Building Machine Learning Systems using Python

Practice to Train Predictive Models and Analyze Machine Learning Results with Real Use-Cases

DEEPTI CHOPRA

bpb

Building Machine Learning Systems Using Python

Practice to Train Predictive Models and Analyze Machine Learning Results with Real Use-Cases

Deepti Chopra

www.bpbonline.com

FIRST EDITION 2021

Copyright © BPB Publications, India

ISBN: 978-93-89423-617

Distributors:

BPB PUBLICATIONS
20, Ansari Road, Darya Ganj
New Delhi-110002
Ph: 23254990/23254991

DECCAN AGENCIES
4-3-329, Bank Street,
Hyderabad-500195
Ph: 24756967/24756400

MICRO MEDIA
Shop No. 5, Mahendra Chambers,
150 DN Rd. Next to Capital Cinema,
V.T. (C.S.T.) Station, MUMBAI-400 001
Ph: 22078296/22078297

BPB BOOK CENTRE
376 Old Lajpat Rai Market,
Delhi-110006
Ph: 23861747

To View Complete BPB Publications Catalogue Scan the QR Code:

Published by Manish Jain for BPB Publications, 20 Ansari Road, Darya Ganj, New Delhi-110002 and Printed by him at Repro India Ltd, Mumbai

www.bpbonline.com

Dedicated to

My Family and Friends
Who are always with me to love, support and care.

About the Author

Dr. Deepti Chopra is working as Assistant Professor (IT) at Lal Bahadur Shastri Institute of Management, Delhi. She has around 7 years of teaching experience. Her area of interest includes Natural Language Processing, Computational Linguistics and Artificial Intelligence. She is author of 3 books and has written several research papers in various International Conferences and Journals.

About the Reviewer

Anmol has 3 years of experience in the software industry. He has honed his skills in Machine Learning, Deep Learning, build and maintain ETL/ELT data pipelines and data-driven systems. Some of the industries Anmol has worked in are Airline, E-Commerce, Human Resource and HealthCare. His everyday work involves analysing and solving complex business problems, breaking down the work into feasible actionable tasks, and collaborating with his team and project manager to plan and communicate delivery commitments to our business clients.

When he is not working, Anmol spends most of his time reading, traveling the world, playing Fifa and catching his favourite Broadway shows. An admitted sports fanatic, he feeds his addiction to football by watching Real Madrid games on Sunday afternoons.

Acknowledgement

I want to thank God most of all, because without God I wouldn't be able to do any of this.

I acknowledge with gratitude and sincerely thank all my family and friends for the blessings and good wishes conveyed to me to achieve the goal to publish this machine learning based book.

My sincere thanks are due to few friends/TR of this book who encourages and motivate me every time and proves that they are always here for me. Such relations are the perfect example of 'Quality over Quantity'.

This book wouldn't have happened if I hadn't had the support from content editor of BPB Publications. My gratitude goes to the editorial and publishing professionals of BPB Publications for their keen interest and support in bringing out this book

Finally, I would like to thank Mr. Manish Jain at BPB Publications for giving me this opportunity to write my first book for them.

Preface

With the increase in availability of data from different sources, there is a growing need of data driven fields such as analytics and machine learning. This book intends to cover basic concepts of machine learning, various learning paradigms and different architectures and algorithms used in these paradigms.

This book is meant for the beginners who want to get knowledge about machine learning in detail. This book can also be used by machine learning users for a quick reference for fundamentals in machine learning.

Following are the chapters covered in this book:

Chapter 1: Introduction to Machine Learning

Description: This chapter covers basic concepts of machine learning, areas in which ML is performed, input-output functions

Topics to be covered:

1. What is machine learning?
2. Utility of ML
3. Applications of ML

Chapter 2: Linear Regression

Description: This chapter discusses about Linear Regression

Topics to be covered: List of topics covered in this chapter are:

1. Linear Regression in one variable
2. Linear Regression in multiple variables
3. Gradient descent
4. Polynomial Regression

Chapter 3: Classification using Logistic Regression

Description: This chapter discusses about classification using Logistic Regression

Topics to be covered: List of topics covered in this book are:

1. Binary Classification
2. Logistic Regression
3. Multi class Classification

Chapter 4: Overfitting and Regularization

Description: This chapter discusses about overfitting and regularization

Topics to be covered: List of topics covered in this chapter are:

1. Overfitting and regularization in linear regression
2. Overfitting and regularization in logistic regression

Chapter 5: Feasibility of Learning

Description: This chapter discusses about feasibility of learning

Topics to be covered: Topics covered in this chapter are:

1. Feasibility of learning an unknown target function
2. In-sample error
3. Out-of-sample error

Chapter 6: Support Vector Machine

Description: This chapter discusses about Support Vector Machine

Topics to be covered: Please provide the list of topics to be covered through the book:

1. Introduction
    - 1.1 Margin
    - 1.2 Large Margin methods
2. Kernel methods

Chapter 7: Neural Network

Description: This chapter discusses about Neural Network

Topics to be covered: List of topics covered in this chapter are:

1. Introduction
2. Early models
3. Perceptron learning
4. Backpropagation
5. Stochastic Gradient Descent

Chapter 8: Decision Trees

Description: This chapter discusses about decision trees

Topics to be covered: List of topics covered in this chapter are:

1. Decision Trees
2. Regression Tree
3. Stopping Criterion and Pruning Loss functions in Decision Tree
4. Categorical Attributes, Multiway Splits and Missing Values in Decision Trees
5. Instability in Decision Trees

Chapter 9: Unsupervised Learning

Description: This chapter discusses about Unsupervised Learning

Topics to be covered: List of topics covered in this chapter are:

1. Introduction
2. Clustering
    - 2.1 K-means Clustering
    - 2.2 Hierarchical Clustering
3. Principal Component Analysis'

Chapter 10: Theory of Generalization

Description: This chapter discusses about theory of generalization

Topics to be covered: List of topics covered in this chapter are:

1. Training versus Testing
2. Bounding the testing error
3. Vapnik Chervonenkis inequality
4. VC Dimension
5. Proof of VC inequality

Chapter 11: Bias and Fairness in ML

Description: This chapter discusses about bias and fairness in ML

Topics to be covered: List of topics covered in this chapter are:

1. Introduction
2. How to detect bias?
3. How to fix biases or achieve fairness in ML?

Downloading the code bundle and coloured images:

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/34fca5

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Table of Contents

Chapter 1
Introduction

Machine learning is one of the applications of artificial intelligence. Machine learning may be defined as the ability of the system to learn automatically through experience without being explicitly programmed. It is based on the development of programs that can access data and use this data to perform learning on their own. In this chapter, we will discuss the classification of machine learning, the various challenges faced in machine learning, and the applications of machine learning.

Structure

- History of machine learning
- Classification of machine learning
- Challenges faced in adopting machine learning
- Applications

Objectives

- Understanding the origin of machine learning

- Understanding the classification of machine learning algorithm
- Challenges faced in machine learning
- Applications of machine learning

History of machine learning

In 1940s, the first manually-operated computer, **ENIAC** (**Electronic Numerical Integrator and Computer**), was invented. At this time, the word *computer* was used which meant, *'a machine having intensive numerical computation capabilities'*. Since 1940s, the idea was to build a machine that could mimic human behavior of learning and thinking. In 1950s, the first computer game program was developed that could beat the checkers world champion. This helped checker players in improving their skills. At this time, *Frank Rosenblatt* invented Perceptron, which is a very simple classifier. Machine learning became popular in 1990s when probabilistic approaches of AI were born as a result of the combination of statistics and computer science. Because of the large data available, scientists started building intelligent systems that could analyze and learn from a large amount of data. For example, the IBMs Deep Blue could beat the World Chess Champion, *Garry Kasparov*. Machine learning is a kind of algorithm in which the software applications can accurately predict the outcomes without being explicitly programmed. The basic essence of machine learning is to build algorithms that, on receiving input data, predicts the output using statistical analysis and updates the output as the new data is made available. The term *Machine learning* was coined by an American scientist, *Arthur Samuel*, in 1959 who had expertise in computer gaming and artificial intelligence. According to *Arthur Samuel*, "*It gives computers the ability to learn without being explicitly programmed*". According to *Tom Mitchell* in 1997, "*A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.*"

Consider a machine learning based system which can help us find the traffic patterns in the busiest location. We can run a machine learning algorithm and use the traffic patterns of the past experience for training the system. If the system has learned successfully, then it will predict the traffic patterns in a better way with performance measure, *P*.

As machine learning is in boom in today's era, it is conducive to know the various applications as well as the challenges faced in machine learning. In this chapter, we will discuss the different machine learning techniques, the challenges faced in adopting machine learning, and the various application areas of machine learning.

Classification of machine learning

On the basis of the nature of learning and the response or output available to the learning system, machine learning implementations are of three types:

- **Supervised learning:** In supervised learning, the learning is performed using example data and its corresponding target response. During testing, when new examples are provided, it predicts the corresponding response. This learning is similar to how a student learns from a teacher. A teacher provides some good examples for the student to memorize. The student is then able to frame general rules to solve problems and draw useful conclusions.
- **Unsupervised learning:** In unsupervised learning, the learning is performed using example data without its associated target response. In this type of algorithm, a restructuring of data is performed where the data is segmented into different classes. The objects that belong to the same class have a high degree of similarity.
- **Reinforcement learning:** Reinforcement learning is similar to unsupervised learning in which, corresponding to the example data, there is no target response and each example is accompanied by a positive or a negative feedback. A positive feedback or credit is given when, during testing, a correct response is obtained corresponding to the example data. In a negative feedback, the error or penalty is awarded because, during testing, an incorrect response is obtained corresponding to the example data.
- **Semi-supervised learning:** In semi-supervised learning, during training, we have example data and some of the corresponding target responses are missing. It is a combination of supervised and unsupervised learning.

On the basis of the desired output, machine learning implementation is divided into the following types:

- **Classification:** In this type of learning, two or more classes are assigned to the input present in the training data. During testing, when we provide the input, it is classified into two or more classes. For example, in spam filtering, it classifies whether an email is spam or not spam.
- **Regression:** Regression is performed during supervised learning. In this type of learning, the output is continuous rather than discrete.
- **Clustering:** Clustering is performed during unsupervised learning in which the testing data is classified into groups and, unlike the task of classification, these groups or classes are not known beforehand.

The different types of machine learning algorithms are depicted in *Figure 1.1* as follows:

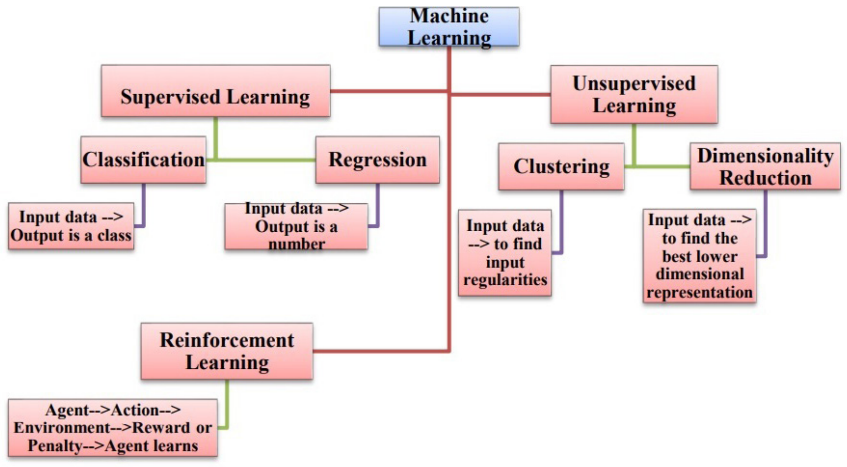

Machine Learning
Supervised Learning
Unsupervised Learning
Classification
Regression
Clustering
Dimensionality Reduction
Input data --> Output is a class
Input data --> Output is a number
Input data --> to find input regularities
Input data --> to find the best lower dimensional representation
Reinforcement Learning
Agent-->Action--> Environment-->Reward or Penalty-->Agent learns

Figure 1.1: *Classification of machine learning algorithm*

Challenges faced in adopting machine learning

There are various challenges in adopting machine learning for developing projects. Some of these are as follows:

- **Requirement of proper experimentation and testing:** We need to conduct frequent tests in a machine learning system in order to obtain the desired outcome with proper experimentation. The method used to test the machine learning algorithm is referred to as stratification. In this method, we randomly split the data set mainly into two subsets, training set and testing set.

- **Inflexible business models:** We should follow an agile and flexible business policy in implementing machine learning. If one of the machine learning strategies is not working, then we need to perform more experimentation and consequently build a new robust machine learning model.

Is the machine learning results ethical? Google is developing software that is used in military project called *Project Maven*. This project makes use of drone and will create autonomous weapons. Consequently, 12 employees of Google resigned in protest and more than 4000, along with over 1000 well-known scientists, signed a petition requesting the company to abandon the project.

- **Impact of machine learning on humans:** A machine learning based system such as a movie recommendation system changes the choice of human over time and narrows them with time. It is interesting to know that people don't notice how they get manipulated by algorithms. Examples include movie recommendation systems, news, propaganda, etc.
- **False correlation:** A false correlation comes into play when two parameters that are completely independent of each other show similar behavior. This creates an illusion that these parameters are somehow connected to each other. They are also known as **spurious correlation**. For example, if there is an increase in the number of car seat belts, there is a decrease in the number of astronaut deaths. This is a false correlation since a car seat belt has nothing to do with accidents occurring in space.
- **Feedback loops:** Feedback loops are worse than false correlations. It is a condition where the decision of an algorithm affects reality while convincing that the conclusion is correct. For example, a crime prevention program suggested that more police officials to be sent to a particular area on the basis of an increase in the crime rate. This led to the local residents reporting crimes more frequently as somebody was right there they can report them. This also led to the police officials writing more reports and implementing

protocols resulting in a higher crime rate, which meant that more police had to be sent to the area. Earlier, when police officials were not present in the area, people didn't report crimes frequently.

- **Poisoned or contaminated reference data:** The outcome of a machine learning algorithm purely depends on the reference data or training data that a machine learns. If the training data or reference data is poisoned or contaminated, then the outcome of machine learning will also be incorrect. For example, if we want to develop a machine translation system, and if the training file consists of incorrect translations, then the output will also be incorrect.
- **Trickery:** Even if a machine learning algorithm is working perfectly, it can be tricked. A noise or distortion can completely alter the outcome of the algorithm. In the near future, if a machine learning algorithm is used for the analyses of X-rays emitted from the luggage at the airport and an object is placed next to a gun, then the algorithm will not be able to detect the gun.
- **Mastering machine learning:** A data scientist is a person who has expertise in machine learning. Those who are not data scientists may not acquire all of the knowledge related to machine learning. They need to find the key issues in a particular domain of machine learning and then try to overcome these issues. For example, a person who is working on predictive modeling may not have a complete knowledge of a **Natural Language Processing** (**NLP**) task.
- **Wrong assumptions are drawn:** A machine learning based system needs to deal with missing values in the data sets. For example, the missing value issue can be resolved by using the mean value as the replacement to the missing value. Here, reliable assumptions need to be drawn related to the replacement of the missing values. So, we must make sure that the data doesn't come with the missing values and assumptions drawn are of substantial amount.
- **Machine learning based systems are still not intelligent:** While machine learning based systems are constantly evolving, there exists failure as well in the current machine learning based systems. For example, as an experiment,

Microsoft's chatbot Tay was released on Twitter that mimicked a teenage girl. It was a failure and consequently the company had to close the experiment and apologize to the whole internet crowd for the hurtful and offensive tweets by chatbot Tay.

- **Computational needs are expensive:** In order to perform large data processing, GPUs are used instead of CPUs. Some companies don't have GPUs, so it takes a longer time for the conventional CPUs to process large amounts of data. In some situations, even with GPUs, it may take days or weeks to complete the processing as compared to the traditional software development that may take a few minutes or hours to complete the task.

Applications

Machine learning is a buzzword today. There are numerous applications of machine learning. Some of the applications are shown in *Figure 1.2*:

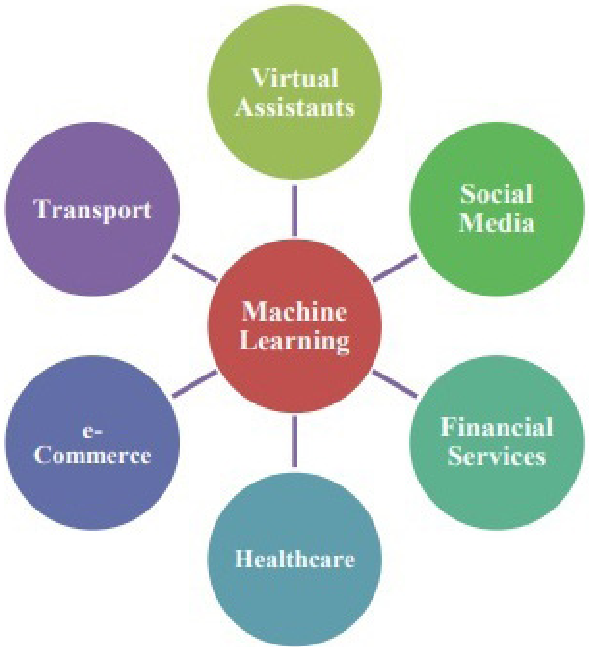


Figure 1.2: *Applications of machine learning*

- **Virtual personal assistants:** Some of the most popular examples of virtual personal assistants used today include Alexa, Siri, and Google Now. These virtual personal assistants help in finding information, whenever asked over voice. We can activate these virtual personal assistants and ask questions like "*Which are the flights from London to Germany?*", "*What are the tasks that need to be performed today?*" For answering such queries, virtual personal assistants collect information or search previously asked queries or collect information from phone apps. Machine learning is an integral part of virtual personal assistants as they collect information and refine it based on the previous information which is then used to generate results based on the given preferences. Virtual personal assistants are integrated to various platforms such as mobile apps (for example, Google Allo), smartphones (for example, Samsung Bixby on Samsung S8), smart speakers (for example, Amazon Echo, Google Home), etc. Virtual personal assistants are small, portable devices. Google Home is shown in *Figure 1.3.*

Figure 1.3: *Google Home*

- **Traffic prediction:** In order to manage traffic, GPS navigation devices are used. GPS devices track the current location and velocity of a vehicle, and store the information in the central server. This information is used to generate the current traffic report. This prevents traffic and helps in congestion analysis. A GPS device equipped in a car is shown in *Figure 1.4*. So, machine learning is used for estimating the areas where congestion can be found on the basis of daily GPS reports.

Figure 1.4: *A GPS device equipped in a car*

- **Online transportation networks:** When we book a cab using an app, it estimates the price of the ride. Machine learning is used to minimize the detour. It plays a very important role in predicting the travel time, price of the ride, and reduce detour.
- **Video surveillance system:** A single person cannot monitor multiple video cameras. A video surveillance system uses machine learning at its back end to detect unusual behavior in people, like napping, stumbling, standing, etc. The video surveillance system on detecting unusual behavior will alert a human attendant and prevent any mishap from taking place.
- **Social media services:** Machine learning is vastly used in social media platforms for personalizing news feed and targeting better ads. Other applications of machine learning in social media services include the following:
    - **People you may know:** Machine learning is based on the concept of gaining knowledge with experience. Facebook notices people that we connect with, the profiles that we often visit, our workplace, groups that we share, our interest, etc. On the basis of all this information, Facebook gives a suggestion of the list of people we would like to become friends with.
    - **Face recognition:** When we upload a photograph of ours along with a friend, Facebook can immediately recognize that friend. This is because Facebook

identifies unique features, poses, and projections and matches them with the photographs of the people in our friend list. It is a very complicated application at the back end as it considers the precision factor, but at the front end, it is a very simple application of machine learning. Facebook uses DeepFace, which is a deep learning project responsible for the identification of a person's face in images. Facebook also provides a feature of alternative tags for an image that is already uploaded on Facebook.

    - **Similar pins:** Computer vision involves extraction of useful information from videos and images. Machine learning is one of the applications of computer vision. Pinterest makes use of computer vision to detect pins or objects in an image and recommend pins which are similar to it.
    - **Sentiment analysis:** Sentiment analysis is one of the applications of machine learning. It is a process of determining emotion or the opinion of a person from the given text. It is also referred to as opinion mining or sentiment classification. The application of sentiment analysis is found in decision-making applications, review-based websites, etc.

