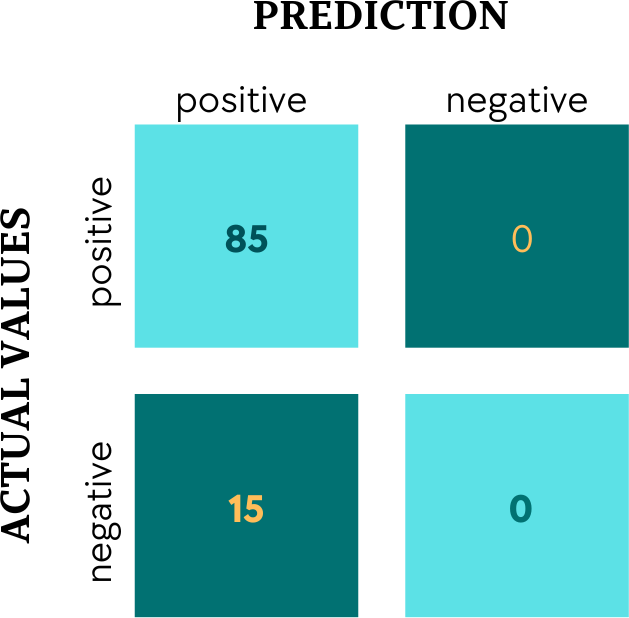
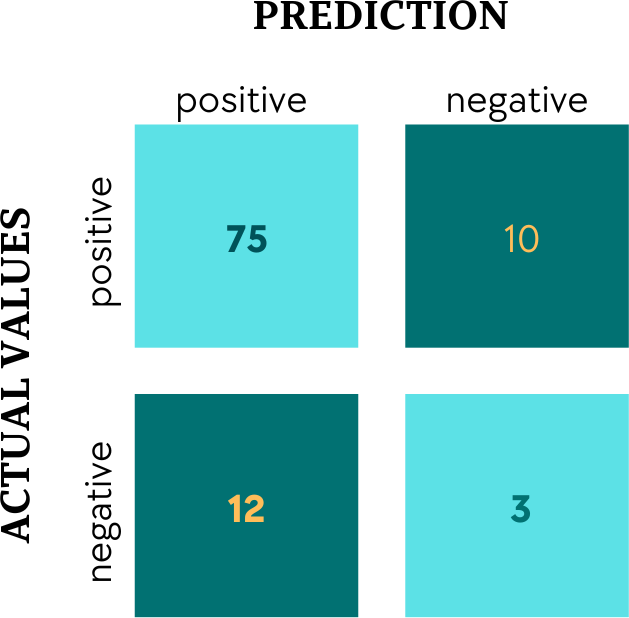
accuracy value is defined as:

*Accuracy* = *T P* + *T N*

*T P* + *FP* + *F N* + *T N*

(3.1)

However, the accuracy value does not reflect the perfor- mance in the case of imbalanced classes. Let’s consider a case where there are 85 number of positive cases and 15 number of negative cases. The confusion matrix for



(a) *Model 1* (b) *Model 2*

Figure 3.3: *Confusion Matrix from Model 1 and Model 2*

model 1 and model 2 are given in Figure [3.3](#_bookmark68). It can be seen from the confusion matrix that model 1 (Figure [3.3a](#_bookmark68)) accurately predicted 88% for positive cases and 100% for negative cases whereas model 2 (Figure [3.3b](#_bookmark68)) accurately predicted 100% for positive cases but 0% for negative

cases. We can see that model 1 performs from confusion matrices, but if we use the accuracy metrics to measure the performance of the model, the accuracy of the model is only 78% while the accuracy of the model 2 is 85%.

The accuracy is not always a good metrics to measure the performance for a problem with imbalanced classes. Other metrics such as Precision and Recalls should be considered.

### Precision

The precision measures how many of the predicted cases are correct true cases and it is defined as:

*Precision* = *T P*

*T P* + *FP*

(3.2)

This measurement is important when the false positive is more important concern than the false negative. For example, in the problem of spam filtering, labeling an important email as spam (false positive) is a concern than having a spam email in the mailbox (false negative). A high precision value is a requirement for the problem like ‘spam filtering system’.

### Recall (Sensitivity)

The recall measures how many positive cases can be ac- curately detected and it is defined as:

*Recall* (sensitivity) = *T P*

*T P* + *F N*

(3.3)

This is an important measurement for medical cases such as ‘cancer detection’. Missing a cancer case (false nega- tive) is more dangerous than having false alarms (detect- ing as a positive for a non-cancer patient). Recall should be as high as possible.

Recall is also called as *sensitivity* as this metric mea- sures how sensitive the model is or *true positive rate* as it is calculated by dividing the number of accurately detected positive cases by the total number of positive cases.

### True Negative Rate (Specificity)

The specificity or *true negative rate* measures how many negative cases are accurately detected and it is defined as:

*T NR*(Specificity) = *T N*

*FP* + *T N*

(3.4)

### F-score

F-score is used to measure both precision and recall at the same time and it is defined as:

*F* − *score* = 2 ∗ *Recall* ∗ *Precision*

*Recall* + *Precision*

(3.5)

The F-score is high when both the precision and recall are high but low when one of them has low value. F-score is an effective metrics in a problem where both the false positive and false negative are equally important.

### Area under the ROC (AUROC or AUC)

The area under the ROC (receiver operating characteris- tic) curve is one of the most important evaluation metrics and this measures the capability of the model on distin- guishing the classes correctly. Before we talk about AUC, we need to understand the ROC curve that plots two met- rics:

* + - * True Positive Rate (TPR = Recall)
      * False Positive Rate (FPR = 1 - specificity)

Many classification methods such as logistic regression produces the probability of an input belonging to the pos- itive class. Then, the class label is obtained by comparing

the resulted probability with a pre-defined threshold. A higher threshold value will reduce the number of positive predictions (both TP and FP). This is demonstrated in Table [3.2](#_bookmark74). There are 117 actual positive cases (FN + TP) and 7042 negative cases (TN + FP). When the threshold value is set at zero, all the predictions will be classified as positive cases (Class 1) and hence, all positive cases will be correctly classified (TP = 117) and then, the true posi- tive rate, TPR will be 1. As the threshold value increases, the true positive will be reduced but the true negative will increase. At another extreme case of threshold value at 1, all the cases will be predicted as negative class and hence, all negative cases will be classified (TN = 7042) but none of the positive classes will be correctly detected.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threshold | TP | TN | TPR | FPR |
| 0 | 117 | 0 | 1 | 1 |
| 0.2 | 101 | 7013 | 0.863 | 0.001 |
| 0.5(default) | 93 | 7037 | 0.795 | 0.0007 |
| 0.8 | 93 | 7039 | 0.752 | 0.0004 |
| 1 | 0 | 7042 | 0 | 0 |

Table 3.2: *How does the rate change based on the threshold value.*

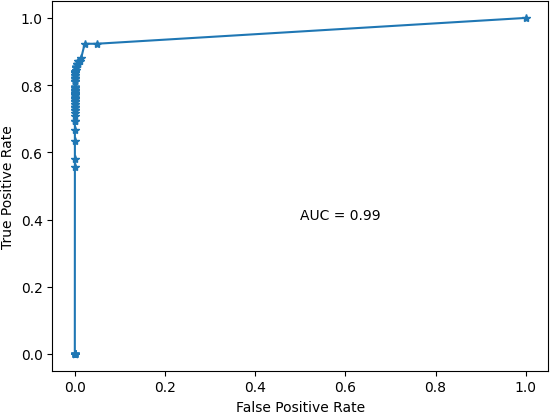


Figure 3.4: *Binary ROC curve*

ROC curve plots the true positive rate vs. the false positive rate as shown in Figure [3.4.](#_bookmark75) It can be seen that the model achieve a high score of area under the curve (AUC=0.99) which means it has a good ability to distin- guish between positive class and negative class.

## Logistic Regression

Logistic regression, similar to a linear regression, maps input values to the estimated target class using a sigmoid function defined as below:

*y*˜ = *g* (*θT x*) = 1

1 + exp(−*θT x*)

, (3.6)

where *θ* is the model parameters and *x* denotes an inde- pendent variable. In the case of a single feature, *θT x* is equal to *ω*1*x* + *b*. The predicted values of *y*˜ will be always zero and one. This can be further illustrated by plotting the *g* (*z*)

*g* (*z*) = 1

1 + exp(−*z*)

where *z* is defined as *z* = *θT x*.

, (3.7)

Figure [3.5](#_bookmark77) shows that the value of *g* (*z*) is always bounded between zero and one. When the values of *z* approaches to infinity, the *g* (*z*) becomes zero and on the other hand, *g* (*z*) becomes one when values of *z* in- creases.

Since the target variable *y* is a categorical value and it is always 0 or 1 in binary classification problem, the cost

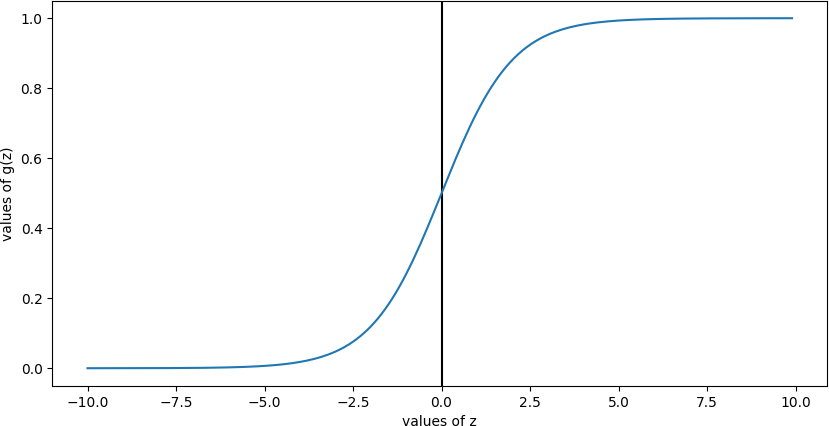


Figure 3.5: *Demonstration of Sigmoid Function*

function for the logistic regression is defined as

*J* (*θ*) = *cost* (*g* (*z*), *y* ) = 1− log(*g* (*z*)) if *yi* = 1 (3.8)

*i*

− log(1 − *g* (*z*)) if *yi* = 0

The above equation can be combined into a single equa- tion as:

*J* (*θ*) = −*yi* log(*g* (*z*) − (1 − *yi* ) ∗ log(1 − *g* (*z*)) (3.9)

If the estimated label *g* (*z*) is the same with the actual output *y* , the cost value will be zero as shown in Table [3.3](#_bookmark79).

|  |  |  |
| --- | --- | --- |
| Actual | Estimated label | cost |
|  |  | *J* (*θ*) = − log(*g* (*z*)) |
| *y* = 1  *y* = 1 | *g* (*z*) = 1  *g* (*z*) = 0 | *J* (*θ*) = − log(1) = 0  *J* (*θ*) = − log(0) = 1 |
|  |  | *J* (*θ*) = − log(1 − *g* (*z*)) |
| *y* = 0  *y* = 0 | *g* (*z*) = 1  *g* (*z*) = 0 | *J* (*θ*) = − log(1 − 1) = −1  *J* (*θ*) = − log(1 − 0) = 0 |

Table 3.3: *Results of Cost function in* [*3.8.*](#_bookmark78)

In general, for multiple number of observations, the cost function can be defined as an average of the cost values for all observations.

*N*

1 [ ]

*J* (*θ*) = − *y* ∗ *log* (*g* (*z*) + (1 − *y* ) ∗ *log* (1 − *g* (*z*)

*N i* *i*

*i* =1

(3.10)

where *N* is the number of data points (sample or ob- servations). Similar to linear regression, we can use the gradient descent method discussed in section [2.2](#_bookmark26) or any optimization algorithm to find the best parameter *θ* that leads to the minimum cost.

### Logistic Regression in Python

Below codes show the implementation of a logistic re- gression classifier using Python sklearn library. Please take note the model implementation steps discussed in

[2.4](#_bookmark37) are also applied here.

In this example, we use the *‘default’* data set to predict if a borrower will default (A default occurs when a bor- rower stops making the required payments on a debt) or not. There are 30 features (columns) in the data set and the target variable (default = 1 or not default = 0) is listed in the ’Class’ column. All the remaining 29 columns are used as independent features.