- **Email spam and malware filtering:** Email clients use numerous spam filtering techniques. In order to ensure that these approaches are continuously updated, machine learning is used. A rule-based spam filtering technique does not track the latest tricks adopted by the spammers. Examples of a spam filtering technique that is influenced by machine learning are multilayer perceptron, C 4.5 Decision Tree Induction, etc. Every day, nearly 325,000 malwares are detected and about 90-98% of the code is similar to its previous version. The system security programs that are based on machine learning are able to the coding pattern. Hence, they are able to find malware with a variation of 2-10% and also provide protection against them.
- **Online customer support:** Many websites today provide the facility to chat with the company's customer support representatives while the user is scrolling the website. But not

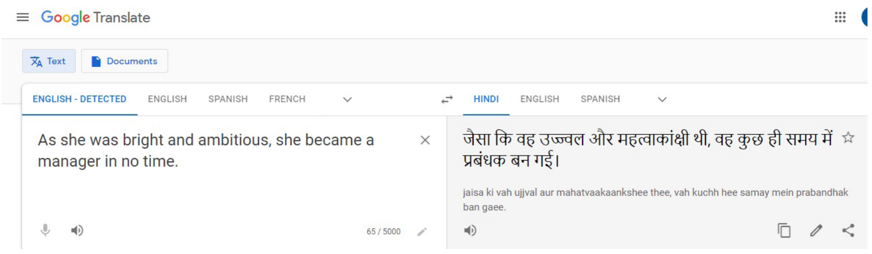
all the websites provide live executives to answer the queries. In some websites, the user talks to a chatbot. These chatbots extract useful information from the website and present it to the customer as a response to their query. Chatbots have advanced with time. They are based on a machine learning algorithm that gains knowledge from past experiences. Chatbots try to understand the user queries and then serve them with better responses every time.

- **Result refinement of a search engine:** Google and many other search engines make use of machine learning in order to improve the searching capability. Whenever we perform a search, an algorithm is run at the back end to see how we respond to the results provided by a search engine. If we open the top-most result and stay on the page for a very long time, then the search engine assumes that the result displayed is appropriate in accordance with the query. Also, if we reach the second or the third page of the search results but do not open any of the pages, then the search engine assumes that the result displayed is not in accordance with the query. In this way, the algorithm running at the back end tries to improve the performance of the search results.
- **Product recommendations:** When we shop an online product from a website, we notice that we start receiving emails containing shopping suggestions. Also, the app or the shopping website recommends items that match our choices. Product recommendations are displayed on the basis of past purchases, items browsed or added to the cart, or brand preferences. This magical shopping experience is due to machine learning.
- **Online fraud detection:** With the help of machine learning, we can track online financial frauds. For example, to prevent money laundering, Paypal uses the machine learning approach. Using certain tools, Paypal can compare millions of transactions occurring between buyers and sellers, and be able to distinguish between legal and illegal transactions.
- **Online gaming:** Machine learning is used in online gaming. For example, in Chess, it uses machine learning. On the basis of previous moves that gave winning situations, the algorithm running at the back end tries to improve its performance and

make similar moves that it considers the best. In this way, machine learning tries to imbibe human-like intelligence in computer that can play like any human chess champion.

- **Financial services:** According to the experts, online credit card fraud has risen to $32 billion in 2020. Companies that are related to the financial sector can see the flow of financial data and prevent financial fraud. This is possible using machine learning. Machine learning helps in the identification of opportunities in trading and investment. With the help of cyber surveillance, we can identify those institutions and individuals that are at financial risk and take timely action to prevent any cyber fraud.
- **Healthcare:** There has been an advancement in technology in the field of medical healthcare by the incorporation of machine learning. Wearable sensors and devices can provide real-time overall health conditions of a person, such as his heartbeat, blood pressure, etc. A doctor can use this information for analyzing the health condition of an individual, draw pattern using patient history, and predict any sort of ailment in the future. A machine may also be trained to have human-like intelligence. A machine may behave as a doctor and predict a disease or suggest medicine on the basis of past health condition records.
- **Oil and gas:** This industry requires the need of machine learning the most. Machine learning applications in oil and gas industry are vast. It not only includes streaming oil distribution but also finding new sources of energy and the analyzing of underground minerals.
- **Self-driving cars:** A self-driving car is one of the latest and most exciting applications of machine learning. Tesla is a famous car manufacturing firm, which is working on developing self-driving cars. It is building a self-driving car using an unsupervised machine learning algorithm that is able to detect objects and people while driving.
- **Automatic text translation:** Today, if you visit any new place, the language is not a barrier to understand the thoughts of the locals or to share your thoughts with them. With the help of machine learning, we can perform translation from one language into another, and also perform the conversion of text to speech and vice versa. **GNMT** (**Google Neural**

Machine Translation) is based on neural machine learning that performs the translation of text from one language into another language. GNMT is also referred to as automatic translation system. Complex English to Hindi translations given by Google Translator is shown in *Figure 1.5*:



As she was bright and ambitious, she became a manager in no time.

जैसा कि वह उज्ज्वल और महत्वाकांक्षी थी, वह कुछ ही समय में प्रबंधक बन गई।

jaisa ki vah ujjval aur mahatvaakaankshee thee, vah kuchh hee samay mein prabandhak ban gaee.

***Figure 1.5:** Complex English to Hindi translation by Google Translator*

- **Dynamic pricing:** Dynamic pricing refers to the pricing strategy that wholly depends on the objective thought. For example, plane ticket, movie ticket, cab fare, etc. are dynamically priced. Using machine learning, buying trends can be found out and the dynamic prices of products can be determined. Uber uses a machine learning model called *Geosurge* to perform the dynamic pricing of individual rides.
- **Classification of news:** Machine learning helps in the classification of vast news into different categories like National News, International News, Sports News, Entertainment, etc. These help the readers to choose the news from their desired category. Machine learning methods used for the classification of news are: Support Vector Machine, Naive Bayes, K-Nearest Neighbor, etc.
- **Information retrieval:** It is one of the significant applications of machine learning as it involves the extraction of knowledge or structured data from unstructured data. Information retrieval plays a crucial role in the big data sector. In the machine learning approach, unstructured data is taken as input and knowledge or structured data as output.
- **Robot control:** One of the applications of machine learning is robot control. Recently, research was carried out to obtain control over helicopter aerobatics and flight. In a robot control based competition sponsored by Darpa, a robot that drove in a desert for one hundred miles won over robot that could notice distant objects.

Conclusion

Machine learning is one of the applications of artificial intelligence that allows the system to automatically learn and improve its performance without being explicitly programmed. Machine learning involves the development of computer systems that can access data and use this data to perform learning by themselves. In this chapter, we have discussed about machine learning, the applications of machine learning, and the challenges of machine learning. In the next chapter, we will discuss about supervised learning and linear regression which is one of the types of supervised learning techniques.

Questions

1. Explain the difference between supervised learning and unsupervised learning with examples.
2. What are the challenges faced in adopting the machine learning technique?
3. Explain with an example how machine learning is used in the healthcare sector.
4. Explain with an example how machine learning is used in social media services.

CHAPTER 2 Linear Regression

Supervised learning is a machine learning task of mapping the input to the output on the basis of labeled input-output example pairs. Supervised learning may be of two types: classification and regression. In this chapter, we will discuss about linear regression in one variable, linear regression in multiple variables, gradient descent, and polynomial regression.

Structure

- Linear regression in one variable
- Linear regression in multiple variables
- Gradient descent
- Polynomial regression

Objectives

- Understanding linear regression in one variable
- Understanding linear regression in multiple variables

- Knowing gradient descent
- Understanding polynomial regression

Linear regression in one variable

Linear regression is a technique to depict the relationship between an independent variable *x* and a dependent variable *y*. Linear regression states that the relationship that exists between one or more input features and the relative output or target vector is approximately linear in nature. Linear regression finds the weighted sum of the input features along with the constant referred to as bias term or intercept. Linear regression has numerous real-life applications. These applications fall into two categories:

- If the application comprises forecasting, prediction, or error reduction, then linear regression may be applied to the data set values and make predictions in the response.
- When there is a need of variations in the response, then it may be attributed by the presence of other explanatory variables.

Linear regression is used to find possible relationships between the variables in the field of behavioral, biological, and social sciences. In linear regression with one variable, hypothesis is defined as:

$$h_\theta (x) = \theta_0+\theta_1{}^*x$$

Here, *x* is referred to as an independent variable on which depends our hypothesis. For example, '*Rainfall*' measured in mm could be *x* and '*The Number of Umbrellas sold*' could be the hypothesis that we are trying to predict. θ_0 and θ_1 are referred to as the bias variable and weight variable, respectively and they together constitute the weight matrix.

Cost function is an equation that gives an estimate of how close we are to the hypothesis. Smaller the value of cost function, closer we are from the required curve. So, we try to minimize the cost function in order to reduce error. The mean squared error cost function is defined as follows:

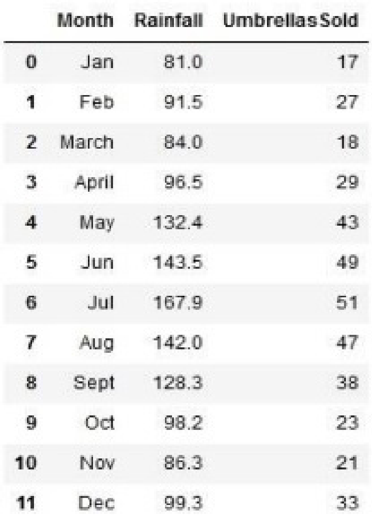
$$J(\theta_0, \theta_1) \ = \ 1/2m \sum i = 1 \ to \ \ m(h_\theta(x^i) - y^i)^2$$

Here, *J* is a cost function, *m* refers to the number of data points in our data set, and *y* refers to the actual values that we will like to predict.

To add normalization to the cost function, we introduce a constant *1/2m* to it.

The code for linear regression in one variable in Python is as follows:

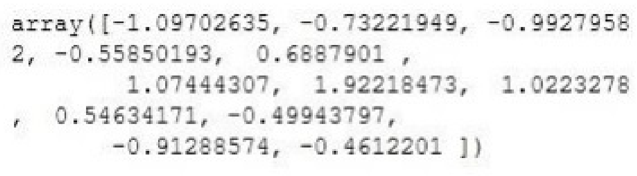
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import __future__
df=pd.read_csv("C:\\Users\\computer\\Desktop\\umbrella.
csv")  # Data is uploaded and a DataFrame is #created
using Pandas pd object
df
```



```
    Month  Rainfall  UmbrellasSold
0     Jan      81.0             17
1     Feb      91.5             27
2   March      84.0             18
3   April      96.5             29
4     May     132.4             43
5     Jun     143.5             49
6     Jul     167.9             51
7     Aug     142.0             47
8    Sept     128.3             38
9     Oct      98.2             23
10    Nov      86.3             21
11    Dec      99.3             33
```

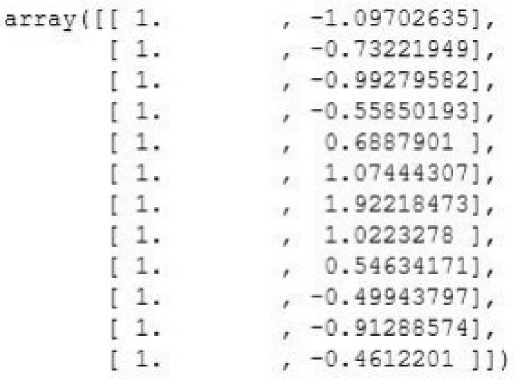
```
# The strength of the relationship is found between
Rainfall and Umbrellas Sold
X = np.asarray(df.Rainfall.values)
y = np.asarray(df.UmbrellasSold.values)
# Scaling and Normalization of the features are performed
def FeatureScalingNormalization(X):
# Xnorm is a copy of X vector
Xnorm = X
# avgx will the contain average value of x in the training
set
 avgx = np.zeros(X.shape[0])
```

```
# rangex will contain the standard deviation values of x
rangex = np.zeros(X.shape[0])
avgx = X.mean()
rangex = X.std(ddof=1)  # Calculated with NumPy. It
requires degreeoffreedom=1
# The number of training examples is stored in p
p = X.shape[0]
# a vector of size p with the average values of x
avgx_matrix = np.multiply(np.ones(p), avgx).T
# a vector of size p with the standard deviation values
rangex_matrix = np.multiply(np.ones(p), rangex).T
# Normalization is applied on x values
Xnorm = np.subtract(X, avgx).T
Xnorm = Xnorm /rangex.T
return [Xnorm, avgx, rangex]
featuresNormalizeresults = FeatureScalingNormalization(X)
# normalized X matrix is obtained
X = np.asarray(featuresNormalizeresults[0]).T
# mean values are obtained
avgx = featuresNormalizeresults[1]
# standard deviation values are obtained
rangex = featuresNormalizeresults[2]
X
```



```
array([-1.09702635, -0.73221949, -0.9927958
2, -0.55850193,  0.6887901 ,
        1.07444307,  1.92218473,  1.0223278
,  0.54634171, -0.49943797,
       -0.91288574, -0.4612201 ])
```

```
p = len(y) # number of training examples
X = np.vstack((np.ones(p), X.T)).T # Training Examples,
column of 1's is added to X
X
```



```
array([[ 1.        , -1.09702635],
       [ 1.        , -0.73221949],
       [ 1.        , -0.99279582],
       [ 1.        , -0.55850193],
       [ 1.        ,  0.6887901 ],
       [ 1.        ,  1.07444307],
       [ 1.        ,  1.92218473],
       [ 1.        ,  1.0223278 ],
       [ 1.        ,  0.54634171],
       [ 1.        , -0.49943797],
       [ 1.        , -0.91288574],
       [ 1.        , -0.4612201 ]])
```

```
plt.scatter(X[:,[1]], y,  color='blue') # Data is plotted
and the Scatter plot is obtained
plt.xlabel("Rainfall")
plt.ylabel("Umbrellas Sold")
```

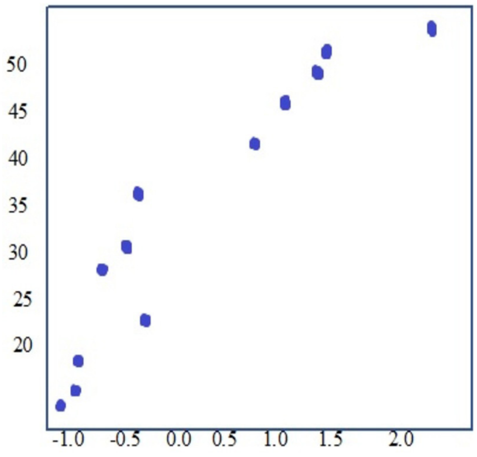


Figure 2.1: *Linear regression plot showing the relationship between Umbrellas Sold and Rainfall*

In the above graph (*Figure 2.1*), we can visualize an increasing pattern in the relationship between rainfall and the umbrellas sold.

```
# We calculate the plot when two parameters, Theta is
randomly chosen as [140.0,5.0]
theta_0 = 140.0
theta_1 = 5.0
theta = np.asarray([theta_0,theta_1]).astype(float)
```

```
# Plot the data
plt.scatter(X[:,[1]], y,  color='black')
# corresponding to the Hypothesis model, the red line is
plotted.
plt.plot(X[:,[1]], np.sum(np.multiply(X,theta), axis=1),
color='red', linewidth=1)
plt.xlabel("Rainfall")
plt.ylabel("Umbrellas Sold")
```

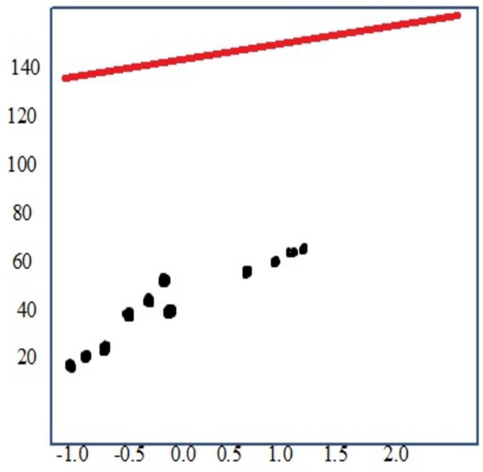


Figure 2.2: *Linear regression plot when theta is randomly chosen*

```
# Calculate the Cost Function using 3 values in a Data set.
We have taken random values of theta as [120.0, 10.0].
X1=X[0:3]
y1=y[0:3]
m1=3
theta_0 = 120.0
theta_1 = 10.0
theta = np.asarray([theta_0,theta_1]).astype(float)
plt.scatter(X1[:,[1]], y1,  color='blue')
plt.plot(X1[:,[1]], np.sum(np.multiply(X1,theta), axis=1),
color='red', linewidth=1)
# Plot red points corresponding to the predicted values.
plt.scatter(X1[:,[1]], np.sum(np.multiply(X1,theta),
axis=1),  color='red')
plt.xlabel("Rainfall")
plt.ylabel("Umbrella Sold")plt.ylabel("Umbrellas Sold")
```

```
Text(0, 0.5, 'Umbrella Sold')
```

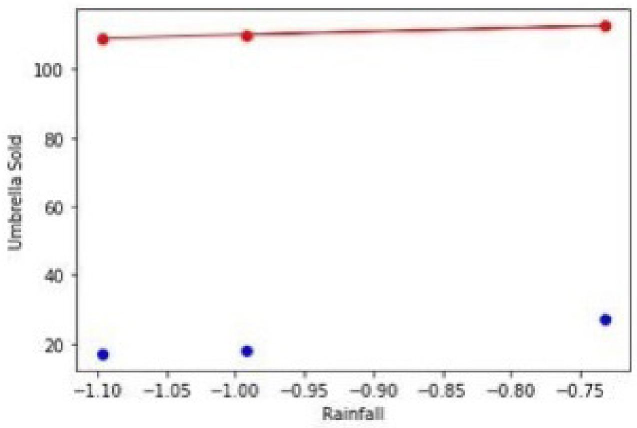
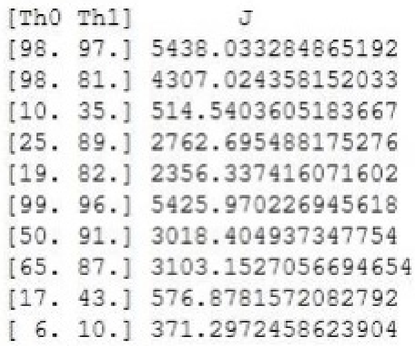


Figure 2.3: *Linear regression plot for the calculation of cost function*

As seen in the above graph (*Figure 2.3*), the blue dots show the value of y as `[18.0,20.0,25.0]`.