*# ==================================================#*

import pandas as pd

df=pd.read\_csv(’..\\data\\fraud.csv’) y = df[’Class’].values

X = df.drop(columns = ’Class’).values

from sklearn.model\_selection import train\_test\_split X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,

test\_size = 0.33, random\_state=1)

*#*

*## Using piepline to implement Logistic regression ## #*

from sklearn.preprocessing import StandardScaler from sklearn.linear\_model import LogisticRegression from sklearn.pipeline import Pipeline

steps = [(’scaler’, StandardScaler()), (’logReg’, LogisticRegression())]

clf\_pipeline = Pipeline(steps) clf\_pipeline.fit(X\_train, y\_train)

*#*

*## Model Evaluation ## #*

from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix from sklearn.metrics import roc\_auc\_score

ypred\_test = clf\_pipeline.predict(X\_test) mat\_clf = confusion\_matrix(y\_test, ypred\_test)

report\_clf = classification\_report(y\_test, ypred\_test)

print(mat\_clf) print(report\_clf)

ypred\_testP = clf\_pipeline.predict\_proba(X\_test) auc = roc\_auc\_score(y\_test, ypred\_testP[:,1]) print(auc)

*# ==================================================#*

### Hyper-parameters

There are many algorithms that can solve the optimiza- tion problem such as Newton method, Stochastic Average Gradient (SAG), etc. The type of solver becomes one of hyper-parameter in implementing the logistic regression method and we can try to find the best solver by using the grid or random search method. The detail explanation of different optimization algorithm is beyond the scope of this book.

Another important hyper-parameter for the logistic regression method is the strength of the regularization term. As discussed in section [2.6,](#_bookmark50) a regularization term can be added to the above cost function to regularize the over-fitting issue.

Python sklearn library allows us to set different hyper- parameters in the *LogisticRegression* function. Refer to [sickit documentation](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression) for the details of the parameters.

## K Nearest Neighbors or k-NN

Birds of a feather flock together.

*English Proverb*

The K nearest neighbor or k-NN is the most simplest classification method and easy to understand. Imagine you are a new student in my class and I do not know much about you yet but I see you are hang out with good students. Intuitively, I will assume that you will also be a good student. The same concept is used in k-NN and this method

* calculates the distance from a new data point to the other data points in the training data set
* selects the K neighbors who are nearest to the new data point
* assigns the new data point to the group of the ma- jority of the nearest K data points

Figure [3.6](#_bookmark83) demonstrates the concept of k-NN using a simple example of binary classification problem. k-NN computes the distance between the new data point (black diamond) and all the data points from both positive (blue

circle) and negative (red square) classes. If the number of nearest neighbour is set at 3 (k = 3), shown in yellow circle, the 2 neighbors of the test data belongs to the positive class and only one belongs to the negative class. Hence, the new data point will be assigned to the positive class.

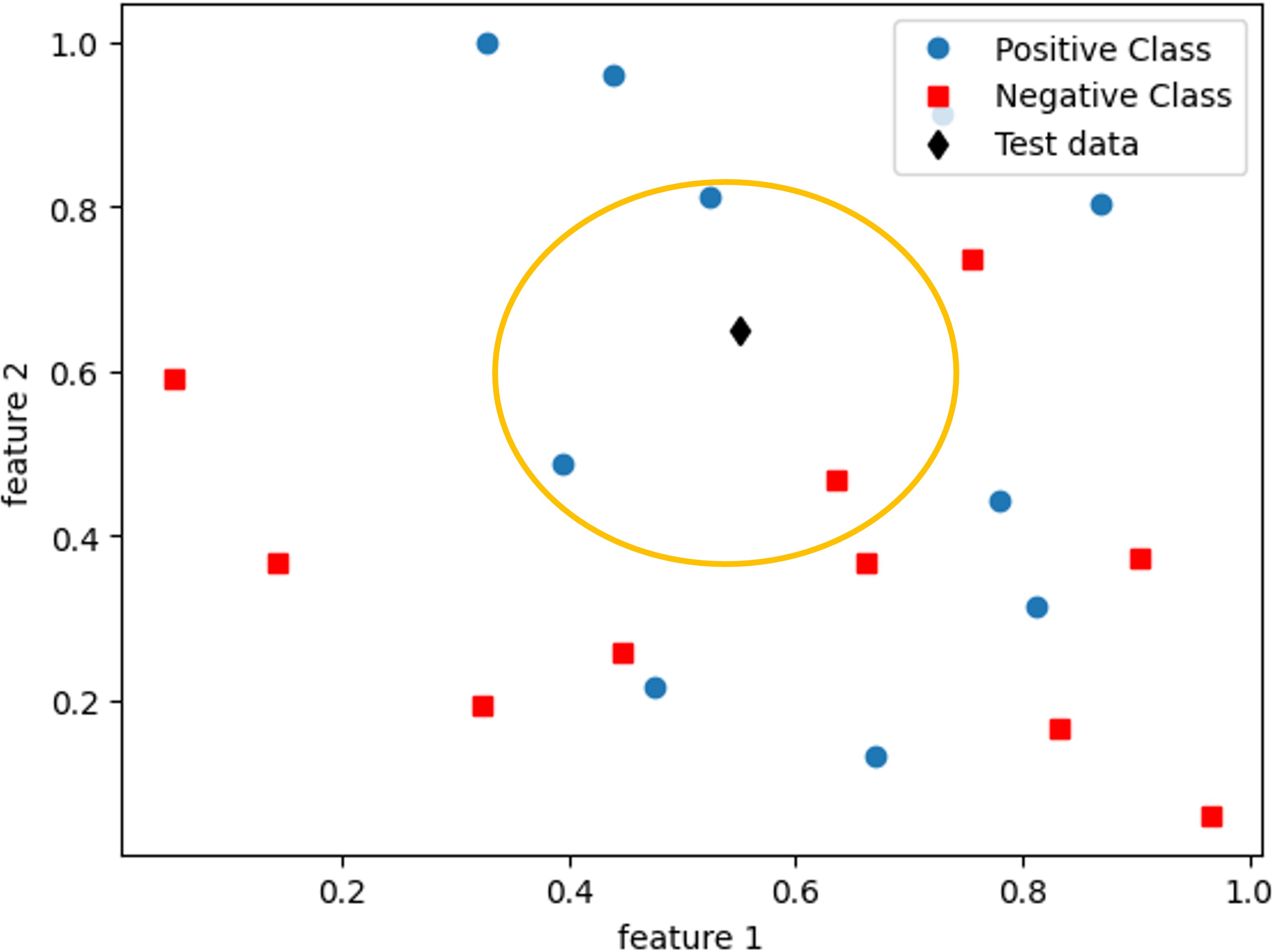


Figure 3.6: *Concept of KNN*

### Distance Metrics in k-NN

The first step of k-NN is to compute the distance from a new data point to all the other data points in the train- ing data set. One of the common metrics is *Euclidean distance* where the distance between two data points (*X*1 and *X*2) in n-dimensional space is defined as:

*n* J *i i* 2

*d* (*X*1, *X*2) =

(*x*1 − *x*2)

(3.11)

*i* =1

Another common metric is *Manhattan distance* that measures the absolute distance between two data points and is defined as:

*d* (*X*1, *X*2) = /l*xi* − *xi* /l (3.12)

*n*

1

2

*i* =1

Other distance metric includes Minkowski distance, cosine distance and hamming distance.

### Values of k in k-NN

Another important parameter in k-NN method is to de- fine the number of neighbors or the value of k. The k value defines how many neighbors will be checked to label the new data point. The performance of the model will depend on the value of k. If the k value is set at smaller number, the model might perform well for the training data set but the variance can be high. On other hand, the higher k value can lead to the low variance but high bias. As k-NN uses the voting, it is also important to choose the odd number of k value to avoid the ties in voting.

### K-NN in Python

Example codes below show how the k-NN can be imple- mented using Python.

*# ==================================================#*

*#*

*## Using piepline to implement k-nn classifier ## #*

from sklearn.neighbors import KNeighborsClassifier

steps = [(’scaler’, StandardScaler()),

(’knn’, KNeighborsClassifier(n\_neighbors = 5))]

knn\_pipeline = Pipeline(steps) knn\_pipeline.fit(X\_train, y\_train)

*#*

*## Model Evaluation ## #*

from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix from sklearn.metrics import roc\_auc\_score

ypred\_test = knn\_pipeline.predict(X\_test) mat\_clf = confusion\_matrix(y\_test, ypred\_test)

report\_clf = classification\_report(y\_test, ypred\_test)

print(mat\_clf) print(report\_clf)

ypred\_testP = knn\_pipeline.predict\_proba(X\_test) auc = roc\_auc\_score(y\_test, ypred\_testP[:,1]) print(auc)

*# ==================================================#*

## Support Vector Machine

Support Vector machine is one of the most commonly used methods in solving the classification problems due to its proven performance. The objective of a support vec- tor machine method or SVM is to find a decision bound- ary that distinctly classifies the data points. There are five key terms used in SVM, namely *‘Hyperplane’, ‘Decision Boundary’, ‘Support Vectors’, ‘Margin’* and ‘*SVM kernel’*. We need to familiar with these terms to understand the SVM better.

### Hyperplane

In geometry, a hyperplane is defined as a subspace of a vector space whose dimension is one less than that of its vector space. For example, in 2-dimensional space, its hyperplane is a line (one-dimensional) and in 3- dimensional space, its hyperplane is a 2-dimensional plane (surface).

### Decision boundary

The decision boundary refers to a line (in 2 dimensional space) or a **hyperplane** (in N-dimensional space where N is the number of features) that separates different classes of data in a feature space.

Figure [3.7](#_bookmark90) demonstrates the concept of the decision boundaries in 2-dimensional and 3-dimensional feature space. In 2D space (only two features), the two classes (class 1 and class 2) can be separated by a straight line but in 3D space, the decision boundary becomes a plane. It is harder to visualize for the higher dimensional space but in reality, most of the classification problem is in higher dimensional space. Hence, the SVM community usually refers the decision boundary as a hyperplane for general. As can be seen in Figure [3.7,](#_bookmark90) there can be more than one hyperplane that can separate the data into different classes.

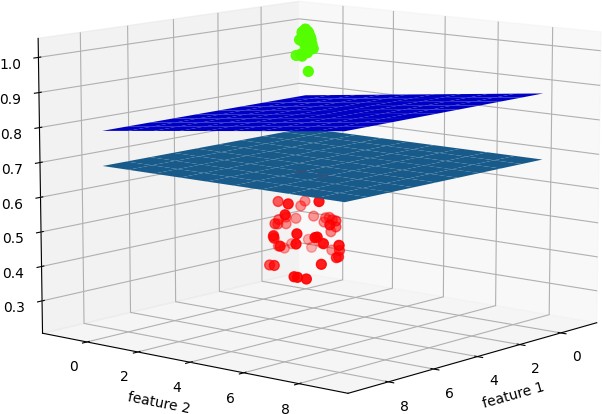
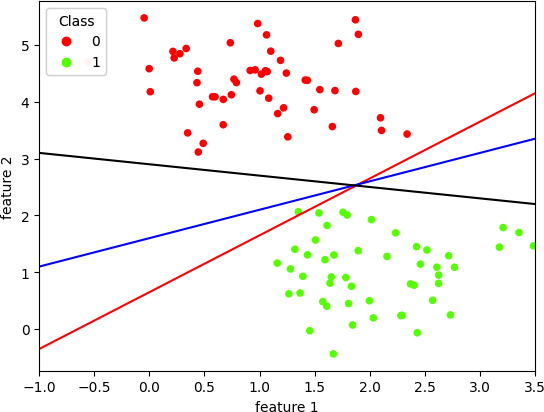


Figure 3.7: *Demonstration of SVM decision boundaries*

### Support Vectors

SVM finds the hyperplane that distinctly classifies the data points by maximizing the distance between the de- cision boundary and data points that are the closet to the decision boundary. These data points are called as ‘*support vectors*’ in SVM.