```
# Cost Function is Calculated
def calcCostFunction(X, y, theta):
    # number of training examples
    p = len(y)
    # Cost J is initialized
    J = 0
    # Calculate h = X * theta
    h = np.sum(np.multiply(X, theta), axis=1)
    # Squared Error = (h - y)^2 (vectorized)
    SquaredError = np.power(np.subtract(h,y), 2)

    # Calculate the Cost J
    J = 1/(2*p) * np.sum(SquaredError)
    return J
calcCostFunction(X,y,theta)
```

```
3791.297245862391


# The following code calculates 10 random theta values and
generates corresponding Cost Function values.
import random # import the random library
print("[Th0 Th1]", "\tJ")
for x in range(10):
    theta_0 = random.randint(1,101)
    theta_1 = random.randint(1,101)
    theta = np.asarray([theta_0, theta_1]).astype(float)
    # Calculate J and print the table
    print(theta, calcCostFunction(X, y, theta))
```



```
[Th0 Th1]        J
[98. 97.] 5438.033284865192
[98. 81.] 4307.024358152033
[10. 35.] 514.5403605183667
[25. 89.] 2762.695488175276
[19. 82.] 2356.337416071602
[99. 96.] 5425.970226945618
[50. 91.] 3018.404937347754
[65. 87.] 3103.1527056694654
[17. 43.] 576.8781572082792
[ 6. 10.] 371.2972458623904
```

In the above output, we find `10 J` values corresponding to 10 randomly chosen theta values. We need an algorithm that can minimize the value of `J` for the given theta values. We can find the minimum value of `J` using the gradient descent algorithm.

Linear regression in multiple variables

Linear regression in multiple variables explains the relationship between a dependent variable *y* and many independent variables. It may be represented as follows:

$$Y = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 \dots\dots \theta_n X_n$$

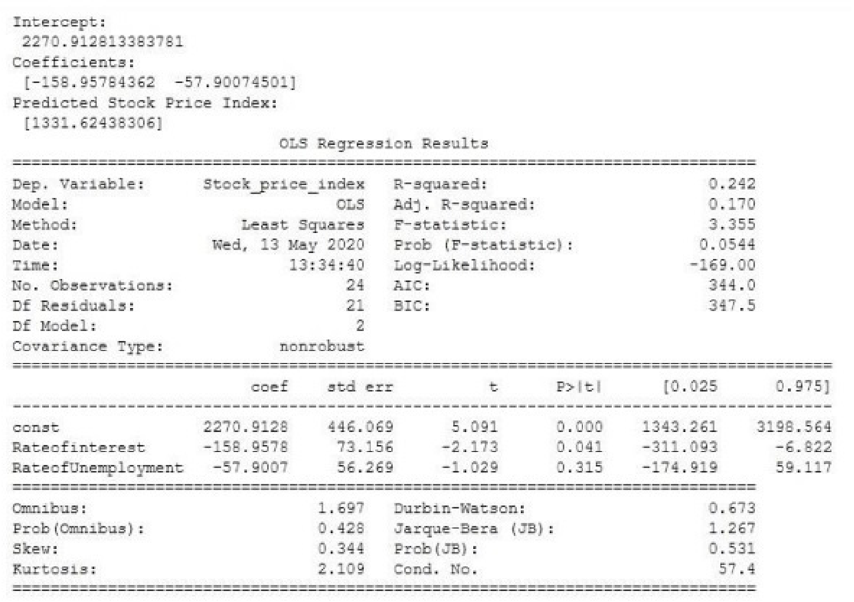
Instead of a 'Yes' or 'No' reply, the value of *Y* will be a number. It is a continuous dependent variable. Here, the theta values are referred to as regression weights and computed in such a way so as to minimize the sum of the squared deviations.

Consider the following code that implements linear regression in multiple variables using `scikitlearn` and `statsmodels`:

```
import pandas as pd

from sklearn import linear_model

import statsmodels.api as sm


Stock_Market = {'Year': [2018,2019,2018,2017,2017,2016,201
7,2019,2018,2018,2019,2019,2016,2017,2017,2018,2018,2018,2
018,2018,2016,2016,2016,2016], 'Month': [10, 12,10,9,8,4,6
,5,7,5,1,2,12,10,11,12,8,9,6,4,5,1,3,2],  'Rateofinterest':
[3.25,4.5,3.5,5.5,4.5,4.5,3.5,5.25,6.25,4.25,5,5,6,5.7
5, 4.75, 5.75,5.75,4.75,5.75,4.75,3.75,4.75,5.75,5.75
],  'RateofUnemployment':[7.3,4.3,3.3,4.3,6.4,5.6,4.5
,5.5,6.5,4.6,6.7,5.9,6,4.9,6.8,5.1,7.2,5.
1,6.1,7 .1,6.9,5.2,7.2,5.1],'Stock_price_
index':[1764,1594,1457,1493,1756,1244,1254,1175,13 29,1 54
7,1230,1375,1057,945,947,958,918,935,834,786,815,852,724,7
49]    }

df = pd.DataFrame(Stock_
Market,columns=['Year','Month','Rateofinterest',
'RateofUnemployment','Stock_price_index'])

X = df[['Rateofinterest','RateofUnemployment']] # here we
have used 2 variables in Linear Regression using #Multiple
Variables

Y = df['Stock_price_index']

# Using sklearn

regr = linear_model.LinearRegression()

regr.fit(X, Y)

print('Intercept: \n', regr.intercept_)

print('Coefficients: \n', regr.coef_)

# prediction using sklearn

New_Rateofinterest = 3.25

New_RateofUnemployment = 7.3

print ('Predicted Stock Price Index: \n', regr.
predict([[New_Rateofinterest ,New_RateofUnemployment]]))
```

```
# using statsmodels
X = sm.add_constant(X) # adding a constant
model = sm.OLS(Y, X).fit()
predictions = model.predict(X)
printmodel = model.summary()
print(printmodel)
```

The output of the above code is shown as follows:



```
Intercept:
 2270.912813383781
Coefficients:
 [-158.95784362  -57.90074501]
Predicted Stock Price Index:
 [1331.62438306]
                            OLS Regression Results
==============================================================================
Dep. Variable:      Stock_price_index   R-squared:                       0.242
Model:                            OLS   Adj. R-squared:                  0.170
Method:                 Least Squares   F-statistic:                     3.355
Date:                Wed, 13 May 2020   Prob (F-statistic):             0.0544
Time:                        13:34:40   Log-Likelihood:                -169.00
No. Observations:                  24   AIC:                             344.0
Df Residuals:                      21   BIC:                             347.5
Df Model:                           2
Covariance Type:            nonrobust
======================================================================================
                        coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------
const              2270.9128    446.069      5.091      0.000    1343.261    3198.564
Rateofinterest     -158.9578     73.156     -2.173      0.041    -311.093      -6.822
RateofUnemployment  -57.9007     56.269     -1.029      0.315    -174.919      59.117
==============================================================================
Omnibus:                        1.697   Durbin-Watson:                   0.673
Prob(Omnibus):                  0.428   Jarque-Bera (JB):                1.267
Skew:                           0.344   Prob(JB):                        0.531
Kurtosis:                       2.109   Cond. No.                         57.4
==============================================================================
```

In the above output, it shows the values of intercept, coefficient, and predicted stock price index using `sklearn` and `statsmodels`:

StockPrice_Index = Intercept + (Rate of interest_Coefficient)X_1 + (Rate of Unemployment_Coefficient)*X_2*

Substituting the values of intercept and coefficient from `sklearn`, we get:

Stock Price_Index=2270.9128 + (-158.9578)X_1 + (-57.9007)*X_2*

From the table above, we infer that we get the same values of intercept, coefficient, and predicted stock price index using `sklearn` and `statsmodels`.

Gradient descent

Gradient descent is a method to minimize the value of Cost Function. It can alter the values of Theta 0 and Theta 1 of a point based on the slope or gradient of the Cost Function curve. The changes introduced in the values of Theta 0 and Theta 1 also bring changes to the hypothesis, thereby bringing a better fit to the data. Gradient descent estimates the derivative of cost function. It is represented by the following formula:

$$\theta^{new} = \theta^{old} - \alpha 1/m \sum i = 1\ to\ m\,[(h_\theta X - y)X^T]$$

Here, h_θ X-y is referred to as error.

is referred to as the learning rate. In this method, we first fetch and clean the data and analyze it. Then, we define the hypothesis, regression parameters, and cost function. Then, we run the gradient descent algorithm on all the data points and update the hypothesis. This algorithm is implemented in Python as follows:

```
def gradientDescent(X, y, theta, alpha, numiters):
    # number of training examples
    p = len(y)
    # Initialize J_history and Theta_history
    Jhistory = []
    Thetahistory = []
    for i in range(numiters):
        # Calculate h = X * theta
        h = np.sum(np.multiply(X,theta), axis=1)
        # Calculate the error = (h - y)
        error = np.subtract(h, y)
        # Calculate the new theta
        thetanew = alpha * 1/p * np.sum(np.multiply(X.T, 
error), axis=1)
        # Update theta
        theta = np.subtract(theta, thetanew)
        # Collect all the theta and J
        Thetahistory.append(theta.tolist())
        Jhistory.append(calcCostFunction(X,y,theta).
tolist())
    return theta, Thetahistory, Jhistory
# Running the Gradient Descent
# Initialize theta
theta = np.asarray([0,0]).astype(float)
# Set the number of iterations for the Gradient Descent
iterations = 2000
# Set the Learning Rate
alpha = 0.01
# Run the gradientDescent() function, and collect the
output in "results"
results = gradientDescent(X, y, theta, alpha, iterations)
# Get the theta from the results
theta = results[0] # new theta
# Get the theta history
Thetahistory = results[1] # Theta history
# Get the J history
Jhistory = results[2] # Cost function history
plt.scatter(X[:,[1]], y,  color='blue')
# Plot Hypothesis (theta as calculated with the Gradient
Descent)
plt.plot(X[:,[1]], np.sum(np.multiply(X,theta), axis=1),
color='red', linewidth=1)
plt.xlabel("Rainfall")
plt.ylabel("Umbrellas Sold")
```

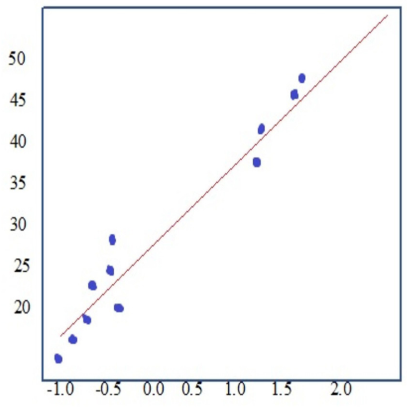


Figure 2.4: *Plot between rainfall and umbrellas*

Now, we'll plot the `Theta` history as follows:

```
theta_0 = np.asarray(Thetahistory)[:,[0]]
theta_1 = np.asarray(Thetahistory)[:,[1]]
plt.plot(theta_0[0:len(theta_0)], color='red', linewidth=1)
plt.plot(theta_1[0:len(theta_1)], color='green',
linewidth=1)
plt.xlabel("Iterations")
plt.ylabel("theta")
```

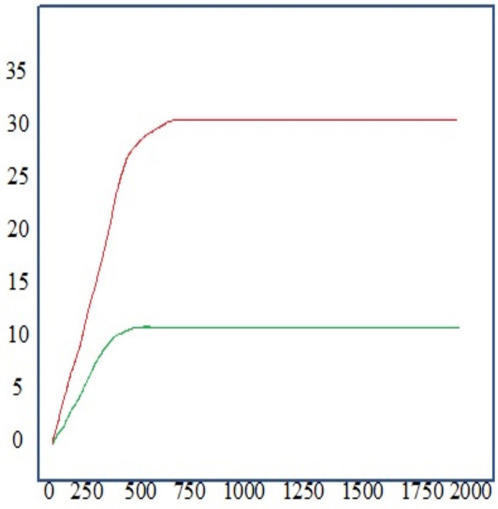


Figure 2.5: *Plot of Theta0 and Theta1 values*

We have performed `2000` iterations. In the above figure, the red curve represents θ0 and the green curve represents θ1. After 2000 iterations, the value of `θ` is `[37.5,11.0]`. Next, we'll plot the `J` history as follows:

```
plt.plot(Jhistory[0:len(Jhistory)], color='blue',
linewidth=1)
# Put labels
```

```
plt.xlabel("Iterations")
plt.ylabel("J")
```

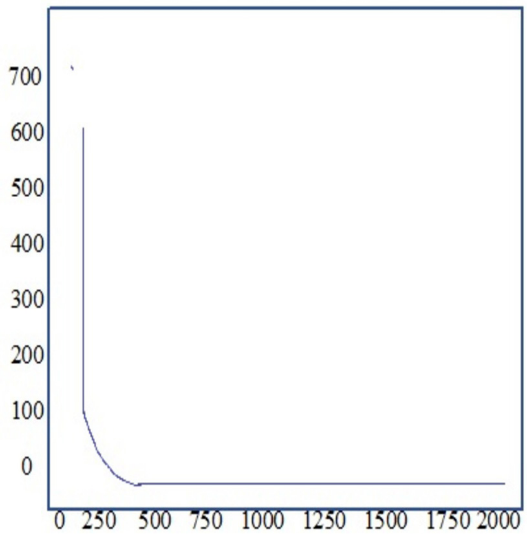


Figure 2.6: *J history plot*

The above graph is a `J` history plot that shows the value of cost function falls down after `200` iterations and then becomes stable till `1500` iterations. The minimum value of `J` is `57.5`:

```
# Now if we predict the rainfall in mm to be 82mm, we
predict the corresponding number of umbrellas sold.
query = [1, 82]
# Scale and Normalize the query
queryNormalized = [1, ((query[1]-avgx)/rangex)]
prediction = np.sum(np.multiply(queryNormalized, theta))
prediction
20.373966510153515
```

So, approximately 20 umbrellas are sold:

```
# Drawing a contour plot of J and θ
nslice = 50
theta0_vals = np.linspace(-100, 200, num=nslice)
theta1_vals = np.linspace(-400, 400, num=nslice)
# Initialize J_val that will collect all the J values
# calculated by calcCostFunction()
J_vals = []
for i in range(len(theta0_vals)):
    for j in range(len(theta1_vals)):
        t = np.asarray([theta0_vals[i], theta1_vals[j]]).
astype(float)
        J_vals.append(calcCostFunction(X, y, t).tolist())
J_vals = np.asarray(J_vals)
J_vals = J_vals.reshape((nslice, nslice)).T
levels = nslice
s = 1
# plot the contour with theta and the J values
plt.contour(theta0_vals, theta1_vals, J_vals, levels)
# Draw the path of the Gradient Descent convergence
for k in range(0, iterations, 10):
    plt.scatter(np.asarray(Thetahistory)[k][0],\
          np.asarray(Thetahistory)[k][1], color='blue',
s=s)
# Draw a red dot i correspondence of the theta associated
to the minimum J
plt.scatter(theta[0], theta[1], color='red', s=10)
plt.xlabel("theta_0")
plt.ylabel("theta_1")
```

```
Text(0, 0.5, 'theta_1')
```

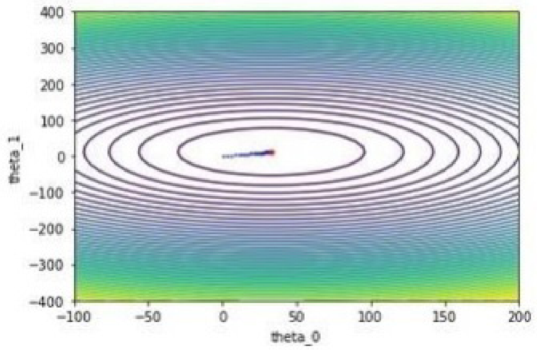


Figure 2.7: Contour plot of J and Theta

In the above plot, the red dot at the center represents minimum value of J, J=57.5 at θ=[37.5,11.0].

Polynomial regression

Polynomial regression is a type of regression analysis in which the relationship that exists between a dependent variable y and an independent variable x is modeled by the nth degree of the polynomial x. It fits a nonlinear relation that exists between x and its corresponding conditional mean of y. This is denoted as E(y|x). In some of the cases, for the unknown parameters that are collected from the data, E(y|x) is linear. So, polynomial regression is referred to as a special case of linear regression with multiple variables.

Polynomial regression is represented as follows:

$$Y= \theta_0+\theta_1 X+ \theta_2 X^2+ \theta_3 X^3.......\theta_n X^n$$

Consider the following code in Python on polynomial regression:

```
import operator
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
np.random.seed(0)
x = 1 - 3 * np.random.normal(0, 1, 50)
y = x - 3 * (x ** 2) + 0.5 * (x ** 3) + np.random.
normal(-3, 3, 50)
# Data is transformed to include another axis
x = x[:, np.newaxis]
y = y[:, np.newaxis]
polyfeatures= PolynomialFeatures(degree=3)
xpoly = polyfeatures.fit_transform(x)
model = LinearRegression()
model.fit(xpoly, y)
ypoly_pred = model.predict(xpoly)
```

```
rmse = np.sqrt(mean_squared_error(y,ypoly_pred))
r2 = r2_score(y,ypoly_pred)
print(rmse)
print(r2)
plt.scatter(x, y, s=20)
# The values of x are sorted according to the degree
before the line plot
sort_axis = operator.itemgetter(0)
sorted_zip = sorted(zip(x,ypoly_pred), key=sort_axis)
x, y_poly_pred = zip(*sorted_zip)
plt.plot(x, ypoly_pred, color='m')
plt.show()
```

In the above code, we performed a polynomial regression, where the degree of the independent variable *x* is 2. We also plotted the graph and generated the RMSE and the *R2* Score of the resultant curve. The output of the above code is as follows:

```
2.5898697824172037
0.9974338440099649
```

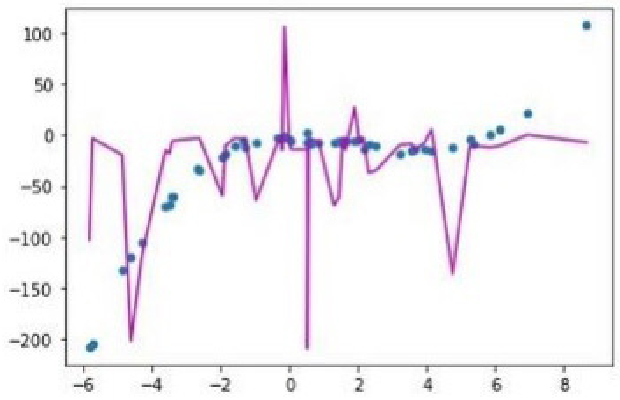


Figure 2.8: *Plot on polynomial regression*

The RMSE score and the *R2* Score obtained are `2.5898697824172037` and `0.9974338440099649`, respectively.

Conclusion

In this chapter, we discussed about linear regression in one variable, linear regression in multiple variables, gradient descent, and polynomial regression. We also discussed their implementation in python using `scikitlearn`, `statsmodels`, etc. In the next chapter, we will discuss about the classification using logistic regression.

Questions

1. Explain linear regression in one variable using an example.
2. Explain linear regression in multiple variables using an example.
3. Explain polynomial regression with an example.
4. Explain gradient descent with an example.

CHAPTER 3
Classification Using Logistic Regression

Introduction

Logistic regression is a kind of regression analysis that comes into play when the dependent variable is binary. It is a kind of predictive analysis that explains the relationship between one dependent variable and one or several nominal, interval, ordinal, or ratio-level independent variables. In this chapter, we will discuss about binary classification, logistic regression, and multi-class classification.

Structure

- Binary classification
- Logistic regression
- Multi-class classification

Objectives

- Understanding binary classification
- Understanding logistic regression

- Understanding multi-class classification

Binary classification

Binary classification, also referred to as binomial classification, may be defined as the process of classification of elements into two groups on the basis of the classification rule. It is found in the following fields:

- In the medical field, it is used to test whether a patient has any disease or not.
- In factories, it is used to decide whether a product is in accordance with the quality standards or not, or to check if some specification is met or not.
- In information retrieval, it is used in deciding whether a given page or an article should be in the result set of a search or not on the basis of the relevance of an article or the usefulness to the user.
- In spam filtering, it is used in deciding whether an email message received is a spam mail or not.
- It is also used in credit card fraudulent transaction detection.

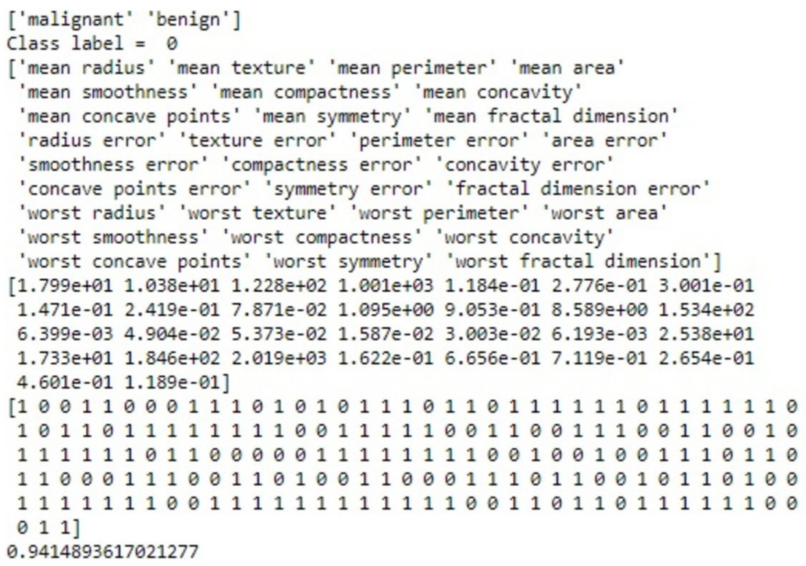
Some of the techniques used for learning binary classifiers include the following:

- Decision trees
- Neural networks
- Support vector machines
- Bayesian classification

Consider the following code in Python that performs a binary classification. Here, we have considered `breast_cancer`, which is a predefined data set in `sklearn`. This data set comprises 569 instances of tumors, 30 features or attributes such as texture, radius, area, and the smoothness of the tumor. It comprises two classes of tumors such as malignant and benign. Malignant tumors are represented by 0 and benign tumors are represented by 1:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
# Loading the predefined iris data set from Sklearn
data = load_breast_cancer()
# Organizing the data
labelnames = data['target_names']
labels = data['target']
featurenames = data['feature_names']
features = data['data']
# data
print(labelnames)
print('Class label = ', labels[0])
print(featurenames)
print(features[0])
# Splitting the data
train, test, train_labels, test_labels = train_test_
split(features,labels,test_size=0.33,random_state=42)
# Initializing the classifier
gnb = GaussianNB()
# Training the classifier
model = gnb.fit(train, train_labels)
# Making predictions
prediction = gnb.predict(test)
print(prediction)
# Evaluating the accuracy
print(accuracy_score(test_labels, prediction))
```

The output of the above code is shown below:



```
['malignant' 'benign']
Class label =  0
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
 1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
 6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
 1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
 4.601e-01 1.189e-01]
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 0 0 1 1 0 0 1 1 1 0 0 1 1 0 0 1 0
 1 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 1 0
 1 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 1 0 1 0 0
 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 1 0 1 1 1 1 1 1 0 0
 0 1 1]
0.9414893617021277
```

From the above code, we infer that the first data instance represents 0. So, it is a malignant tumor and its mean radius is *1.799e+01*. There are different testing analysis methods used in binary classification. Consider the following testing data on which we will apply the testing analysis methods:

Instance	Target	Outcome
1	1	0.89
2	1	0.75
3	1	0.60
4	1	0.50
5	0	0.45
6	1	0.44
7	0	0.42
8	1	0.41
9	0	0.42
10	1	0.39
11	1	0.31
12	0	0.30
13	0	0.18

Instance	Target	Outcome
14	0	0.17
15	0	0.16
16	0	0.15
17	0	0.14
18	0	0.13
19	0	0.10
20	0	0.10

Let us now consider the following testing analysis methods:

- **Confusion matrix**: It is used to present the performance of a binary classifier. The decision threshold, *T* may be used to find out whether the given set of instances is positive or negative. If the probability allotted to the given instance of a classifier is greater than *T*, then it is referred to as *positive* otherwise, it is referred to as *negative*. When all the given testing instances are classified, then the target labels are compared with the outcome labels to generate the following four terms:
    - **True positives (TP)** – The total number of instances that are positive and are classified as positive
    - **False positives (FP)** – The total number of instances that are negative and are classified as positive
    - **False negatives (FN)** – The total number of instances that are positive and are classified as negative
    - **True negatives (TN)** – The total number of instances that are negative and are classified as negative

  The confusion matrix is represented as follows:

	Predicted Positive	Predicted Negative
Real Positive	True Positives	False Negatives
Real Negative	False Positives	True Negatives

Here, the column represents the outcome classes and the rows represent the target classes. The diagonal cells show the number of cases that are classified correctly and the off-diagonal cells show the number of cases that are not classified correctly.

Let the value of the decision threshold be *T=0.4*. Consider the testing data table given above. We get the following confusion matrix:

	Predicted Positive	**Predicted Negative**
Real Positive	6	2
Real Negative	3	9

- **Binary classification tests**: These are the parameters that are derived from the confusion matrix. They involve the following parameters:
  - **Classification accuracy:** It may be stated as the ratio of the instances that are correctly classified. It can be depicted as follows:

$$Classification\_Accuracy = \frac{(True\_Positives + True\_Negatives)}{(Total\ instances)}$$

  - **Error rate:** It may be stated as the ratio of instances that will not be classified correctly. It can be depicted as follows:

$$Error\ Rate= \frac{(False\_Positives + False\_Negatives)}{(Total\ instances)}$$

  - **Sensitivity:** It can be stated as the ratio of the correct positives and the total number of positives. It is also referred to as recall or true positive rate. It can be depicted as follows:

$$Sensitivity= \frac{(True\_Positives\ )}{(Total\ Positive\ instances)}$$

  - **Specificity:** It can be stated as the ratio of the correct negatives and the total number of negatives. It is also referred to as the true negative rate. It can be depicted as follows:

$$Specificity = \frac{(True\_Negatives)}{(Total\ Negative\ instances)}$$

From the above mentioned confusion matrix, we obtain 25% error rate, 75% accuracy, 75% sensitivity, and 75% specificity.

Logistic regression

Logistic regression is a statistical model that makes use of the logistic function to model a binary dependent variable. It may be defined as the transformation of linear regression *mot del* that can probabilistically model the binary variables.

It may be represented by the following probability:

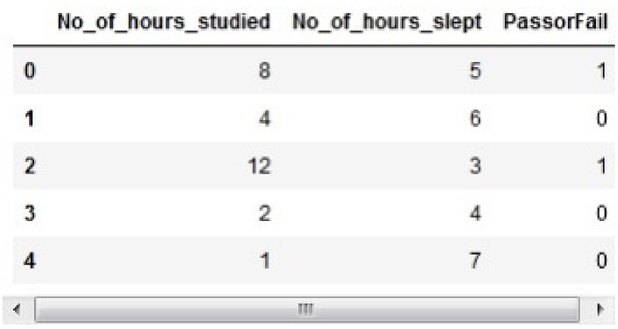
$$P = \frac{1}{1 + e^{-(\theta 0 + \theta 1X1 + \cdots . \theta nXn)}} = \frac{1}{1 + e^{-(\theta 0 + \Sigma \theta iXi)}}$$

The output in linear regression is continuous, whereas in logistic regression, the output is discrete. Assumptions of logistic regression include the following:

- In logistic regression, the dependent variable should be binary.
- In logistic regression, Linearity must exist between `logit` and independent variables.
- There is no chance of multicollinearity.
- It requires a large sample size.
- It remove outliers and misclassified instances from the training data. Logistic regression assumes that there is no error in the output variable.

Consider an example of a logistic regression, given number of hours, a student studied for exams and number of hours, a student slept. We have to predict whether a student will pass (represented by 1) or fail (represented by 0). It is represented by the following Python code:

```
import pandas as pd

import numpy as np

from sklearn import metrics

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

df = pd.read_csv("C:\\Users\\computer\\Desktop\\student.
csv")

df.head()
```



	No_of_hours_studied	No_of_hours_slept	PassorFail
0	8	5	1
1	4	6	0
2	12	3	1
3	2	4	0
4	1	7	0

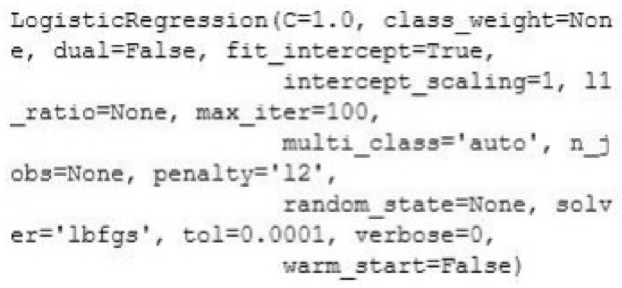
```
x = df.drop("PassorFail",axis = 1)
y = df.PassorFail
```

In the above code, the `drop` method is used to remove `PassorFail` attribute from `x` and, `x` is assigned to other attributes. `PassorFail` attribute is assigned to `y`.

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
random_state=4)
```

The above code splits the data set into 75% training data and 25% testing data:

```
logistic_regression = LogisticRegression()
logistic_regression.fit(x_train,y_train)
```



```
LogisticRegression(C=1.0, class_weight=Non
e, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1
_ratio=None, max_iter=100,
                   multi_class='auto', n_j
obs=None, penalty='l2',
                   random_state=None, solv
er='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Here, the `fit` method is used to train the model. The `predict` method is used to perform predictions on `x_test` values. The output of the prediction is stored in `y_prediction`. The `accuracy_score` method of metrics class is used to estimate the accuracy of the model. The accuracy obtained in the logistic regression model is 50%.

```
y_prediction = logistic_regression.predict(x_test)
accuracy = metrics.accuracy_score(y_test, y_prediction)
```

```
accuracy_percentage = 100 * accuracy
accuracy_percentage
50.0
first_student = logistic_regression.predict((np.array([7,
5]).reshape(1, -1)))
first_student
array([1], dtype= int64)
```

In the above code, we predict whether the first student will pass or fail on the basis of the number of hours studied; the number of hours slept is 7 and 5. The prediction result is that the first student will pass as indicated by the `array([1],dtype=int64)`. Also, we predict whether the second student will pass or fail on the basis of the number of hours studied; the number of hours slept is 2 and 10. The prediction result is that the second student will fail as indicated by the `array([0],dtype=int64)`.

```
second_student = logistic_regression.predict((np.array([2,
10]).reshape(1, -1)))
second_student
array([0], dtype=int64)
```

Multiclass classification

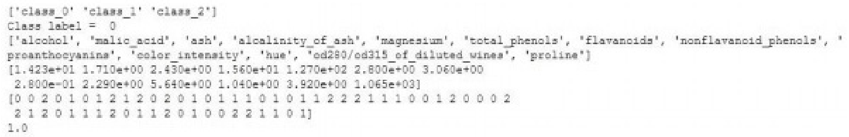
Multinomial or multiclass classification may be defined as the problem of the classification of instances into two or more classes. Multiclass classification makes sure that each sample is assigned only one label.

Consider the following Python code on multiclass classification. Here, we use a predefined data set `wine` from `sklearn`. This data set comprises 178 instances of wine and 13 features or attributes related to the wine, such as the amount of alcohol, malic acid, ash, alkalinity of ash, the amount of magnesium, total phenol, flavonoids, non-flavonoid phenols, color intensity, hue, etc.

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
# Predefined Sklearn Wine data set is loaded
```

```
data = load_wine()
# The data is organized
labelnames = data['target_names']
labels = data['target']
featurenames = data['feature_names']
features = data['data']
# Look at our data
print(labelnames)
print('Class label = ', labels[0])
print(featurenames)
print(features[0])
# Splitting our data
train, test, train_labels, test_labels = train_test_
split(features,labels,test_size=0.33,random_state=42)
# Initializing our classifier
gnb = GaussianNB()
# Training our classifier
model = gnb.fit(train, train_labels)
# Making predictions
prediction = gnb.predict(test)
print(prediction)
# Evaluate the accuracy
print(accuracy_score(test_labels, prediction))
```

The following is the output obtained from the above code:

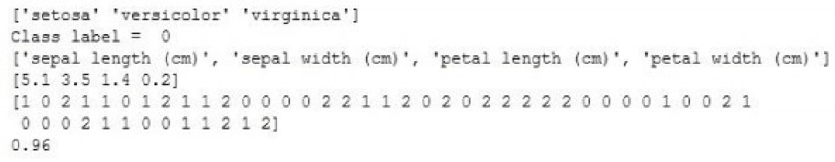


```
['class_0' 'class_1' 'class_2']
Class label =  0
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid_phenols', '
proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']
[1.423e+01 1.710e+00 2.430e+00 1.560e+01 1.270e+02 2.800e+00 3.060e+00
 2.800e-01 2.290e+00 5.640e+00 1.040e+00 3.920e+00 1.065e+03]
[0 0 2 0 1 0 1 2 1 2 0 2 0 1 0 1 1 1 0 1 0 1 1 2 2 2 1 1 1 0 0 1 2 0 0 0 2
 2 1 2 0 1 1 1 2 0 1 1 2 0 1 0 0 2 2 1 1 0 1]
1.0
```

In the above code, it shows that sample 1 is of class 0 and the amount of alcohol in it is *1.423e+01*. Here, the sample is classified into 3 classes of wines, such as class 0, class 1 and class 2.

Consider another code on multiclass classification. Here, we have taken iris, a predefine data set from `sklearn`. The iris data set comprises 150 instances of iris (a kind of flower) and 4 features or attributes of iris, such as the sepal length, sepal width, petal length, and petal width:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
# Loading the predefined iris data set from Sklearn
data = load_iris()
# Organizing the data
labelnames = data['target_names']
labels = data['target']
featurenames = data['feature_names']
features = data['data']
# data
print(labelnames)
print('Class label = ', labels[0])
print(featurenames)
print(features[0])
# Splitting the data
train, test, train_labels, test_labels = train_test_
split(features,labels, test_size=0.33, random_state=42)
# Initializing the classifier
gnb = GaussianNB()
# Training the classifier
model = gnb.fit(train, train_labels)
# Making predictions
prediction = gnb.predict(test)
print(prediction)
# Evaluating the accuracy
print(accuracy_score(test_labels, prediction))
```



```
['setosa' 'versicolor' 'virginica']
Class label =  0
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[5.1 3.5 1.4 0.2]
[1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 2 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0 1 1 2 1 2]
0.96
```

The above code divides the sample into 3 classes of iris, such as `setosa`, `versicolor`, and `virginica`. It shows that the first sample of iris is of type `setosa` and its sepal length is `5.1` cm.

Conclusion

In this chapter, we discussed about binary classification, logistic regression, and multi-class classification. We also discussed their implementation in Python using `scikitlearn`. In the next chapter, we will discuss about overfitting and regularization in linear regression, and overfitting and regularization in logistic regression.

Questions

1. Explain binary classification with an example.
2. Explain logistic regression using an example.
3. Explain multiclass classification with an example.

Chapter 4
Overfitting and Regularization

Logistic regression is a regression analysis that occurs when the dependent variable is binary. Binary classification involves the classification of elements on the basis of the classification rule into two groups. In the previous chapter, we discussed about binary classification, logistic regression, and multi class classification and their implementation in Python using `scikitlearn`. Overfitting is one of the common problems in machine learning, and it can be handled using regularization. In this chapter, we will discuss about overfitting and regularization in linear regression and overfitting and regularization in logistic regression.

Structure

- Overfitting and regularization in linear regression
- Overfitting and regularization in logistic regression

Objectives

- Understanding overfitting and regularization in linear regression

- Understanding overfitting and regularization in logistic regression

Overfitting and regularization in linear regression

In a linear model, a straight line is drawn in a prediction model. A linear model can be depicted using a single attribute that can predict a value or multiple attributes that can predict a value. The equation is given as follows:

For single variable:

$$h(X) = \theta_0 + \theta_1 X$$

For multiple variables:

$$h(X) = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 \ldots\ldots \theta_n Xn$$

In the above equations, is the intercept and to are the slopes of the respective attributes from to . Consider the 2 graphs given below based on the linear model:

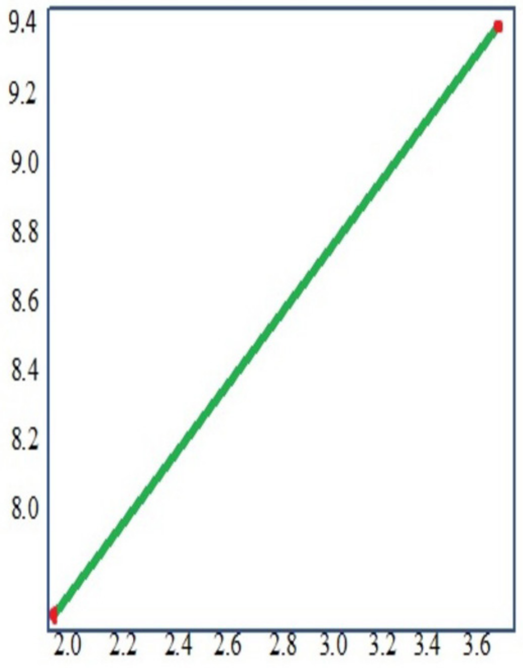


Figure 4.1: *Linear model having two points*

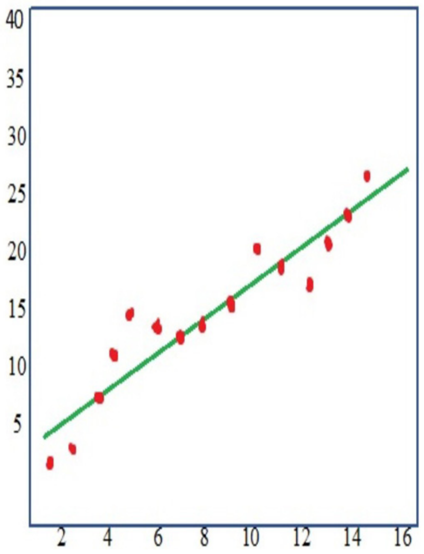


Figure 4.2: *Linear model having multiple points*

Figure 4.1 represents two points and *Figure 4.2* represents multiple points. Consider the following equation that can be used to represent many possibilities that can fit a straight line:

$$h(X)= \theta_0+\theta_1 X+ \theta_2 X^2+ \theta_3 X^3.......\theta_n X^n$$

Figure 4.3 shown below may represent the above equation:

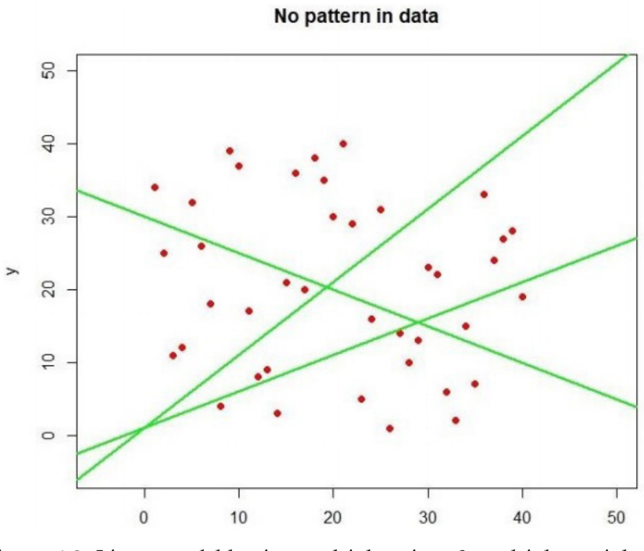


Figure 4.3: *Linear model having multiple points & multiple straight lines*

If there is one theta, then it represents one slope of direction and we obtain a straight line. If there are multiple thetas, then they represent multiple slopes of direction and we obtain a curved line. A linear model having multiple points, multiple curves, and multiple theta values is shown in *Figure 4.4*:

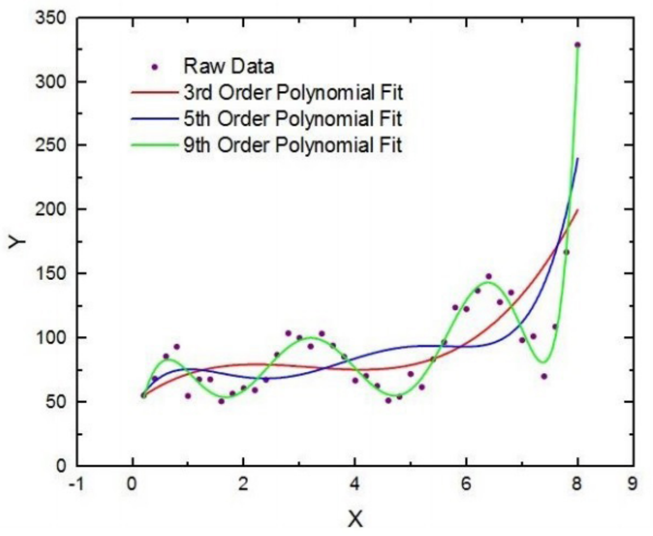


***Figure 4.4:** Linear model having multiple points & multiple curves*

We need regularization in order to prevent overfitting in a model. Our model may be in one of the following states:

- Overfitting
- Appropriate fitting or good fitting
- Under-fitting

Overfitting, appropriate fitting, and under-fitting are shown in *Figure 4.5*, *Figure 4.6*, and *Figure 4.7*. In a classification problem, when we work on a data set to perform the task of prediction, we calculate the accuracy by implementing on the training data and the testing data. If it obtains a satisfactory performance, we try to enhance the performance by either adding or removing certain features set. Sometimes, the designed model may behave poorly. It shows poor performance when either the designed model is too simple or too complex to address the target:

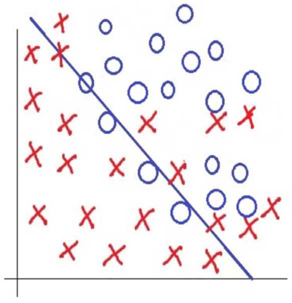


Figure 4.5: *Under-fitting*

In *Figure 4.5*, under-fitting of data takes place as all the points are not covered by the line. It is also referred to as high bias. We can avoid under-fitting by using a polynomial equation that generates a curved line instead of a straight line:

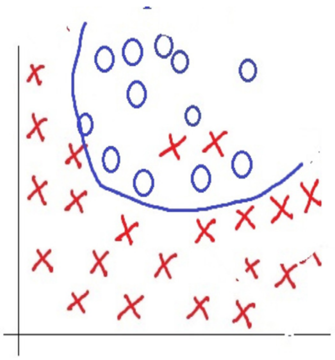


Figure 4.6: *Appropriate fitting or good fitting*

In *Figure 4.6*, almost all the points are covered by the line. So, it is referred to as good fitting or appropriate fitting. Also, it maintains a balance between variance and bias:

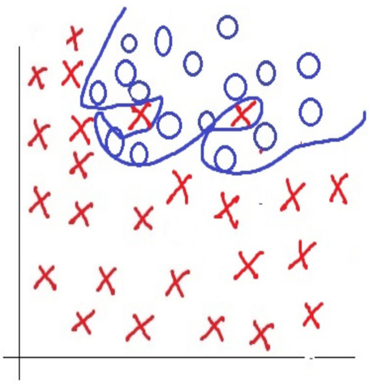


Figure 4.7: *Overfitting*

In *Figure 4.7*, all the points including outliers and noise are covered by the line. So, it is also referred to as overfitting. As this model is too complex, it gives a poor result. It is also known as **high variance**. In order to avoid overfitting, we have to use regularization. Regularization performs the task of reducing the variance by increasing bias, which thereby decreases the expected error.

The general equation for linear regression is given as follows:

$$h(X)= \theta_0+\theta_1 X_1+ \theta_2 X_2+ \theta_3 X_3 \ldots\ldots \theta_n X_n$$

Here, θ represents the coefficients of the independent variable *X*. Residual Sum of Squares is used to reduce the error existing in the coefficients. Residual Sum of Squares (*R*) may be defined as follows:

$$R=\Sigma_{i=1to\,n}(y_i-\theta_0- \Sigma_{j=1to\,p}\vartheta_j x_{ij})^2$$

Ridge Regression may be defined as a technique that can be used to analyze multiple regression data that exhibit multicollinearity. Ridge Regression may be defined as follows:

Ridge Regression$=\Sigma_{i=1to\,n}(y_i-\theta_0- \Sigma_{j=1to\,p}\theta_j x_{ij})^2+\lambda\Sigma_{i=1to\,p}\theta_j^2 = R+\lambda\Sigma_{i=1to\,p}\vartheta_j^2$

The value of λ is very crucial for our outcome. If the value of λ is zero then there is no affect in the outcome, but if it is equal to infinity then it affects the result and it is not desirable.