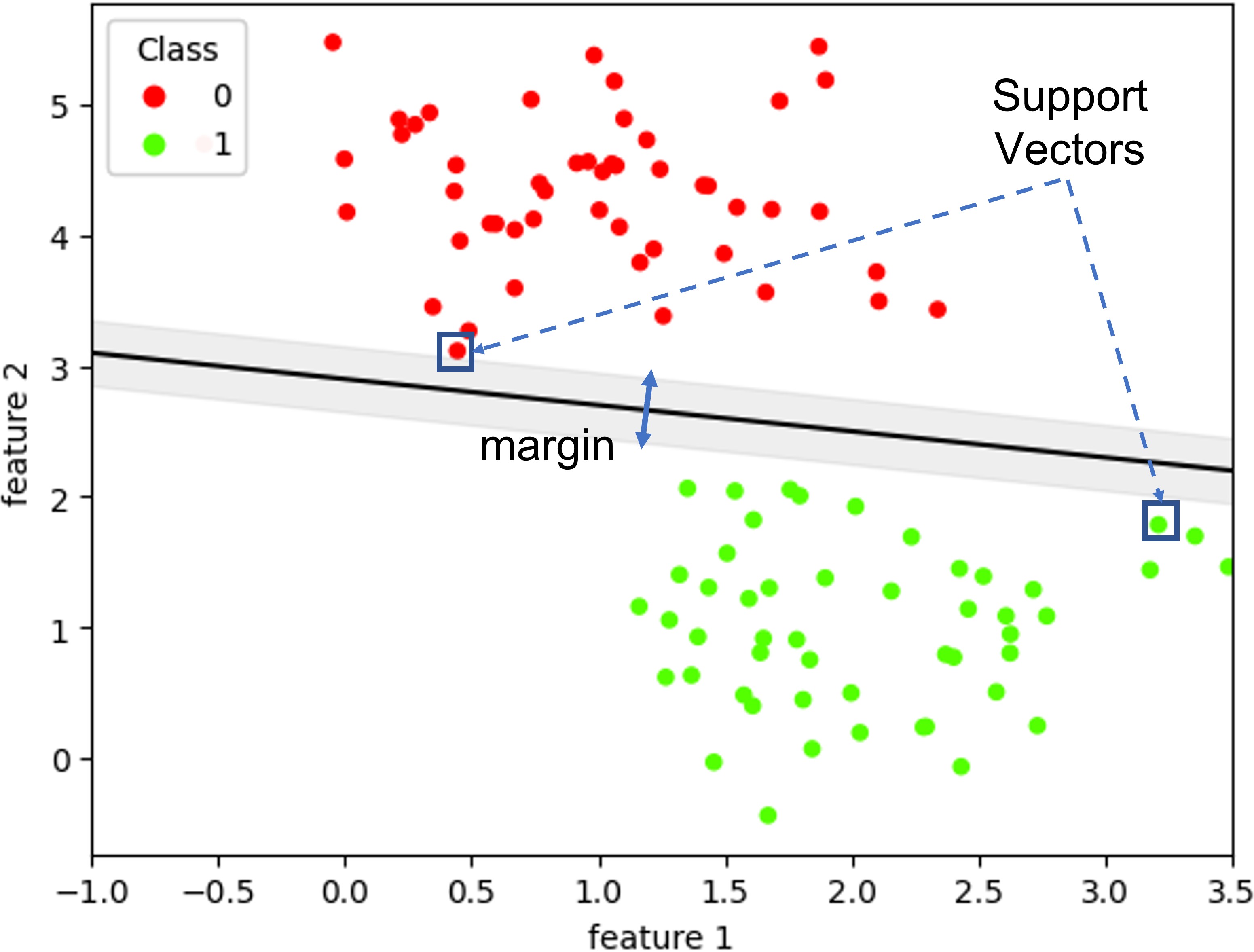


Figure 3.8: *Support Vectors in 2-Dimensional feature space*

In Figure [3.8,](#_bookmark92) the support vectors are shown in the square box and removing these data points can alter the position and the orientation of the decision boundary.

### Margin

Margin is the perpendicular distance between the sup- port vectors and the hyperplane. The objective of SVM is to find the maximum margin that can separate the data points between two classes. The margin is shown with the gray color in Figure [3.8.](#_bookmark92)

### SVM Kernels

The data points shown in Figure [3.8](#_bookmark92) are linearly separable and SVM can find the best hyperplane that can distinctly classify between two classes. However, in most of the real problems, the data points from two classes are not lin- early separable. An example of a linearly non-separable data points from two classes are shown in Figure [3.9](#_bookmark95). One important key advantage of SVM is using a kernel to clas- sify the non-linearly separable classes.

The function of a kernel is to transform the data into a higher dimension so that a hyperplane can separate them into different classes. There are many different types of SVM-Kernels and three kernels are discussed in

this section.

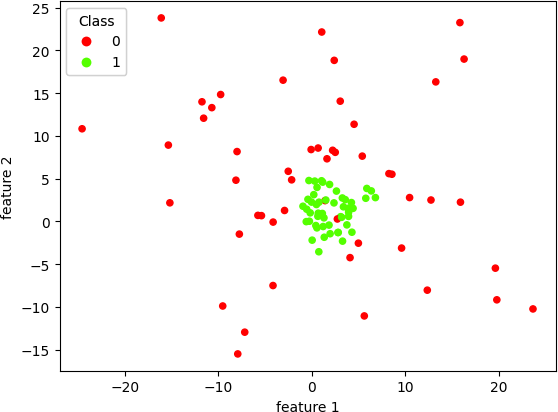


Figure 3.9: *Linearly non-separable data set*

##### Linear Kernel

A linear kernel is defined as:

*K* (*X*1, *X*2) = *X T X*2 (3.13)

1

This kernel is used only when the classes can be sepa- rated linearly.

##### Polynomial Kernel

A polynomial kernel uses a polynomial function to map the data into a higher-dimensional space. Mathemati- cally,

*K* (*X*1, *X*2) = (*X T X*2 + *c*)*d* (3.14)

1

where *c* is the constant parameter and usually set at 0 or 1. *d* is the degree (or order) of the polynomial function and *X*1 and *X*2 are n-dimensional data points. Similar to the polynomial regression, the degree *d* is a hyper-parameter that controls the complexity of the kernel and the large *d* value may over-fit the classification problem and it is important to choose the right degree *d* to balance the bias-variance trade off as discussed in Section [2.7](#_bookmark55).

##### Radial Bias Function (RBF) kernel

RBF kernel computes the similarity between two data points *X*1 and *X*2 as follows:

*K* (*X*1, *X*2) = exp(−*γ*/l*X*1 − *X*2/l2) (3.15)

where

1

*γ* = 2*σ*2 (3.16)

and *σ* is the variance of the kernel and control the width of the similarity region. /l*X*1 − *X*2/l is Euclidean (*L*2-norm) distance between two data points *X*1 and *X*2.

The value of *σ* decides which points should be con- sidered as similar. Figure [3.10](#_bookmark96) shows how the *σ* controls the region of the similarity. It can be seen that when the distance between two data points /l*X*1 − *X*2/l is zero, the value of the RBF kernel will be always one regardless of the *σ* value. The value of the RBF kernel decreases exponentially as the distance between two data points

/l*X*1 − *X*2/l increases.

**case 1:** *σ* = 0.5 the value of the RBF kernel is zero when the distance is greater than 2.

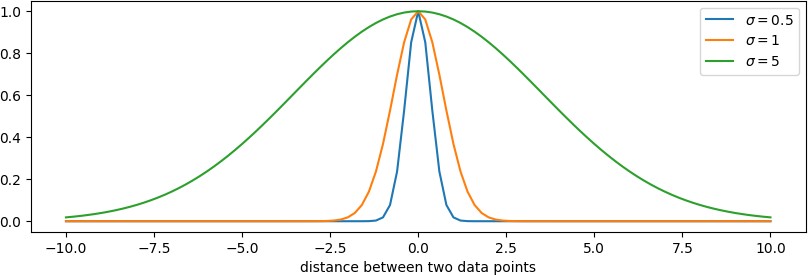


Figure 3.10: *RBF kernel for different σ values*

**case 2:** *σ* = 1 the value of the RBF kernel is zero when the distance is greater than 4.

**case 3:** *σ* = 5 the value of the RBF kernel is zero only when the distance is greater than 10.

The width of the region of similarity increases as the values of the *σ* increases. The smaller value of *σ* or the larger value of the *γ* tends to over-fit the classification problem as it considers the data points are close only if the distance between two data points is very small. The right choice of the hyper-parameter *γ* value will affect the performance of the RBF-SVM classifier.

### Hyper-parameters

Hyper-parameter optimization plays a crucial role in de- termining the performance of the SVM classifier. They are:

##### Kernel

The type of kernel is one of hyper-parameter in imple- menting the SVM classifier and the right choice of the kernel will lead to a good performance. In Python, we can use the grid or random search method to find the right kernel.

##### Regularization parameter (C)

As discussed in Section [2.6,](#_bookmark50) a regularization term can be added to prevent the over-fitting and the parameter *C* in Python sklearn controls the strength of the regularization. [A practical guide to Support Vector Classifier](https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf) guided that the value of the *C* should be searched in the range of

*C* ∈ [2−5, 215].

##### Degree (d)

In polynomial kernel, the degree or order of the polyno- mial function is a hyper-parameter to be tuned for the SVM classifier. Generally, the value of *d* is set between 0 and 10.

##### Kernel coefficient (γ)

In RBF kernel, the value of *γ* is an important hyper- parameter to be tuned. [A practical guide to Support Vec-](https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf)

[tor Classifier](https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf) guided the value of *γ* to be searched in the rage of *γ* ∈ [2−15, 23] .

### Linear SVM in Python

Below codes show how to implement a linear SVM classi- fier using Python sklearn library.

*# ==================================================# #*

*## Using piepline to implement SVM classifier ## #*

from sklearn.svm import SVC

steps = [(’scaler’, StandardScaler()), (’svc’, SVC(kernel = ’linear’))]

svc\_pipeline = Pipeline(steps) svc\_pipeline.fit(X\_train, y\_train)

*#*

*## Model Evaluation ## #*

from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix from sklearn.metrics import roc\_auc\_score

ypred\_test = svc\_pipeline.predict(X\_test) mat\_clf = confusion\_matrix(y\_test, ypred\_test)

report\_clf = classification\_report(y\_test, ypred\_test)

print(mat\_clf) print(report\_clf)

auc = roc\_auc\_score(y\_test, ypred\_test) print(auc)

*# ==================================================#*

### Polynomial Kernel SVM in Python

In this section, we show how to implement a non-linear SVM classifier using the polynomial kernel. The degree value *d* is set at 5 in SVC function. This hyper-parameter value can be tuned using the ‘GridSearchCV’. The code is given in Section [2.7](#_bookmark55).

*# ==================================================# #*

*## Using piepline to implement SVM Poly classifier ## #*

from sklearn.svm import SVC

steps = [(’scaler’, StandardScaler()),

(’svc’, SVC(kernel = ’poly’, degree = 5))]

svc\_pipeline = Pipeline(steps) svc\_pipeline.fit(X\_train, y\_train)

*#*

*## Model Evaluation ## #*

from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix from sklearn.metrics import roc\_auc\_score

ypred\_test = svc\_pipeline.predict(X\_test) mat\_clf = confusion\_matrix(y\_test, ypred\_test)

report\_clf = classification\_report(y\_test, ypred\_test)

print(mat\_clf) print(report\_clf)

auc = roc\_auc\_score(y\_test, ypred\_test) print(auc)

*# ==================================================#*

### RBF Kernel SVM in Python

Below codes show how to implement RBF kernel using

the Python. The gamma value is set at *γ* = *N* 1 that is

*σX*

defined as a default value in python sklearn library and called as ‘scale’.

*# ==================================================# #*

*## Using piepline to implement SVM RBF classifier ## #*

from sklearn.svm import SVC

steps = [(’scaler’, StandardScaler()),

(’svc’, SVC(kernel = ’rbf’, gamma = ’scale’))]

svc\_pipeline = Pipeline(steps) svc\_pipeline.fit(X\_train, y\_train)

*#*

*## Model Evaluation ## #*

from sklearn.metrics import classification\_report from sklearn.metrics import confusion\_matrix from sklearn.metrics import roc\_auc\_score

ypred\_test = svc\_pipeline.predict(X\_test) mat\_clf = confusion\_matrix(y\_test, ypred\_test)

report\_clf = classification\_report(y\_test, ypred\_test)

print(mat\_clf) print(report\_clf)

auc = roc\_auc\_score(y\_test, ypred\_test) print(auc)

*# ==================================================#*

## Project: Fraud Detection

There are many other classifiers such as Naivee Bayes classifier, tree-based classifiers, random forest, and deep learning methods that are not covered by this book. Deep learning methods have overshadowed other methods in many research publications.

However, many researchers and practitioners from in- dustries have pointed out that the choice of a machine learning method depends on several factors, including the specific problem at hand, the size and complexity of the data-set, and the computational resources that you have.

In this section, we will show the complete codes of the fraud detection using Logistic Regression, K-NN and SVM classifiers and compare the performance of the clas- sifiers using ‘accuracy’ and ‘recall’ metrics discussed in Section [3.1.](#_bookmark62) The data set contains 21, 693 entries and 29 independent features. The data set is imbalanced; there are 21, 337 non-fraud cases and 356 fraud cases. The data is already clean and there is no missing data. The data set can be downloaded from [here.](https://github.com/yonycherkos/Applied-Data-Science-with-Python-Specialization.git)

### Data Importing

We first import the necessary modules and read the data from the computer.

*# ==================================================#*

import pandas as pd

df=pd.read\_csv(’..\\data\\fraud.csv’, index\_col = 0) y = df[’Class’].values

df = df.iloc[:,1:]

X = df.drop(columns = ’Class’).values

*# ==================================================#*

### Splitting the data

The data is then split into the training data set and testing data set. Sixty percent of the data is used for training and 40% is kept for testing.

*# ==================================================#*

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,

test\_size = 0.40, random\_state=1)

*# ==================================================#*

### Modelling

Five different classifiers are developed. The names and hyper-parameters of the classifiers are given in Table [3.4](#_bookmark105). We add the regularization for all the classifiers except K- NN and the strength of the regularization is given by *C* . In logistic regression, the *L*2 term is used for regularization and squared *L*2 is used for SVM classifiers.

The number of neighbours in K-NN is given by the value of *k* and the order or degree of the polynomial function in SVM is defined as 3. The kernel coefficient *γ*

for SVM-RBF kernel is defined as *γ* = *N* 1 .