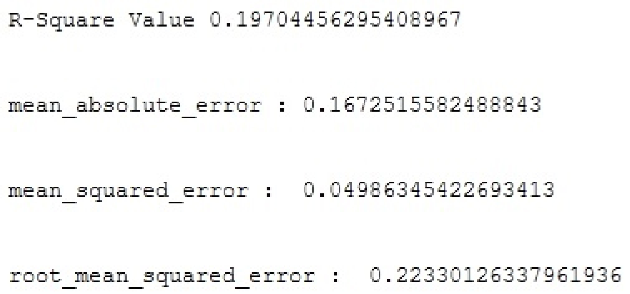
Lasso regression helps in shrinking the coefficients to zero and hence, remove them from the model. If there are many features which seem to be irrelevant and can be ignored, then Lasso regression is used. Lasso regression is computationally more intensive. Elastic-net regression is a combination of Lasso regression and ridge regression.

Ridge regression helps in shrinking the coefficients to almost zero but not completely zero.

Consider following code on Lasso regression:

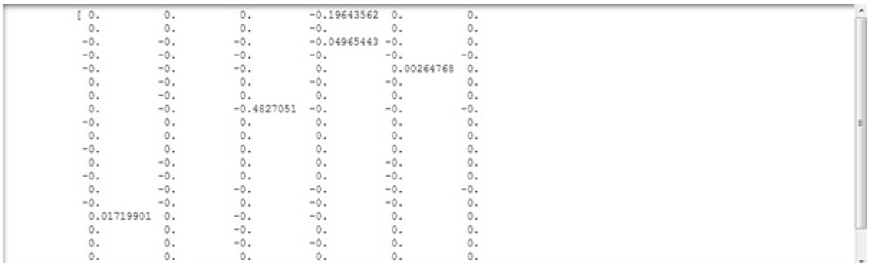
```
import numpy as np
import pandas as pd
import matplotlib.pyplot
%matplotlib inline
import seaborn as sns
from sklearn.linear_model import Lasso
url 1= 'http://archive.ics.uci.edu/ml/machine-learning-
databases/communities/communities.data'
```

```
data = pd.read_csv(url1, header=None, na_values=['?'])
X = data.drop(127, axis=1)
y = data[127]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, 
y,test_size = 0.3, random_state=1)
from sklearn.linear_model import LinearRegression
linearreg = LinearRegression()
linearreg.fit(X_train, y_train)
from sklearn import metrics
print("R-Square Value",r2_score(y_test,y_pred))
print ("\nmean_absolute_error :",metrics.mean_absolute_
error(y_test, y_pred))
print ("\nmean_squared_error : ",metrics.mean_squared_
error(y_test, y_pred))
print ("\nroot_mean_squared_error : ",np.sqrt(metrics.
mean_squared_error(y_test, y_pred)))
```



```
R-Square Value 0.19704456295408967


mean_absolute_error : 0.1672515582488843


mean_squared_error :  0.04986345422693413


root_mean_squared_error :  0.22330126337961936
```

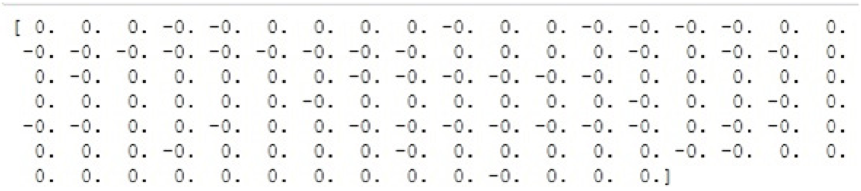
```
lassoregre = Lasso(alpha=0.005, normalize=True)
lassoregre.fit(X_train, y_train)
print(lassoregre.coef_)
```



```
[ 0.          0.          0.          -0.19643562  0.          0.
  0.          0.          0.          -0.          0.          0.
 -0.         -0.         -0.          -0.04965443 -0.          0.
 -0.         -0.         -0.          -0.         -0.         -0.
 -0.         -0.         -0.           0.          0.00264768  0.
  0.         -0.          0.          -0.         -0.          0.
  0.         -0.          0.           0.          0.          0.
  0.         -0.         -0.4827051   -0.         -0.         -0.
 -0.          0.          0.           0.          0.          0.
  0.          0.          0.           0.          0.          0.
 -0.          0.          0.           0.          0.          0.
  0.         -0.          0.           0.         -0.          0.
 -0.         -0.          0.           0.         -0.          0.
  0.         -0.         -0.          -0.         -0.         -0.
 -0.         -0.          0.          -0.         -0.          0.
  0.01719901  0.         -0.          -0.          0.          0.
  0.          0.         -0.           0.          0.          0.
  0.          0.         -0.          -0.          0.          0.
  0.          0.          0.           0.          0.          0.
```

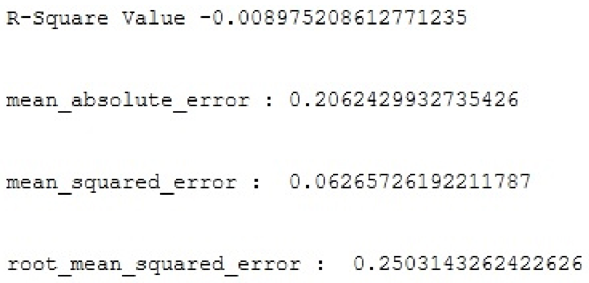
```
lassoregre = Lasso(alpha=0.05, normalize=True)
```

```
lassoregre.fit(X_train, y_train)
print(lassoregre.coef_)
```



```
[ 0.  0.  0. -0. -0.  0.  0.  0.  0. -0.  0.  0. -0. -0. -0. -0.  0.  0.
 -0. -0. -0. -0. -0. -0. -0. -0. -0.  0.  0.  0.  0. -0.  0. -0. -0.  0.
  0. -0.  0.  0.  0.  0.  0. -0. -0. -0. -0. -0. -0.  0.  0.  0.  0.  0.
  0.  0.  0.  0.  0.  0. -0.  0.  0.  0.  0.  0.  0. -0.  0.  0. -0.  0.
 -0. -0.  0.  0. -0.  0.  0. -0. -0. -0. -0. -0. -0. -0.  0. -0. -0.  0.
  0.  0.  0. -0.  0.  0.  0.  0. -0.  0.  0.  0.  0.  0. -0. -0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -0.  0.  0.  0.]
```

```
# compute RMSE (for alpha=0.01)
y_pred = lassoregre.predict(X_test)
# compute MAE, MSE, RMSE
# compute R square value, MAE, MSE, RMSE
from sklearn.metrics import r2_score
from sklearn import metrics
print("R-Square Value",r2_score(y_test,y_pred))
print ("mean_absolute_error :",metrics.mean_absolute_
error(y_test, y_pred))
print ("\nmean_squared_error : ",metrics.mean_squared_
error(y_test, y_pred))
print ("\nroot_mean_squared_error : ",np.sqrt(metrics.
mean_squared_error(y_test, y_pred)))
```



```
R-Square Value -0.008975208612771235


mean_absolute_error : 0.2062429932735426


mean_squared_error :  0.06265726192211787


root_mean_squared_error :  0.2503143262422626
```

```
from sklearn.linear_model import LassoCV
lassoregrecv = LassoCV(n_alphas=100, normalize=True,
random_state=1)
lassoregrecv.fit(X_train, y_train)
print('alpha : ',lassoregrecv.alpha_)
```

```
alpha : 0.001687868294707203
```

```
print(lassoregrecv.coef_)
```

```
[ 0.          0.          0.         -0.26320105  0.          0.
  0.          0.          0.          0.          0.          0.
 -0.         -0.         -0.         -0.1285917   0.          0.
 -0.         -0.         -0.         -0.         -0.         -0.
 -0.          0.         -0.          0.          0.11511909  0.
  0.         -0.          0.         -0.         -0.          0.
  0.         -0.          0.04667596  0.          0.          0.08875875
  0.         -0.09735466 -0.3276063  -0.         -0.         -0.
 -0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
 -0.          0.          0.          0.          0.          0.
  0.         -0.          0.          0.         -0.          0.
 -0.         -0.          0.          0.          0.          0.
  0.         -0.         -0.         -0.         -0.         -0.
 -0.         -0.          0.         -0.         -0.          0.
  0.14776487  0.         -0.         -0.         -0.         -0.
  0.          0.         -0.          0.          0.          0.
  0.00639313  0.         -0.         -0.          0.          0.
  0.          0.          0.          0.         -0.          0.
  0.          0.          0.02772175  0.         -0.          0.
 -0.          0.        ]
```

```
#  the best alpha value is predicted using the predict
method
y_pred = lassoregrecv.predict(X_test)
# Compute R square value, MAE, MSE, RMS
print("R-Square Value",r2_score(y_test,y_pred))
print("\n")
print ("\nmean_absolute_error :",metrics.mean_absolute_
error(y_test, y_pred))
print ("\nmean_squared_error : ",metrics.mean_squared_
error(y_test, y_pred))
print ("\nroot_mean_squared_error : ",np.sqrt(metrics.
mean_squared_error(y_test, y_pred)))
```

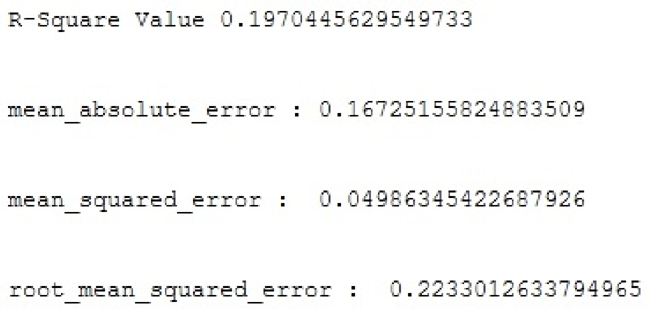
```
R-Square Value 0.5506049113488305


mean_absolute_error : 0.13416628182873494


mean_squared_error :  0.0279073911189985


root_mean_squared_error :  0.16705505415580368
```

In the above code, the value of alpha should be positive. The value of alpha can be increased for achieving high regularization. In the above code, `normalize` is set to true. It is used to scale the features.

The value of R-Square increases when we apply regularization. The value of mean absolute error, mean squared error, and root mean squared error decreases when we apply regularization.

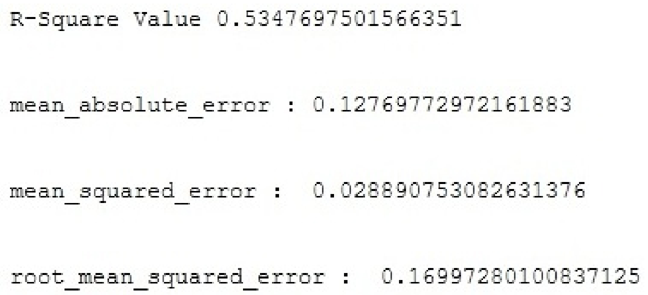
Consider the following code on Ridge regression:

```
from sklearn.linear_model import Ridge
ridgereg = Ridge(alpha=0, normalize=True)
ridgereg.fit(X_train, y_train)
y_pred = ridgereg.predict(X_test)
# calculate R square value, MAE, MSE, RMSE
from sklearn import metrics
print("R-Square Value",r2_score(y_test,y_pred))
print ("\nmean_absolute_error :",metrics.mean_absolute_
error(y_test, y_pred))
print ("\nmean_squared_error : ",metrics.mean_squared_
error(y_test, y_pred))
print ("\nroot_mean_squared_error : ",np.sqrt(metrics.
mean_squared_error(y_test, y_pred)))
```



```
R-Square Value 0.1970445629549733


mean_absolute_error : 0.16725155824883509


mean_squared_error :  0.04986345422687926


root_mean_squared_error :  0.2233012633794965
```

```
ridgeregre = Ridge(alpha=0.1, normalize=True)
ridgeregre.fit(X_train, y_train)
y_pred = ridgeregre.predict(X_test)
# calculate R square value, MAE, MSE, RMSE
from sklearn import metrics
print("R-Square Value",r2_score(y_test,y_pred))
print ("\nmean_absolute_error :",metrics.mean_absolute_
error(y_test, y_pred))
print ("\nmean_squared_error : ",metrics.mean_squared_
error(y_test, y_pred))
print ("\nroot_mean_squared_error : ",np.sqrt(metrics.
mean_squared_error(y_test, y_pred)))
```



```
R-Square Value 0.5347697501566351


mean_absolute_error : 0.12769772972161883


mean_squared_error :  0.028890753082631376


root_mean_squared_error :  0.16997280100837125
```

In the above code, we can see that the value of R square increases and the value of mean squared error, mean absolute error, and root mean squared error decreases when regularization is applied.

Overfitting and regularization in logistic regression

Logistic regression may be defined as a generalized linear model, but instead of continuous output, it produces a categorical output. One of the common problems occurring in machine learning is overfitting where a model is able to respond well on the training data rather than the testing data. Overfitting takes place when the model is complex due to the number of observations being more as compared to the number of parameters.

Regularization is one way of dealing with overfitting. Regularization can handle high correlation among various features, filtering out noise from data and prevent overfitting. The regularization term may be represented as follows:

$$\frac{\lambda}{2}||w||^2=\frac{\lambda}{2}\Sigma_{j=1 \text{ to } m} W_j^2$$

Here, λ is the regularization parameter. In order to apply regularization to logistic regression, the regularization term is added to the cost function to shrink weights.

The inverse of regularization is represented by the parameter *C* as:

C=1/λ

If we decrease the value of the parameter *C*, then the regularization strength λ increases and weight the coefficients or complexity decreases. Consider the following Python code on regularization in logistic regression:

```
%pylab inline
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=0)

from sklearn.preprocessing import StandardScaler
ssc = StandardScaler()
ssc.fit(X_train)
X_train_std = ssc.transform(X_train)
X_test_std = ssc.transform(X_test)

from sklearn.linear_model import LogisticRegression

weights, params = [], []
for c in np.arange(1, 5):
lrr = LogisticRegression(C=10**c, random_state=0)
lrr.fit(X_train_std, y_train)
weights.append(lrr.coef_[1])
params.append(10**c)

weights = np.array(weights)

# Decision region drawing
import matplotlib.pyplot as plt

plt.plot(params, weights[:, 0], color='blue', marker='x',
label='petal length')
plt.plot(params, weights[:, 1], color='green',  marker='o',
label='petal width')
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.legend(loc='right')
plt.xscale('log')
plt.show()
```

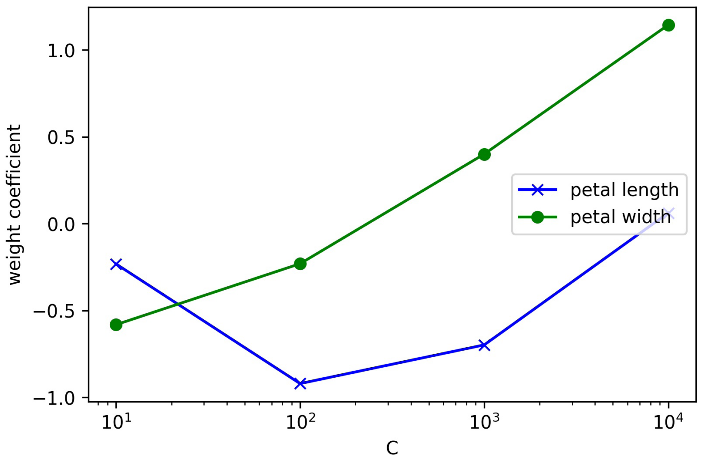


Figure 4.8

In the above code, we used 10 logistic regression models with different values of *C*. The plot given above shows that as the value of *C* (inverse regularization) decreases, the value of the weight coefficient decreases.

Conclusion

In this chapter, we discussed about overfitting and regularization in linear regression and overfitting and regularization in logistic regression. We discussed how the R square, mean absolute error, mean squared error, and root mean squared error changes by applying regularization. In the next chapter, we will discuss about the feasibility of learning an unknown target function, in-sample error, and out-of-sample error.

Questions

1. Explain overfitting and regularization in linear regression with an example.
2. Explain overfitting and regularization in logistic regression with an example.

Chapter 5
Feasibility of Learning

Introduction

Regularization is one way of dealing with overfitting. Regularization can handle high correlation among various features, filtering out noise from data, and prevent overfitting. In the previous chapter, we discussed about overfitting and regularization in linear regression and overfitting and regularization in logistic regression. In this chapter, we will discuss about the Feasibility of Learning an unknown target function, in-sample error, and out-of-sample error.

Structure

In this chapter, we will study the following topics:

- Feasibility of learning an unknown target function
- In-sample error
- Out-of-sample error

Objectives

- To know the steps involved in building a machine learning model
- Understanding feasibility of learning using Hoeffding's Inequality
- Understanding in-sample error
- Understanding generalization error

Feasibility of learning an unknown target function

For building a machine learning model, the following hierarchy is followed:

1. **Collection of data**: It involves gathering of data on the basis of the machine learning project that we desire to make. Data may be gathered from various sources such as files, sensors, databases, etc.
2. **Pre-processing of data**: The data collected from different sources for building the machine learning model cannot be directly used for analysis purpose, as it may contain a large amount of noisy data, unorganized text, missing values, large values, or irrelevant text. All such unwanted data may be eliminated in order to obtain clean data. While developing a machine learning model, we must follow the 80/20 rule. According to this rule, we must spend 80% of time in pre-processing of the data and 20% of time in analysis. Data may be classified into the following categories:

    a. **Numerical**: For e.g., age, salary, etc.

    b. **Categorical**: For e.g., nationality, gender, etc.

    c. **Ordinal**: For e.g., high, medium, low, etc.

    Data pre-processing may be performed in the following ways:

    a. **Dealing with null values**: We can solve the problem of null values by either deleting the rows and columns that comprise null values or by using imputation, which is a process of substituting the missing values with some substituted values.

  b. **Standardization**: It is a process that involves the manipulation of values so that the mean of all the values is 0 and the standard deviation is 1.

  c. **Dealing with categorical variables**: Categorical variables are those which are discrete and not continuous.

  d. **Feature scalin**g: It is technique in which we make the values of all the features same by scaling down the features that are insignificant and have a large range of values.

  e. **Splitting the data**: In machine learning, we usually split the data in 70:30, meaning 70% of the data is used for training and 30% of the data is used for testing.

3. **Finding the model that will be best for the data:** Find the machine learning model which is best suited for our problem.
4. **Training and testing of the developed model**
5. **Evaluation**

Steps involved in building a machine learning model are depicted in *Figure 5.1*:

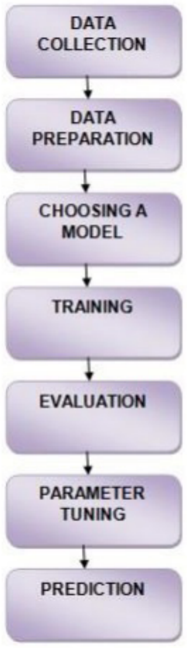


Figure 5.1: *Building a machine learning model*

In the above figure, data preparation refers to data pre-processing and prediction refers to the testing phase of the developed machine learning model. Feasibility of machine learning may also be expressed using Hoeffding's Inequality. According to Hoeffding's Inequality, for a given sample size M and ϵ as tolerance, the probability of the difference of in-sample estimation (μ) and the expected outcome (v) is smaller than a constant quantity.

It is defined as follows:

$$P(|v-\mu|>\epsilon)<=2exp^{-2\epsilon 2M}$$

In a given binary classification problem, the output is 1 or -1. Here, μ represents out-of-sample error $E_{out}(h)$, which is an expected error as a result of the preferred hypothesis h and v represents the in-sample error $E_{in}(h)$, which is an error arising as a result of the classification of data points by hypothesis.

It is represented as follows:

$$P(|E_{in}(h) - E_{out}(h)|> \epsilon) <= 2exp^{-2\epsilon 2M}$$

In-sample error and out-of-sample error

The data points that are used for making a model are referred to as in-sample data. In-sample error may be defined as the error that is obtained on the same data set that is used for building the machine learning model.

The data points that are new and do not belong to any of the training data sample are referred to as out-of-sample data. Out-of-sample error may be defined as the error that is obtained on the new data set. It is also referred to as the generalization error.

In the previous chapter, we had discussed about overfitting and underfitting. Consider the following table that distinguishes underfitting, overfitting, and just right fit condition in terms of training error and testing error:

	Underfitting	**Overfitting**	**Just right fit**
Symptoms	• Training Error is High. • Training Error is very close to the Test Error. • High Bias	• Training Error is Low. • Training Error is much lower than the Test Error. • High Variance	• Training Error is Low. • Training Error is slightly lower than the Test Error.