*σX tr*

|  |  |  |
| --- | --- | --- |
| Name | Hyper-parameters | |
| LR | C = 1.0 |  |
| KNN | k = 5 |  |
| SVM-Linear | C = 1.0 |  |
| SVM-Poly | C = 1.0 | degree = 3 |
| SVM-RBF | C = 1.0 | *γ* = *N* 1  *σX tr* |

Table 3.4: *Classifier model and its hyper-parameters*

*# ==================================================#*

from sklearn.preprocessing import StandardScaler from sklearn.pipeline import Pipeline

*#*

*## Logistic Regression ## #*

from sklearn.linear\_model import LogisticRegression

steps = [(’scaler’, StandardScaler()),

(’logReg’, LogisticRegression(penalty = "l2",

C = 1.0))]

LR\_pipeline = Pipeline(steps) LR\_pipeline.fit(X\_train, y\_train) *#*

*## K-NN Classifier ## #*

from sklearn.neighbors import KNeighborsClassifier

steps = [(’scaler’, StandardScaler()),

(’knn’, KNeighborsClassifier(n\_neighbors = 5))]

knn\_pipeline = Pipeline(steps) knn\_pipeline.fit(X\_train, y\_train)

*#*

*## SVM Classifier ## #*

from sklearn.svm import SVC

*## Linear Kernel ---------------*

steps = [(’scaler’, StandardScaler()), (’svc’, SVC(kernel = ’linear’,

class\_weight=’balanced’))]

svcL\_pipeline = Pipeline(steps) svcL\_pipeline.fit(X\_train, y\_train) *## Polynomial Kernel*

steps = [(’scaler’, StandardScaler()),

(’svc’, SVC(kernel = ’poly’, degree = 3, class\_weight=’balanced’))]

svcPoly\_pipeline = Pipeline(steps) svcPoly\_pipeline.fit(X\_train, y\_train) *## RBF Kernel*

steps = [(’scaler’, StandardScaler()),

(’svc’, SVC(kernel = ’rbf’, gamma = ’scale’, class\_weight=’balanced’))]

svcRBF\_pipeline = Pipeline(steps) svcRBF\_pipeline.fit(X\_train, y\_train)

*# ==================================================#*

### Model Evaluation

Next, the classifiers are evaluated using three metrics: ‘accuracy’, ‘recall’ and ‘area under the curve’. As discussed in Section [3.1,](#_bookmark62) ‘recall’ is an important metric for the fraud detection as we do not want to miss any fraud case. It is better to have false alarm than missing a fraud. Below codes calculate the metrics using the *classification report function* under sklearn.metrics module and extract the accuracy and recall parameters. The function roc-auc- score is used to compute the area under the curve. The evaluation is done for both training and testing set and the results are saved in tabular format.

*# ==================================================#*

from sklearn.metrics import classification\_report from sklearn.metrics import roc\_auc\_score

result\_df = pd.DataFrame(columns = [’Tr\_accuracy’,

’Test\_accuracy’, ’Tr\_recall’, ’Test\_recall’, ’Train\_auc’, ’Test\_auc’])

model\_name = [LR\_pipeline, knn\_pipeline, svcL\_pipeline, svcPoly\_pipeline, svcRBF\_pipeline]

for idx, model in enumerate(model\_name):

*## for training data*

ypred\_train = model.predict(X\_train) report\_clf = classification\_report(y\_train,

ypred\_train, output\_dict=True)

df\_r = pd.DataFrame(report\_clf).transpose() acc\_tr = df\_r.loc[’accuracy’, ’recall’].round(3) recall\_tr = df\_r.iloc[1,1].round(3)

auc\_tr = roc\_auc\_score(y\_train, ypred\_train)

*## for testing data*

ypred\_test = model.predict(X\_test) report\_clf = classification\_report(y\_test,

ypred\_test, output\_dict=True)

df\_r = pd.DataFrame(report\_clf).transpose() acc = df\_r.loc[’accuracy’, ’recall’].round(3) recall = df\_r.iloc[1,1].round(3)

auc = roc\_auc\_score(y\_test, ypred\_test)

result\_df.loc[idx,:]=[acc\_tr, acc, recall\_tr,

recall, auc\_tr.round(3), auc.round(3)]

result\_df.index = [’LR’, ’K-NN’, ’SVM-Linear’,

’SVM-Poly’, ’SVM-RBF’]

*# ==================================================#*

### Discussion on Performance

##### Accuracy

Table [3.5](#_bookmark108) lists the accuracy score for different classifiers. SVM-linear model performs the worst with the accuracy score of 97%. The performance is similar for both training and testing set.

However, as discussed in Section [3.4,](#_bookmark105) the accuracy score is not a good measure for the imbalanced classes. Our data set has 98% of non-fraud cases and only 2% of the fraud cases. This is common in many problems such as cancer detection, spam detection, etc. The negative class (class - zero) has always more data than the positive class (class - one).

|  |  |  |
| --- | --- | --- |
|  | Train | Test |
| LR | 0.996 | 0.996 |
| K-NN | 0.997 | 0.996 |
| SVM-Linear | 0.972 | 0.97 |
| SVM-Poly | 0.996 | 0.99 |
| SVM-RBF | 0.991 | 0.986 |

Table 3.5: *Accuracy for Fraud Detection using different methods.*

##### Recall

Table [3.5](#_bookmark108) lists the recall values for different classifiers. SVM with RBF kernel performs the best with the score of 99% for the training data set. It means 99% of the fraud cases in the training set is successfully detected by the SVM-RBF classifier. However, we can see that the SVM- RBF over-fits the training data and it performs the worst for the testing data set.

SVM classifier with the linear model produces the rea- sonable results for both training and testing data set at 91% and 88% respectively. We can see that the SVM- linear classifier has a good balance between the bias and variance.

|  |  |  |
| --- | --- | --- |
|  | Train | Test |
| LR | 0.79 | 0.788 |
| K-NN | 0.822 | 0.796 |
| SVM-Linear | 0.913 | 0.883 |
| SVM-Poly | 0.973 | 0.796 |
| SVM-RBF | 0.991 | 0.774 |

Table 3.6: *Recall for Fraud Detection using different methods.*

##### Area under the Curve

Table [3.5](#_bookmark108) lists the areas under the curve for fraud detec- tion using different classifiers. Similar to recall score, SVM classifier with the linear model produces the best re- sults (94% for training and 92.7% for testing set) without over-fitting the problem.

|  |  |  |
| --- | --- | --- |
|  | Train | Test |
| LR | 0.895 | 0.894 |
| K-NN | 0.911 | 0.898 |
| SVM-Linear | 0.943 | 0.927 |
| SVM-Poly | 0.984 | 0.895 |
| SVM-RBF | 0.991 | 0.882 |

Table 3.7: *Area under the curve using different methods.*

The results from this projet indicates that the model complexity is not directly proportional to the perfor- mance of the model. Higher model complexity can lead to the over-fitting and high variance can be observed. The hyper-parameters can be tuned to improve the per- formance of the classifier.

*Chapter 4*

# Further Reading

This book provides an overview of the basic concepts in machine learning, including linear and polynomial regression and parametric classifiers such as logistic re- gression and SVM, as well as non-parametric classifiers like K-NN. However, it should be noted that there are many other machine learning techniques such as gradi- ent boosting, tree-based classifiers, random forest, and deep learning methods that are not covered in this book. Deep learning methods have become increasingly pop- ular in recent years and are often highlighted in research publications. However, it’s important to keep in mind

that the choice of a machine learning method should be based on factors such as the specific problem, the size and complexity of the data-set, and the available compu- tational resources. Many researchers and practitioners in industry stress the importance of considering these factors when choosing a machine learning method.

ChatGPT [[2](#_bookmark115)] recommends the following math topics, online courses, and books for beginners interested in studying machine learning. This chapter is included to show how AI tools can be used to enhance the effective- ness and efficiency of educators and learners. It’s im- portant to note that the field of machine learning is con- stantly evolving, and this list may not always be up to date. Therefore, it’s important not to rely solely on the recommendations provided by ChatGPT and to choose resources that align with your learning style. However, the list provided by ChatGPT can serve as a useful starting point for quickly and comprehensively learning about the field

**The list is not comprehensive and generated by Chat- GPT** [**[2**](#_bookmark115)**].**

## Mathematic Topics recommended by ChatGPT

Mathematics plays a crucial role in understanding ma- chine learning methods, and ChatGPT has suggested the following topics as important to learn.

* + - Linear Algebra: understanding concepts such as matrix operations, eigenvalues and eigenvectors, and singular value decomposition. Linear alge- bra is used in many machine learning algorithms, including neural networks, principal component analysis (PCA), and singular value decomposition (SVD).
    - Calculus: understanding concepts such as deriva- tives and gradients, which are used in optimiza- tion algorithms such as gradient descent.
    - Probability and Statistics: understanding concepts such as probability distributions, Bayes’ theorem, and hypothesis testing. These concepts are used in many machine learning algorithms, including Bayesian networks and Gaussian mixture models.
    - Optimization: understanding concepts such as convex optimization, gradient descent, Newton’s

method etc. which are used in many machine learning algorithms, including linear regression and support vector machines.

* + - Information theory and entropy: understanding of the concept of entropy, mutual information, and cross-entropy which are used in many machine learning algorithms such as decision tree, infor- mation gain, and KL divergence.

## Courses recommended by ChatGPT

There are many different platforms that offer machine learning-related courses, including LinkedIn, Coursera, Edx, Udemy, Udacity, and DataCamp. The list provided below is based on ChatGPT’s suggestions. It’s recom- mended to select one of these courses to begin your learning journey and assess if it aligns with your preferred method of learning.

* + - "Introduction to Machine Learning" by Andrew Ng on Coursera
    - "Machine Learning" by Georgia Tech on Udacity
    - "Deep Learning" by Andrew Ng on Coursera
    - "Applied Data Science with Python" on Coursera
    - "Introduction to Machine Learning with Python" by Sarah Guido and Andreas Müller on Coursera
    - "Machine Learning A-Z: Hands-On Python and R In Data Science" by Kirill Eremenko and Hadelin de Ponteves on Udemy
    - "Machine Learning for Data Science and Analyt- ics" by Columbia University on edX

## Books recommended by ChatGPT

Online courses can be a fast and convenient way to learn a topic, but it’s important to be aware that not all courses are of equal quality. Some courses may be high-quality and effective in helping you to quickly grasp a topic, while others may be low-quality and not worth your time and energy. In some cases, low-quality courses can even discourage you from continuing your learning journey. While online courses can be a great resource, traditional methods like books can also provide a solid foundation for learning.

The following books are recommended by ChatGPT. I am particularly pleased to see "Pattern Recognition and Machine Learning" as the first recommendation, as it was the textbook I used during my undergraduate studies in 2003.

* + - "Pattern Recognition and Machine Learning" by Christopher M. Bishop: This book provides a com- prehensive introduction to the field of machine learning and covers a wide range of topics, from the basics of probability and statistics to advanced machine learning techniques.
    - "Deep Learning" by Yoshua Bengio, Ian Goodfel-

low, and Aaron Courville: This book provides a comprehensive introduction to deep learning, in- cluding the theory and practice of training deep neural networks.

* + - "Machine Learning: A Probabilistic Perspective" by Kevin P. Murphy: This book provides a compre- hensive introduction to machine learning from a probabilistic perspective and covers a wide range of topics, from supervised learning to unsuper- vised learning and Bayesian methods.
    - "Introduction to Machine Learning" by Alex Smola and S.V.N. Vishwanathan : This book provides an introduction to machine learning and covers a wide range of topics, including supervised learn- ing, unsupervised learning, and reinforcement learning.
    - "The Hundred-Page Machine Learning Book" by Andriy Burkov: This book is a concise and acces- sible introduction to machine learning, providing an overview of the most important concepts and algorithms in the field.
    - "Python Machine Learning" by Sebastian Raschka and Vahid Mirjalili: This book provides a hands-on

introduction to machine learning using Python, including coverage of a wide range of machine learning algorithms and libraries.

* + - "Machine Learning for Dummies" by John Paul Mueller: This book provides an easy-to- understand introduction to the basics of machine learning, including supervised and unsupervised learning techniques.
    - "Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow" by Aurélien Géron: This book provides an in-depth introduction to ma- chine learning using popular Python libraries such as Scikit-Learn, Keras, and TensorFlow.

# Bibliography

1. Microsoft. Visual studio code, 2015. cross-platform code editor. [1.3](#_bookmark7)
2. OpenAI. Chatgpt, 2022. developed by openAI. [1.1](#_bookmark2), [4](#_bookmark109)
3. Fernando Pérez. Juypter notebook, 2012. born from IPython project. [1.3](#_bookmark7)
4. Google Research Team. Google colab, 2018. Free to use. [1.3](#_bookmark7)
5. Myo Thida. Introduction to python, jupyter and google colab, 2022. Free source to learn Python Pro- gramming. [1.3.1](#_bookmark8)

Dr Myo Thida *Introduction to Supervised Machine Learning*

1. Guido van Rossum. Python, 1991. Python Software Foundation. [1.3](#_bookmark7)