Table 5.1: *Comparison of underfitting, overfitting, and just right fit*

Consider the following code on in-sample error and out-of-sample error:

```
print(__doc__)

import numpy as np
from sklearn import linear_model
train_samples, test_samples, n_features = 80, 200, 700
np.random.seed(0)
coefficient = np.random.randn(n_features)
coefficient[50:] = 0.0
X = np.random.randn(train_samples + test_samples, n_
features)
y = np.dot(X, coefficient)

# Splitting train and test data
train_X, test_X = X[:train_samples], X[train_samples:]
train_y, test_y = y[:train_samples], y[train_samples:]
# Finding train and test errors
alphas = np.logspace(-5, 1, 60)
enet = linear_model.ElasticNet(l1_ratio=0.7, max_
iter=10000)
train_errors = list()
test_errors = list()
for alpha in alphas:
    enet.set_params(alpha=alpha)
    enet.fit(train_X, train_y)
    train_errors.append(enet.score(train_X, train_y))
    test_errors.append(enet.score(test_X, test_y))

i_alpha_optimum = np.argmax(test_errors)
alphaoptimum = alphas[i_alpha_optimum]
print("Optimal regularization parameter is: %s" %
alphaoptimum)

# Finding the coefficient on full data with optimal
regularization parameter

enet.set_params(alpha=alphaoptimum)
coefficient1 = enet.fit(X, y).coef_

# Plotting results functions

import matplotlib.pyplot as plt
plt.subplot(2, 1, 1)
plt.semilogx(alphas, train_errors, label='Train')
plt.semilogx(alphas, test_errors, label='Test')
plt.vlines(alphaoptimum, plt.ylim()[0], np.max(test_
errors), color='k',
linewidth=3, label='Optimum on test')
plt.legend(loc='lower left')
plt.ylim([0, 1.2])
plt.xlabel('Regularization parameter')
plt.ylabel('Performance')

# Show the estimated coef_ versus true coef
plt.subplot(2, 1, 2)
plt.plot(coefficient, label='True coef')
plt.plot(coefficient1, label='Estimated coef')
plt.legend()
plt.subplots_adjust(0.07, 0.05, 0.87, 0.87, 0.22, 0.29)
plt.show()
```

The above code gives the following output:

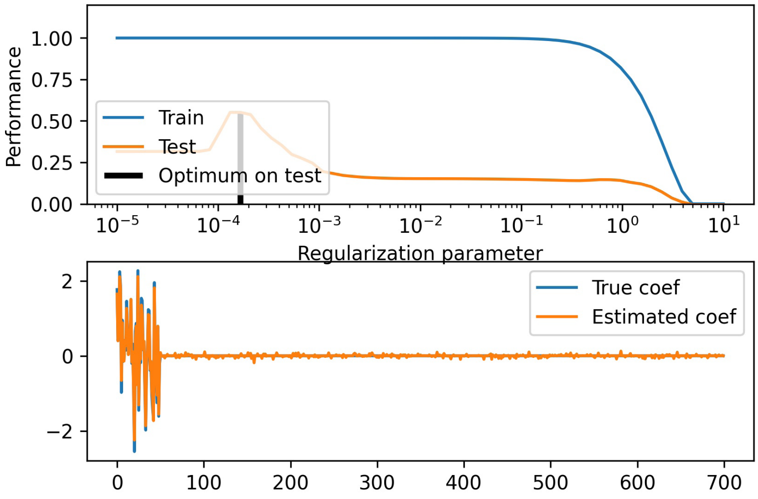


Figure 5.2: *Test error and train error*

The above code illustrates that the performance on the unseen or test data is different from the performance on the training data. It is seen that as we increase regularization, the performance on the training data degrades and the performance on the test data is optimal within the given range of values.

Conclusion

In this chapter, we discussed the feasibility of learning an unknown target function and in-sample error and out-of-sample error with example of its value is affected by increasing regularization.

In the next chapter, we will discuss about Support Vector Machine, margin, large margin methods, and kernel methods in terms of Support Vector Machine.

Questions

1. Explain the feasibility of learning an unknown target function with an example.
2. Explain out-of-sample error with an example.
3. Explain in-sample error with an example.

CHAPTER 6
Support Vector Machine

Introduction

Support Vector Machine (**SVM**) may be defined as a machine learning algorithm that can be used for regression and classification. It is generally used for classification purpose. In this chapter, we will discuss about Margin and Large Margin Methods and Kernel Methods.

Structure

- Margin and Large Margin methods
- Kernel methods

Objectives

- To know the significance of Margin and Large Margin methods
- Understanding Kernel methods

Margin and Large Margin methods

In SVM, the data item is plotted in an n-dimensional space, where n represents the number of features. A classification is performed by finding the hyperplane that can differentiate the two classes. Consider *Figure 6.1*. Here, the classification of two different shapes is performed by finding the hyperplane:

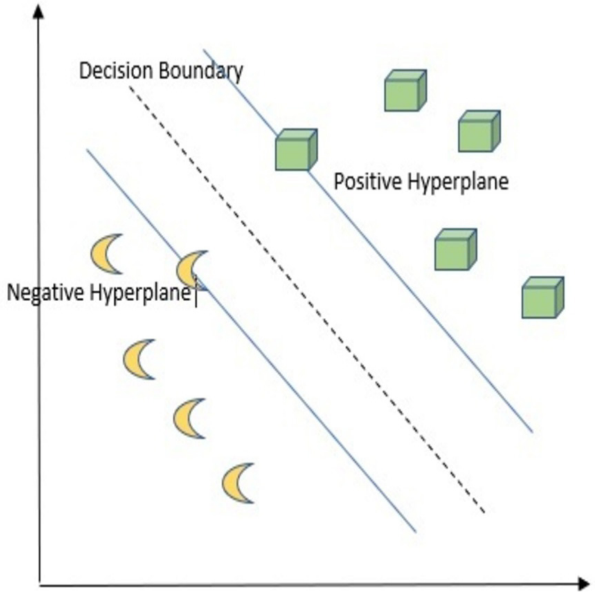


Figure 6.1: *The classification of shapes using hyperplane in SVM*

Selecting the right hyperplane for a given problem can be done in the following ways:

1. Choose the hyperplane that classifies the data points in a better way. Consider *Figure 6.2*. Here, we have three hyperplanes, namely **A**, **B** and **C**. We need to choose one hyperplane out of these. We choose hyperplane **A** as it classifies the data points efficiently as compared to the other hyperplanes:

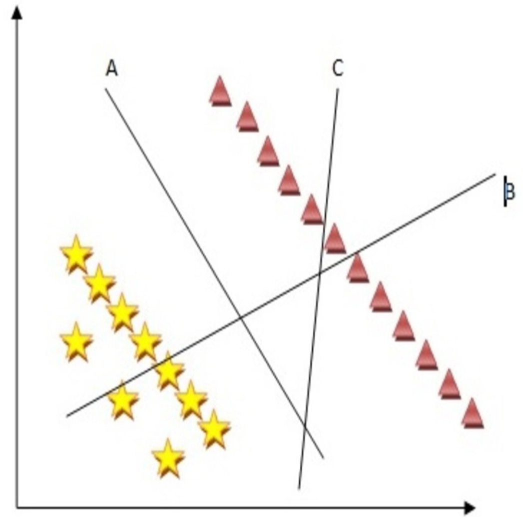


Figure 6.2: *Hyperplane A classifies the data points efficiently*

2. Use Margin and Large Margin Methods to find the appropriate hyperplane. The distance between the nearest data point and the hyperplane is referred to as margin. We must select the hyperplane that has a larger margin; this prevents a chance of miss classification. Consider *Figure 6.3*. Here, the margin of hyperplane **B** is the largest as compared to the margin of the hyperplanes **A** and **C**. So, we choose hyperplane **B** for classifying our data points:

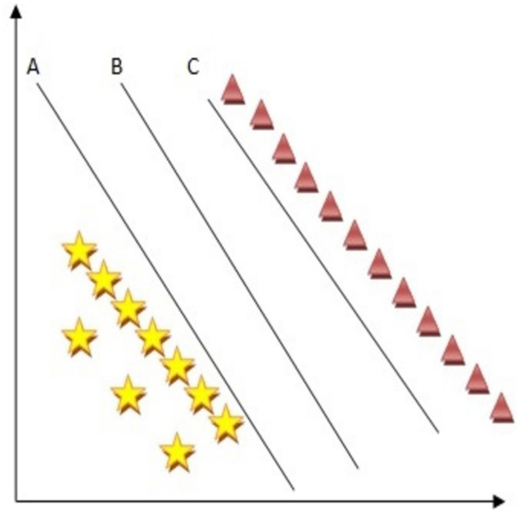


Figure 6.3: *Margin and Large Margin Methods*

3. Identify the correct hyperplane that can classify all the data points correctly without any error. Consider *Figure 6.4*. Here, if we choose the hyperplane **B** instead of hyperplane **A**, since hyperplane **B** has a larger margin as compared to hyperplane **A**, then hyperplane **B** doesn't classify all the data points correctly resulting in an error. Hyperplane **A** has a smaller margin but it classifies all the data points correctly. So, SVM will choose hyper plane **A** over hyper plane **B**.

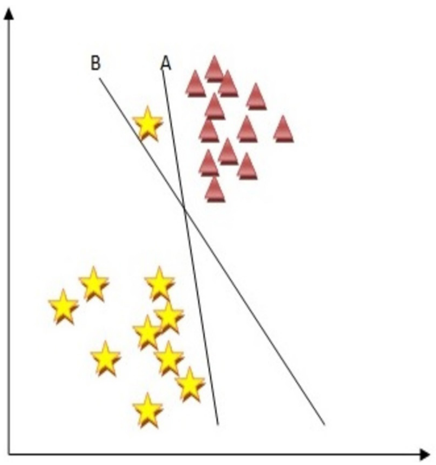


Figure 6.4: *Choosing hyperplane A over hyperplane B for classification*

4. SVM has a characteristic of ignoring the outliers or the noise. Consider *Figure 6.5*. Here, the hyperplane chosen has a large margin and at the same time, the SVM ignores the star that lies in the other boundary and this star is treated as a noise or an outlier.

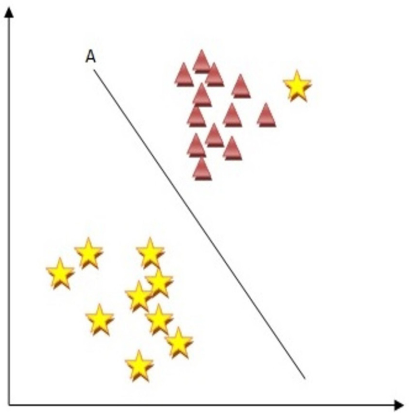


Figure 6.5: *Classification by SVM is robust to outliers*

Consider the following code that depicts the data points that are classified correctly using SVM:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns; sns.set()
from sklearn.datasets.samples_generator import make_blobs
X, y = make_blobs(n_samples=100, centers=2, random_state=0,
cluster_std=0.60)
plt.scatter(X[:, 0], X[:, 1], c=y, s=100, cmap='autumn');
```

It gives the following output as shown in *figure 6.6*:

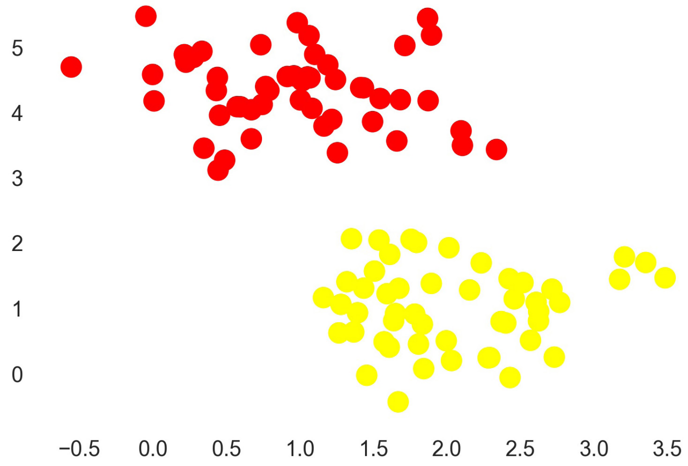


***Figure 6.6:** Classification of data points using SVM*

Consider the following code that displays straight lines and separates or classifies the different sets of data. This is shown in *figure 6.7*:

```
fit = np.linspace(-1, 3.5)
plt.scatter(P[:, 0], P[:, 1], c=q, s=100, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='green',
markeredgewidth=2, markersize=10)
for m, b in [(1, 0.65), (0.5, 1.6), (-0.2, 2.9)]:
    plt.plot(fit, m * fit + b, '-k')
plt.xlim(-1, 3.5);
```

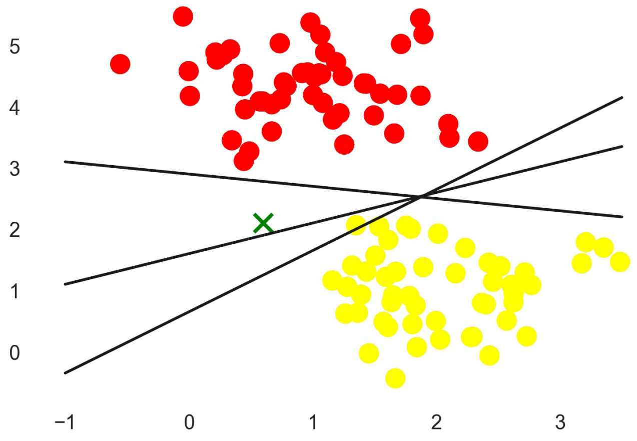


Figure 6.7: *Straight lines separating different sets of data*

Instead of drawing zero width straight lines, margins may be drawn around straight lines up to the nearest data point. This is shown in *figure 6.8*:

```
fit = np.linspace(-1, 3.5)
plt.scatter(P[:, 0], P[:, 1], c=q, s=100, cmap='autumn')
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2,
2.9, 0.2)]:
    fit2 = m * fit + b
    plt.plot(fit, fit2, '-k')
    plt.fill_between(fit, fit2 - d, fit2 + d, edgecolor='none',
                     color='#AAAAAA', alpha=0.4)
plt.xlim(-1, 3.5);
```

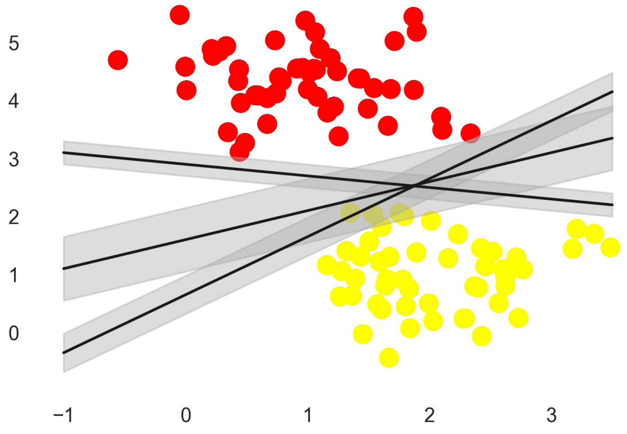


Figure 6.8: *Margins drawn around straight lines for classification in SVM*

Consider the following code using Scikit Learn that can be used to train the SVM model. This is depicted in *figure 6.9*:

```
from sklearn.svm import SVC # "Support vector classifier"
model = SVC(kernel='linear', C=1E10)
model.fit(P,q)
SVC(C=10000000000.0, break_ties=False, cache_size=200,
class_weight=None, coef0=0.0, decision_function_
shape='ovr', degree=3, gamma='scale',kernel='linear',
max_iter=-1, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False)
def plot_svc_decision_function(model, px=None, plot_
support=True):
    """Decision function is plotted for a 2D SVC"""
    if px is None:
        px = plt.gca()
    limx = px.get_xlim()
    limy = px.get_ylim()
    # To evaluate model, grid is created
    p = np.linspace(limx[0], limx[1], 30)
    q = np.linspace(limy[0], limy[1], 30)
    Q, P = np.meshgrid(q, p)
    pq = np.vstack([P.ravel(), Q.ravel()]).T
    P = model.decision_function(pq).reshape(P.shape)

    # Margins and decision boundary are plotted
    px.contour(P, Q, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])
    # plot support vectors
    if plot_support:
        px.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=400, linewidth=1, facecolors='none');
    px.set_xlim(limx)
    px.set_ylim(limy)
plt.scatter(P[:, 0], P[:, 1], c=q, s=100, cmap='autumn')
```

```
plot_svc_decision_function(model);
model.support_vectors_
array([[0.44359863, 3.11530945],
       [2.33812285, 3.43116792],
       [2.06156753, 1.96918596]])
```

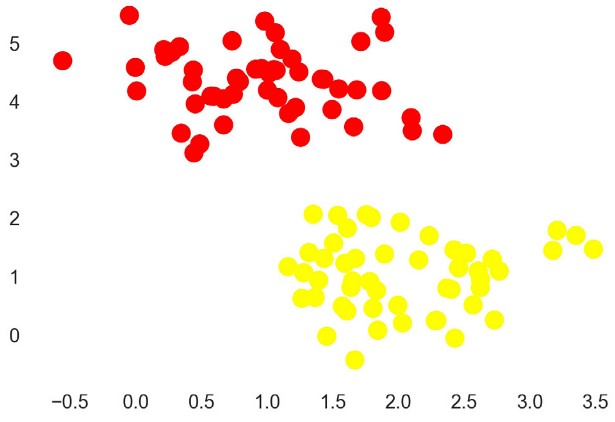


Figure 6.9: *Classification of the outcome using SVM*

Consider the following code that considers the first 50 and 130 points of the data set. This is shown in *figure 6.10*:

```
def plot_svm(N=10, ax=None):
    P, q = make_blobs(n_samples=200, centers=2,
                      random_state=0, cluster_std=0.60)
    P = P[:N]
    q = q[:N]
    model = SVC(kernel='linear', C=1E10)
    model.fit(X, q)

    px = px or plt.gca()
    px.scatter(X[:, 0], X[:, 1], c=q, s=100, cmap='autumn')
    px.set_xlim(-1, 4)
    px.set_ylim(-1, 6)
    plot_svc_decision_function(model, px)

fig, px = plt.subplots(1, 2, figsize=(16, 6))
```

```
fig.subplots_adjust(left=0.0625, right=0.95, wspace=0.1)
for pxi, N in zip(px, [50, 130]):
    plot_svm(N, pxi)
    pxi.set_title('N = {0}'.format(N))
```

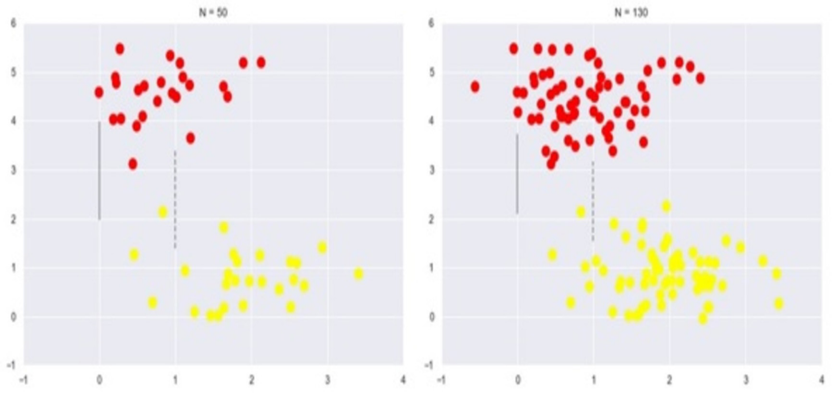


Figure 6.10: *Classification of the first 50 and 130 data points using SVM*

Consider the following code that shows the classification of data set which is not linearly separable. This is shown in *figure 6.11*:

```
from sklearn.datasets.samples_generator import make_circles
P, q = make_circles(200, factor=.1, noise=.1)
clf1 = SVC(kernel='linear').fit(P, q)
plt.scatter(P[:, 0], P[:, 1], c=q, s=50, cmap='autumn')
plot_svc_decision_function(clf1, plot_support=False);
```

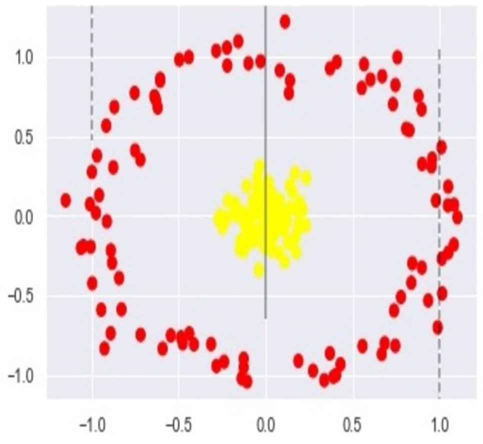


Figure 6.11: *Classification of data which is not linearly separable*

Kernel methods

Support Vector Machine kernel is a function that accepts low dimensional input space and converts it into a higher dimensional space. SVM performs the conversion of a not separable problem into a separable problem. It is generally used in non-linear separation problems. Consider the following *figure 6.12*. Here, we have a circular hyperplane that separates the stars and the triangles. We can convert the circular hyperplane into a linear hyperplane by introducing a new feature *z*, *z=x^2+y^2*. This is shown in *figure 6.13*:

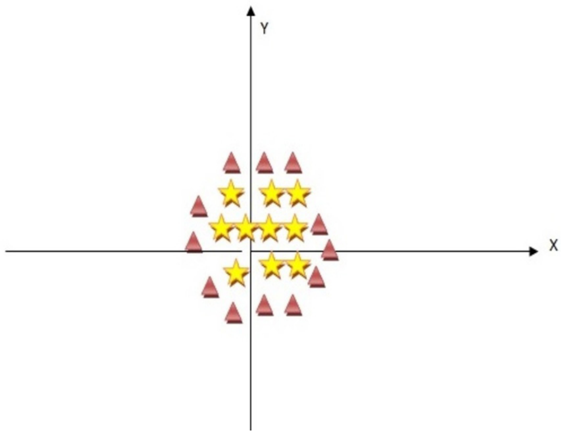


Figure 6.12: *Circular hyperplane separating the stars and the triangles*

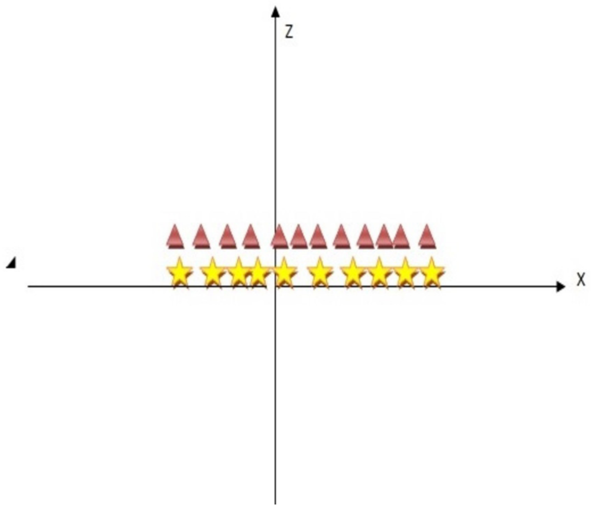


Figure 6.13: *Linear hyperplane separating the stars and the triangles*

Conclusion

In this chapter, we discussed the feasibility of learning an unknown target function and in-sample error and out-of-sample error with example of its value is affected by increasing Regularization.

Questions

1. Explain the significance of a large margin in SVM.
2. Explain SVM kernel.

CHAPTER 7 Neural Network

Introduction

Support Vector Machine is used for the purpose of classification and regression. In the previous chapter, we discussed about Support Vector Machine, margin and large margin methods, and kernel methods. Neural network refers to the parallel computing device that attempts to mimic the model of the brain. In the following chapter, we will discuss the early models of Neural Network as well as Perceptron learning model, back propagation, and Stochastic Gradient Descent. We discussed the implementation of perceptron learning, back propagation, and stochastic gradient descent in Python.

Structure

Neural network refers to a collection of algorithms that recognizes the relationships in data sets and mimics the working of a human brain. In this chapter, we will cover the following topics:

- Early Models
- Perceptron Learning

- Back propagation
- Stochastic Gradient Descent

Objectives

Neural network is a means of performing a machine learning task, in which a computer learns by the analysis of training examples. Following are the objectives covered in this chapter:

- To know early models and perceptron learning
- Understanding back propagation
- Understanding stochastic gradient descent

Early models

Neural network is one of the subfields of machine learning. Neural network accepts the input data, performs training on the data, and produces the output based on the training performed. In 1943, *Warren Mc Culloch* and *Walter Pits* described the working of neurons. They modeled neural networks with the help of electrical circuits in order to explain the working of neurons in the brain. In 1949, *Donald Hebb* wrote *The Organization of Behavior*, which pointed that the connection between the two neurons is enhanced if they are fired together. In 1959, *Bernard Widrow* and *Marcian Hoff* at Stanford developed neural network based models called **'ADALINE'** and **'MADALINE'**. **ADALINE** stands for **Adaptive Linear Elements** and **MADALINE** stands for **Multiple Adaptive Linear Elements**. ADALINE was used for the recognition of binary patterns. For a given stream of bits, it can predict the occurrence of the next bit. MADALINE stands for Multiple Adaptive Linear Elements. It is the first neural network that is applied to real-world problems and is still in use for commercial purposes.

Perceptron learning

Perceptron is based on a neuron, which is the basic processing unit of a brain. A neuron comprises of dendrites, cell body, and axon. Signal flows from the axon to the dendrites. An action signal is fired by a neuron when a particular threshold is met by a cell. This action either takes place or it does not. There is no concept of partial firing by a neuron. Consider *Figure 7.1* depicting the Perceptron model:

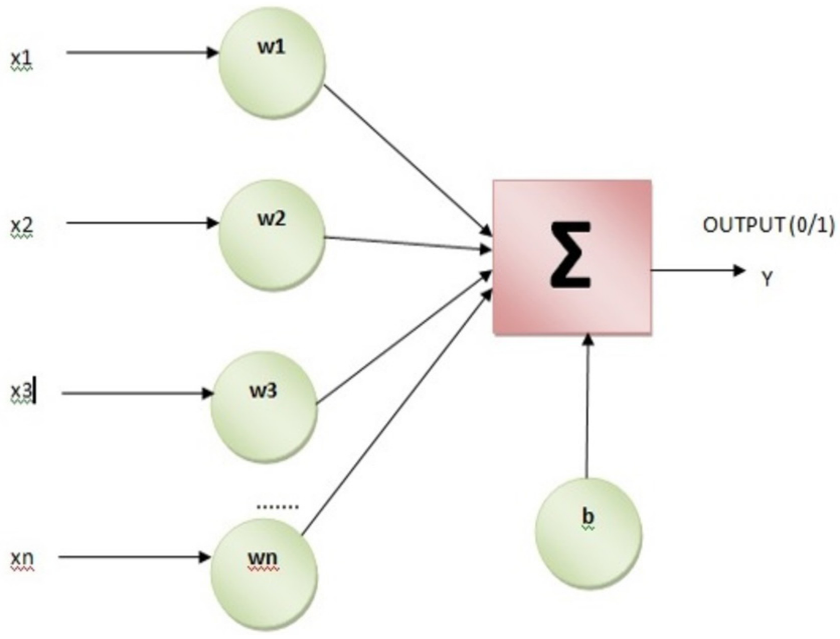


Figure 7.1: *Perceptron model*

Perceptron can be used for solving binary classification problems where the sample that needs to be identified belongs to two classes. Many features or inputs are sent to the linear unit of a Perceptron and it generates one binary output. Consider the following code on Perceptron learning. In this code, we have added 1 to the `input_size` in order to include bias in the weight vector:

```
import numpy as np


class Perceptronlearning(object):
    """Implementation of  a perceptron learning network"""
    def __init__(self, input_size):
        self.W = np.zeros(input_size+1)
def activation_fn(self, p):
""" 1 is returned if p>=0 otherwise it returns 0”””
    return 1 if p >= 0 else 0
""" Prediction is a process of sending an input to the
perceptron and returning an output. Bias is added to the
```

```
input vector. We can compute the inner product, and the
activation function is applied """
def predict(self, p):
    p = np.insert(p, 0, 1)
    q = self.W.T.dot(p)
    r = self.activation_fn(q)
    return r
def __init__(self, input_size, lr=1, epochs=10):
    self.W = np.zeros(input_size+1)
    # add one for bias
    self.epochs = epochs
    self.lr = lr
def fit(self, P, d):
    for _ in range(self.epochs):
        for i in range(d.shape[0]):
            y = self.predict(P[i])
            e = d[i] - y
            self.W = self.W + self.lr * e * np.insert(P[i],
0, 1)
class Perceptronlearning(object):
    """Implements a perceptron network"""
    def __init__(self, input_size, lr=1, epochs=100):
        self.W = np.zeros(input_size+1)
        # add one for bias
        self.epochs = epochs
        self.lr = lr

    def activation_fn(self, p):
        #return (p >= 0).astype(np.float32)
        return 1 if p >= 0 else 0

    def predict(self, p):
        q = self.W.T.dot(p)
        r = self.activation_fn(q)
        return r

    def fit(self, P, d):
        for _ in range(self.epochs):
            for i in range(d.shape[0]):
                p = np.insert(P[i], 0, 1)
                y = self.predict(p)
                e = d[i] - y
                self.W = self.W + self.lr * e * p
if __name__ == '__main__':
    P = np.array([
        [0, 0],
        [0, 1],
        [1, 0],
        [1, 1]
    ])
    d = np.array([1, 1, 0, 1])

    perceptron = Perceptronlearning(input_size=2)
    perceptron.fit(P, d)
    print(perceptron.W)
[ 0. -1.  2.]
```

So, a single neuron neural network is referred to as a Perceptron. It accepts the input and the weight, performs the weighted sum of the inputs, and applies an activation function over it. It accepts and generates only binary values. One of the limitations of the Perceptron learning model is that it can solve only linearly separable problems.

Back propagation

Back propagation is also referred to as Gradient Computation. Back propagation learning algorithm comprises two phases, namely: Gradient Computation Phase and Weight Updation Phase. The first phase is the Propagation phase. It involves the following steps:

1. **Forward Propagation** - Here, the training input pattern is sent to the neural network and the propagation's output activation is generated.
2. **Backward Propagation** – Here, the input is the propagation's output activation that is sent to the neural network, and it generates the deltas of all the output and hidden neurons.

Second phase is the Weight Updation phase. It involves the following steps:

1. Gradient of the weight is calculated by multiplying the output delta and the input activation.
2. A ratio or percentage of the gradient is subtracted from the weight. This ratio or percentage affects the quality of learning and speed. It is referred to as the learning rate. A neuron is able to train faster if the learning rate is higher. If the learning rate is lower, then the training is considered accurate.

Consider the following code on back propagation.

```
import numpy as np
def sigmoidfun(x):
    return 1.0/(1.0 + np.exp(-x))
def sigmoid_primefun(x):
    return sigmoidfun(x)*(1.0-sigmoidfun(x))
def tanh(x):
    return np.tanh(x)
def tanh_prime(x):
    return 1.0 - x**2

class NeuralNetwork:
    def __init__(self, layers, activation='tanh'):
        if activation == 'sigmoid':
            self.activation = sigmoidfun
            self.activation_prime = sigmoid_primefun
        elif activation == 'tanh':
            self.activation = tanh
            self.activation_prime = tanh_prime
        # Setting weights
        self.weights = []
        # let layers is [2,2,1]
        # weight values range= (-1,1)
        # hidden and input layers - random((2+1, 2+1)) : 3 x 3
        for i in range(1, len(layers) - 1):
            r = 2*np.random.random((layers[i-1] + 1,
layers[i] + 1)) -1
```

```
            self.weights.append(r)
        # output layer - random
        r = 2*np.random.random( (layers[i] + 1,
layers[i+1])) - 1
        self.weights.append(r)

    def fit(self, P, q, learning_rate=0.2, epochs=100000):
        # Adding the column of ones to P
        # This is to add the bias unit to the input layer
        ones = np.atleast_2d(np.ones(P.shape[0]))
        P = np.concatenate((ones.T, P), axis=1)
        for k in range(epochs):
            i = np.random.randint(X.shape[0])
            a = [P[i]]
            for l in range(len(self.weights)):
                  dot_value = np.dot(a[l], self.weights[l])
                    activation = self.activation(dot_value)
                    a.append(activation)
            # output layer
            error = q[i] - a[-1]
            deltas = [error * self.activation_prime(a[-1])]
            # we need to begin at the second to last layer
            # (a layer before the output layer)
            for l in range(len(a) - 2, 0, -1):
                deltas.append(deltas[-1].dot(self.
weights[l].T)*self.activation_prime(a[l]))

            # reverse
            # [level3(output)->level2(hidden)]  =>
[level2(hidden)->level3(output)]
            deltas.reverse()

            # backpropagation
            # 1. Multiply its output delta and input activation
            #    to get the gradient of the weight.
            # 2. Subtract a ratio (percentage) of the
gradient from the weight.
```

```
            for i in range(len(self.weights)):
                layer = np.atleast_2d(a[i])
                delta = np.atleast_2d(deltas[i])
                self.weights[i] += learning_rate *
layer.T.dot(delta)
            if k % 10000 == 0: print('epochs:', k)
    def predict(self, x):
        c = np.concatenate((np.ones(1).T, np.array(x)))
        for l in range(0, len(self.weights)):
            c = self.activation(np.dot(c, self.weights[l]))
        return c

if __name__ == '__main__':
    neu = NeuralNetwork([2,2,1])
    P = np.array([[0, 0],
                  [0, 1],
                  [1, 0],
                  [1, 1]])
    q= np.array([0, 1, 1, 0])
    neu.fit(P,q)
    for x in P:
        print(x,neu.predict(x))
epochs: 0
epochs: 10000
epochs: 20000
epochs: 30000
epochs: 40000
epochs: 50000
epochs: 60000
epochs: 70000
epochs: 80000
epochs: 90000
[0 0] [4.28817961e-05]
[0 1] [0.99667242]
[1 0] [0.99666444]
[1 1] [4.62864326e-05]
```

Stochastic Gradient Descent

Stochastic gradient descent is referred to as an iterative method that can be used for the optimization of an objective function with the necessary properties (subdifferentiable or differentiable). Stochastic gradient descent may be used to minimize the computations to be performed. It randomly selects a data point at a given iteration that reduces the computations enormously. Consider the following code using stochastic gradient descent. Here, we take 2 arrays, `P` and `Q`. Array `P` comprises the training samples. Array `Q` holds the target values.

```
import numpy as np
from sklearn import linear_model
P = np.array([[-1, 2], [1, -1], [2, 1], [2, 2]])
Q = np.array([1, 2, 1, 1])
SGDClassif = linear_model.SGDClassifier(max_iter = 1000,
tol=1e-3,penalty = "elasticnet")
SGDClassif.fit(P, Q)
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0,
fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal',
loss='hinge',
            max_iter=1000, n_iter_no_change=5, n_jobs=None,
              penalty='elasticnet', power_t=0.5, random_
state=None,
              shuffle=True, tol=0.001, validation_
fraction=0.1, verbose=0,
              warm_start=False)
SGDClassif.predict([[3.,3.]])
array([1])
SGDClassif.coef_
array([[  9.77200712, -19.54811198]])
SGDClassif.intercept_
array([-10.])
SGDClassif.decision_function([[3., 3.]])
array([-39.32831456])
import numpy as np
from sklearn import linear_model
```

```
nsamples, nfeatures = 9, 4
rng = np.random.RandomState(0)
q = rng.randn(nsamples)
p = rng.randn(nsamples, nfeatures)
SGDReg =linear_model.SGDRegressor(
   max_iter = 1000,penalty = "elasticnet",loss =
'huber',tol = 1e-3, average = True
)
SGDReg.fit(p, q)
SGDRegressor(alpha=0.0001, average=True, early_
stopping=False, epsilon=0.1,
             eta0=0.01, fit_intercept=True, l1_ratio=0.15,
             learning_rate='invscaling', loss='huber',
max_iter=1000,
             n_iter_no_change=5, penalty='elasticnet',
power_t=0.25,
             random_state=None, shuffle=True, tol=0.001,
             validation_fraction=0.1, verbose=0, warm_
start=False)
SGDReg.coef_
array([-0.00602635,  0.00566224, -0.00235155,  0.01238438])
SGDReg.intercept_
array([0.0043785])
SGDReg.t_
55.0
```

Conclusion

In this chapter, we learned about the early models of neural network, Perceptron model, back propagation, and stochastic gradient descent. In the next chapter, we will discuss about decision trees and regression trees, and their implementation in Python.

Questions

1. Explain the steps involved in back propagation.
2. Explain stochastic gradient descent.
3. Explain the early neuron models.
4. Write the disadvantage of the Perceptron model.

Chapter 8 Decision Trees

Introduction

Neural networks are a way to mimic the working of a human brain. Decision trees refer to the decision support structure that uses a tree to make decisions and draw all possible consequences. Decision trees are a way to display conditional control statements. In the following chapter, we will discuss about decision trees, regression trees, stopping criterion and pruning loss functions in a decision tree, categorical attributes, multiway splits and missing values in decision trees, and instability in decision trees.

Structure

Decision trees may be used for representing decisions and decision-making. This chapter will comprise the following topics:

- Decision trees
- Regression trees
- Stopping criterion and pruning loss functions in a decision tree

- Categorical attributes, multiway splits, and missing values in decision trees
- Instability in decision trees

Objectives

Decision trees refer to a popular and powerful tool for the purpose of prediction and classification. Following are the objectives of this chapter:

- To know about decision trees and regression trees
- Understanding the stopping criterion and pruning loss functions in decision trees
- Understanding categorical attributes, multiway splits, and missing values in decision trees
- Understanding the instability in decision trees

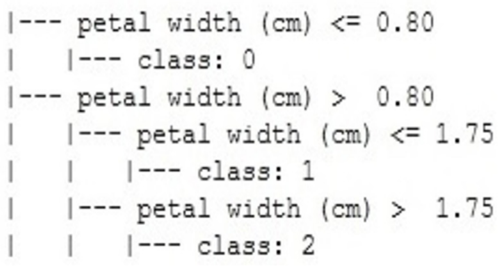
Decision trees

Decision trees refer to the non-parametric method of supervised learning. Decision trees are used for the purpose of regression and classification.

Consider the following code on decision trees using scikit learn. Here, `DecisionTreeClassifier` is a class that can perform multi-class classification on a given data set. `DecisionTreeClassifier` takes two arrays as input, namely `P` and `Q`.

```
from sklearn import tree
P = [[0, 0], [1, 1]]
Q = [0, 1]
DTclf = tree.DecisionTreeClassifier()
DTclf = DTclf.fit(X, Y)
DTclf.predict([[2., 2.]])
array([1])

DTclf.predict_proba([[2., 2.]])
array([[0., 1.]])
```

```
from sklearn.datasets import load_iris
from sklearn import tree
P, q = load_iris(return_X_y=True)
DTclf = tree.DecisionTreeClassifier()
DTclf = DTclf.fit(P, q)
tree.plot_tree(DTclf)
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_text
iris = load_iris()
decisiontree = DecisionTreeClassifier(random_state=0, max_
depth=2)
decisiontree = decisiontree.fit(iris.data, iris.target)
er = export_text(decisiontree, feature_names=iris['feature_
names'])
print(er)
```



```
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- class: 1
|   |--- petal width (cm) >  1.75
|   |   |--- class: 2
```

```
from sklearn import tree
X = [[0, 0], [2, 2]]
y = [0.5, 2.5]
DTclf = tree.DecisionTreeRegressor()
DTclf = DTclf.fit(X, y)
DTclf.predict([[1, 1]])
array([0.5])
```

In the above code, `DTclf.predict()` is used for the prediction of the samples of a class. `DTclf.predict_proba()` is used to find the probability of each class. `DecisionTreeClassifier` can be used for binary classification and multi-class classification. `plot_tree` function can be used to plot the tree.

Regression trees

Decision tree algorithms may be used for the process of the prediction of results based on the given data. Decision tree algorithms are of two types, namely: classification tree and regression tree algorithm. **Classification and Regression Tree** (**CART**) methodology came into existence in 1984. It was introduced by *Leo Breiman*, *Jerome Friedman*, *Richard Olshen*, and *Charles Stone*. In a classification tree algorithm, the outcome variable is categorical or fixed. For example, using the classification tree algorithm, we may decide what type of car a customer will purchase. In regression tree algorithm, the target outcome value is a real number. For example, the selling price of residential places may be predicted using the regression tree algorithm. In the classification tree algorithm, the data set is split into classes such as Yes or No. In regression tree algorithm, the target variable is continuous, for example, temperature, price, etc. The classification of decision tree algorithms is shown in *figure 8.1*:

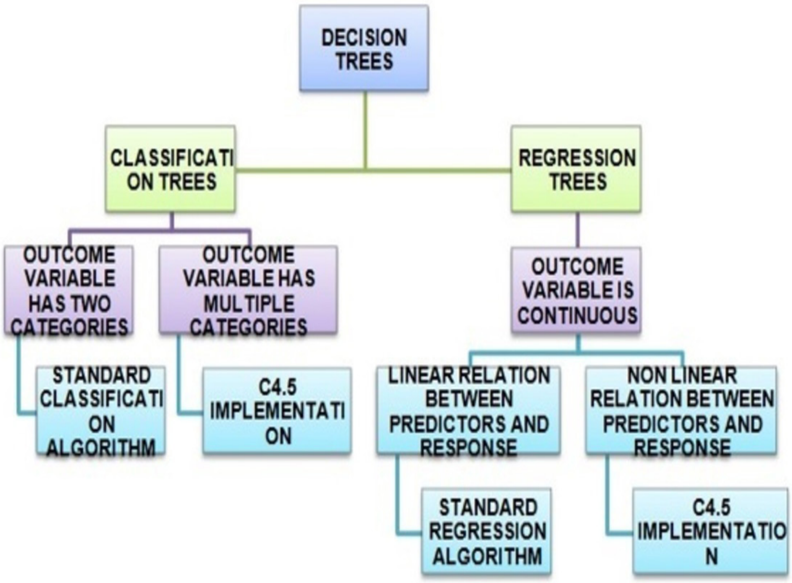

DECISION TREES
CLASSIFICATI ON TREES
REGRESSION TREES
OUTCOME VARIABLE HAS TWO CATEGORIES
OUTCOME VARIABLE HAS MULTIPLE CATEGORIES
OUTCOME VARIABLE IS CONTINUOUS
STANDARD CLASSIFICATI ON ALGORITHM
C4.5 IMPLEMENTATI ON
LINEAR RELATION BETWEEN PREDICTORS AND RESPONSE
NON LINEAR RELATION BETWEEN PREDICTORS AND RESPONSE
STANDARD REGRESSION ALGORITHM
C4.5 IMPLEMENTATIO N

Figure 8.1: *Classification of decision trees algorithms*

Stopping criterion and pruning loss functions in decision trees

The pruning technique is associated with the decision trees that can perform a reduction in the size of the decision trees by eliminating the parts of the tree that do not classify instances. Overfitting can be prevented by including pruning along with the decision trees. Overfitting occurs when the training is done so thoroughly that it also learns noise along with the pattern. Under-fitting occurs when the amount of training is so insufficient that all the patterns cannot be identified. Pruning means that the tree is cut back. *Quinlan* in 1987 suggested a simple method for pruning decision trees, referred to as the reduced error pruning. In reduced error pruning, internal nodes are traversed from bottom to top and pruned only if it doesn't reduce the tree's accuracy. *Olaru* and *Wehenkel* in 2003 suggested the use of minimum error pruning. In minimum error pruning, for every node we perform a comparison of 1-probability error rate estimation without and with pruning. Pessimistic pruning is a fast method of pruning in which the nodes are traversed in a top to down manner. If a given internal node is pruned then all the descendents of this internal node are not sent for the pruning process. Optimal pruning is used to guarantee optimality and is based on the concept of dynamic programming. In optimal pruning, the tree obtained after pruning is much smaller as compared to the original tree and the number of internal nodes are much smaller as compared to the number of leaves.

Categorical attributes, multiway splits, and missing values in decision trees

CART refers to the decision tree algorithm that either generates binary regression or classification trees based on whether the target variable is numeric or categorical. Optimal partitioning must be followed while performing partitioning of decision trees. In CART, same variables may be reused in the different parts of decision trees. Splitting may be binary or multiway.

In a binary splitting, each node is further divided into at most two subgroups, whereas in the case of multiway splitting, each node is further split into multiple subgroups. Decision trees are very easy to comprehend if we follow multiway splitting as a particular attribute rarely reappears while traversing a path from the root to the leaf.

There are several methods used for dealing with missing values in decision trees. Missing values may be ignored or may be assigned some other category.

Missing values instances may be distributed among the child nodes as follows:

1. Everything goes to a node having the largest number of instances.
2. Distribution is done among all child nodes but with minimum weights, which is proportional to the number of instances from every child node.
3. Distribution is done randomly according to the categorical distribution to a single child node.
4. Sort, build, and use input features that decide how the distribution of instances is done in a child node.

Instability in decision trees

The instability problem in a decision tree classifier means the resulting constructed rules might be different from the original or the actual ones if there is a modification in the training sample. This instability is caused due to invalid selection of the split candidate. The split candidate is found using a split evaluation function that can be used to partition the data. If at a particular stage, no dominant split is found, then the split candidate is used to partition the node. If a different split is selected at any stage, then it results in a tree which is absolutely different from the original tree. So, the selection of the correct splitting candidate is very important in order to prevent inaccuracy or instability in decision trees.

Conclusion

In this chapter, we learned about decision trees, the difference between regression trees and classification trees, stopping criterion and pruning loss functions in decision trees, categorical attributes, multiway splits, and missing values in decision trees, and instability in decision trees. In the next chapter, we will discuss about unsupervised learning, clustering and principle component analysis.

Questions

1. Explain the difference between classification trees and regression trees.
2. Explain the stopping criterion and pruning loss functions in decision trees.
3. What are multiway splits in decision trees?
4. Explain instability in decision trees.

Chapter 9
Unsupervised Learning

Introduction

Decision trees are a way to display conditional control statements. In the previous chapter, we discussed about decision trees, regression trees, stopping criterion and pruning loss functions in a decision tree, categorical attributes, multiway splits and missing values in decision trees, and instability in decision trees. Unsupervised learning is a kind of machine learning algorithm that can be used to draw useful conclusions without the presence of labeled responses in the input data. In the following chapter, we will discuss about Clustering (K-means Clustering, Hierarchical Clustering), and Principal Component Analysis.

Structure

Unsupervised learning is a complex processing task involving the identification of patterns in data sets having data points that are neither labeled nor classified. This chapter will comprise the following topics:

- Clustering
- Principal Component Analysis

Objectives

In unsupervised learning, an uncategorized and unlabeled data is sent to the AI system and the algorithms act on this data without any prior training. Following are the objectives of this chapter:

- To know about unsupervised learning
- Understanding Clustering and Clustering Algorithms (K-means Clustering, Hierarchical Clustering)
- Understanding Principal Component Analysis

Clustering

The term Clustering was first used in 1932 in an anthropology by *Driver* and *Kroeber*. It was used in 1938 in psychology by *Joseph Zubin* and in 1939 by *Robert Tryon*. It was used for trait theory classification in personality psychology in 1943 by *Cattell*. Clustering also referred to as cluster analysis is a process of grouping together similar objects into same group called as cluster in such a way that objects in one cluster are not similar to the objects in another cluster. Cluster analysis is used in many fields such as data compression, pattern recognition and image processing, machine learning, computer graphics, information retrieval, and bioinformatics. In the following chapter, we will discuss about clustering algorithms such as k-means clustering algorithm and hierarchical clustering algorithm.

K-means clustering

K-means clustering involves the partitioning of observations into k clusters in which a given observation belongs to the cluster having the closest mean. Consider the following code in python using k-means clustering:

```
print(__doc__)
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn import datasets

np.random.seed(5)

irisdata = datasets.load_iris()
P = irisdata.data
q = irisdata.target
```

In the following code, `n_init` is set to 1 instead of 10 which is a default value. So, this bad initialization has an impact on the classification process as it reduces the number of times an algorithm runs with different centroid seeds.

```
estimatorskmeans = [('k_means_iris_8', KMeans(n_
clusters=8)),
              ('k_means_iris_3', KMeans(n_clusters=3)),
              ('k_means_iris_bad_init', KMeans(n_
clusters=3, n_init=1,

init='random'))]
fignum=1
titles = ['8 clusters', '3 clusters', '3 clusters, bad
initialization']
for name, est in estimatorskmeans:
    fig = plt.figure(fignum,figsize=(4,3))
    px = Axes3D(fig, rect=[0, 0, .82,1], elev=52,azim=147)
    est.fit(P)
    labels = est.labels_

    px.scatter(P[:, 3], P[:, 0], P[:, 2],
               c=labels.astype(np.float), edgecolor='k')

    px.w_xaxis.set_ticklabels([])
    px.w_yaxis.set_ticklabels([])
    px.w_zaxis.set_ticklabels([])
    px.set_xlabel('Petal width')
    px.set_ylabel('Sepal length')
    px.set_zlabel('Petal length')
    px.set_title(titles[figno-1])
    px.dist = 12
    fignum=fignum+1
```

Plotting the ground truth values takes place:

```
fig = plt.figure(fignum, figsize=(4, 3))
px = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

for name, label in [('Setosa', 0),
                    ('Versicolour', 1),
                    ('Virginica', 2)]:
    px.text3D(P[q == label, 3].mean(),
              P[q == label, 0].mean(),
              P[q == label, 2].mean() + 2, name,
              horizontalalignment='center',
              bbox=dict(alpha=.2, edgecolor='w',
facecolor='w'))
```

Labels are reordered so that the colors are matched with the cluster results:

```
q= np.choose(q,[1, 3, 0]).astype(np.float)
px.scatter(P[:, 3], P[:, 0], P[:, 2], c=q, edgecolor='k')

px.w_xaxis.set_ticklabels([])
px.w_yaxis.set_ticklabels([])
px.w_zaxis.set_ticklabels([])
px.set_xlabel('Petal width')
px.set_ylabel('Sepal length')
px.set_zlabel('Petal length')
px.set_title('Ground Truth')
px.dist = 12

fig.show()
```

The output of the above code is shown in *figure 9.1*:

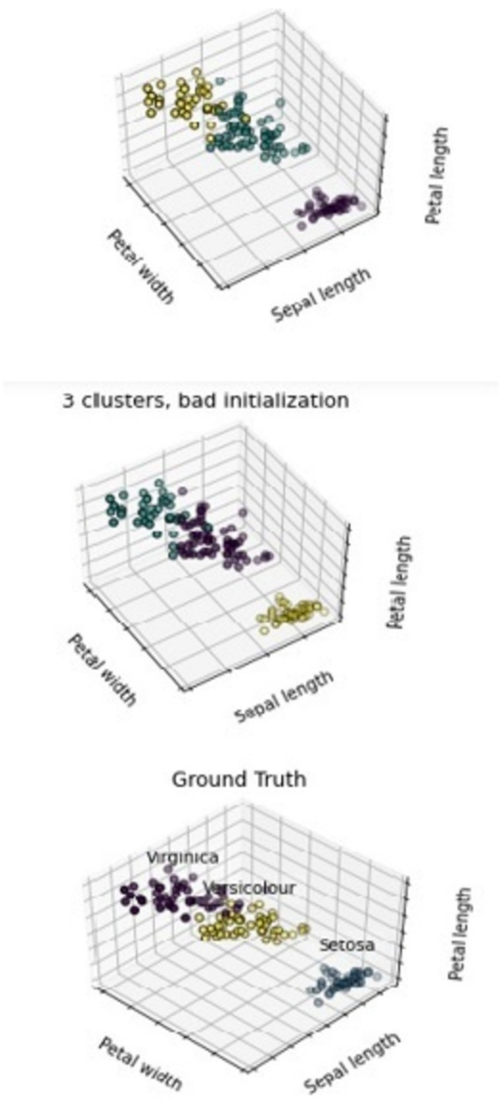


Figure 9.1: *K-means clustering*

Hierarchical clustering

Hierarchical clustering is also referred to as Hierarchical Cluster Analysis. This method involves building a hierarchy of clusters. Two approaches used in hierarchical clustering analysis include the following:

- Agglomerative clustering
- Divisive clustering

Agglomerative clustering is a 'bottom-up' approach that involves each individual cluster, and the pairs of clusters merge and move higher in the hierarchy. Divisive clustering is a 'top-down' approach in which a single cluster is further split into multiple clusters as we proceed down in the hierarchy.

Cluster dissimilarity is a metric that measures the distance between the pair of observations and the linkage criterion that specifies a dissimilarity between the pair of observations. Cluster dissimilarity is used in agglomerative clustering to decide which cluster would join together to form the bigger cluster. It is also used in divisive clustering to decide which cluster would finally break.

Principal Component Analysis (PCA)

PCA refers to the dimensionality reduction methodology that can be used for reducing the dimensions of large data sets into smaller ones, while still preserving as much as information as possible.

Steps performed in PCA include the following:

1. Standardization
2. Computation of Covariance Matrix
3. Identification of Principal Components by computing the Eigen Values and Eigen Vectors of Covariance Matrix
4. Creating a Feature Vector
5. Recasting the data

In the first step, standardization is important because if there are some values that are large and some that are small, then the larger ranges would dominate over the smaller ranges. So, standardization is performed to solve this problem. Standardization can be done by subtracting the mean from the value and dividing by the standard deviation. This is represented as follows:

Z=(value-mean)/standard deviation

In the second step, covariance matrix is computed to find whether there exists redundant information in the input data. In the third step, the principle components are generated, which are nothing but the variables that are formed by the linear combination of the initial variables. In the fourth step, a feature vector is created by considering those principle components that are of a higher significance over the ones with a lower significance.

Consider the following code in Python on PCA. We are considering a two-dimensional data set having 600 points:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
ranstat = np.random.RandomState(1)
P = np.dot(ranstat.rand(2, 2), ranstat.randn(2, 600)).T
plt.scatter(P[:, 0], P[:, 1])
plt.axis('equal');
```

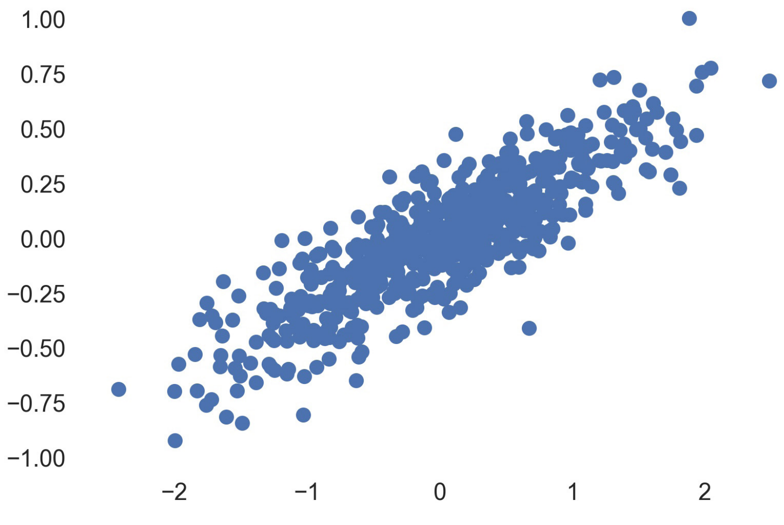


Figure 9.2: *Example on Principal Component Analysis*

In the above figure, there exists a linear relation between X and Y. In Principal Component Analysis, a list of principal axes is found and these axes are used to describe the data set. Using `scikit-learn`, we can compute the principal axes as follows:

```
from sklearn.decomposition import PCA
principalaxis = PCA(n_components=2)
principalaxis.fit(P)
PCA(copy=True, iterated_power='auto', n_components=2,
random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

Principal axis fit learns components and explained variance as follows:

```
print(principalaxis.components_)
[[ 0.9513521  0.3081058]
[ 0.3081058 -0.9513521]]

print(principalaxis.explained_variance_)
[0.70649509 0.02104542]

def draw_vector(n0, n1, px=None):
    px = px or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    px.annotate('', n1, n0, arrowprops=arrowprops)

# plot data
plt.scatter(P[:, 0], P[:, 1], alpha=0.2)
for length, vector in zip(principalaxis.explained_
variance_, principalaxis.components_):
    p = vector * 3 * np.sqrt(length)
    draw_vector(principalaxis.mean_, principalaxis.mean_ + p)
plt.axis('equal');
```

We can define the input data as vectors and represent the direction of a vector using the components and squared length of vector is defined using explained variance. The principal axis is represented by the data. The variance of the data is represented by the length of the vector. This is shown in the code given below:

```
def draw_vector(n0, n1, px=None):
    px = px or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    px.annotate('', n1, n0, arrowprops=arrowprops)

# plot data
plt.scatter(P[:, 0], P[:, 1], alpha=0.2)
for length, vector in zip(principalaxis.explained_
variance_, principalaxis.components_):
    p = vector * 3 * np.sqrt(length)
    draw_vector(principalaxis.mean_, principalaxis.mean_ + p)
plt.axis('equal');
```

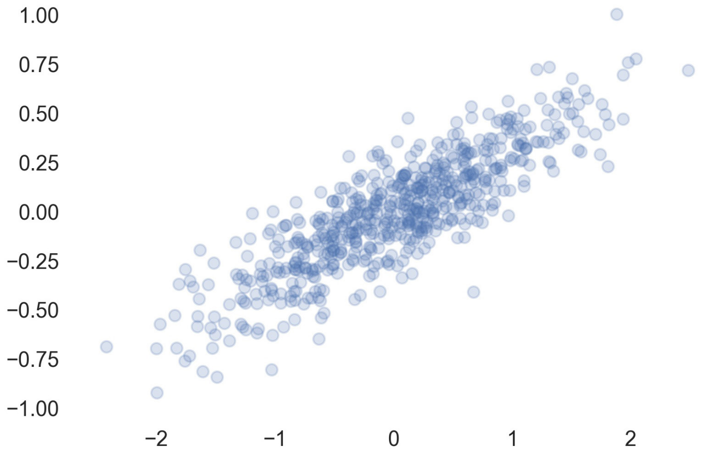


Figure 9.3: *Variance represented as the length of the vector*

The value of one or more of the principal components are made 'zero' in order to perform dimensionality reduction. Inverse transform on the reduced data can be found and the plot can be drawn on the original data.

```
principalaxis = PCA(n_components=1)
principalaxis.fit(P)
X_principalaxis = principalaxis.transform(P)
```

```
print("original shape:   ", P.shape)
print("transformed shape:", X_principalaxis.shape)
original shape:    (600, 2)
transformed shape: (600, 1)
P_new = principalaxis.inverse_transform(X_principalaxis)
plt.scatter(P[:, 0], P[:, 1], alpha=0.2)
plt.scatter(P_new[:, 0], P_new[:, 1], alpha=0.8)
plt.axis('equal');
```

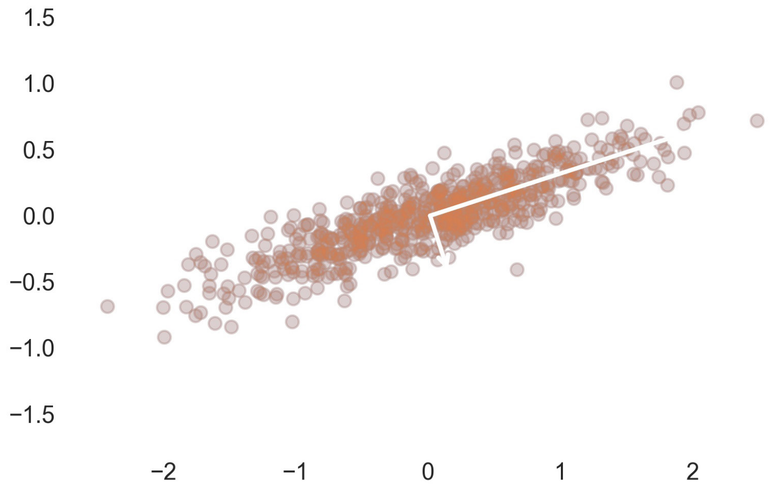


Figure 9.4: Inverse transform on the reduced data

Conclusion

In this chapter, we learnt about clustering, k-means clustering, hierarchical clustering, and Principal Component Analysis. In the next chapter, we will discuss the theory of generalization, training versus testing, bounding the testing error, Vapnik Chervonenkis inequality, VC Dimension, and the proof of VC inequality.

Questions

1. Explain k-means clustering with an example.
2. Explain hierarchical clustering.
3. Explain Principal Component Analysis with an example.

CHAPTER 10
Theory of Generalization

Introduction

Unsupervised learning is a kind of machine learning algorithm that can be used to draw useful conclusions without the presence of labeled responses in the input data. In the previous chapter, we discussed about clustering (k-means clustering, hierarchical clustering) and Principal Component Analysis. In the following chapter, we will discuss about training versus testing, bounding the testing error, and VC dimension.

Structure

Unsupervised learning is a complex processing task involving the identification of patterns in data sets having data points that are neither labeled nor classified. This chapter will comprise the following topics:

- Training versus testing
- Bounding the testing error
- VC dimension

Objectives

In unsupervised learning, an uncategorized and an unlabeled data is sent to the AI system, and algorithms act on this data without any prior training. Following are the objectives of this chapter:

- To know about the difference between training and testing in machine learning
- Understanding VC dimension

Training versus testing

Training data helps the algorithm to learn from experience. In supervised learning, each observation comprises input variable and the corresponding target variable. For building a model, a training set is implemented and for validating a test set, a testing set is required. The data set is divided into the training set and the test set.

In machine learning, a model is created in order to perform testing on the test data. To fit the model, training data is used and to perform testing, test data is used. It is not necessary to use 70% of the data set for developing the training set and the rest for the purpose of testing. It depends on the data set that is being used and the task that needs to be accomplished.

Bounding the testing error

Principal Component Analysis refers to the dimensionality reduction methodology that can be used for reducing the dimensions of large data sets into smaller ones, while still preserving as much as large amounts of information possible.

Steps performed in Principal Component Analysis include the following:

1. Standardization
2. Computation of Covariance Matrix
3. Identification of Principal Components by computing the Eigen Values and the Eigen Vectors of Covariance Matrix
4. Creating a Feature Vector
5. Recasting the data

The first step, Standardization, is important because if there are some values that are large and some small, then the larger ranges would dominate over the smaller ranges. So, Standardization is performed to solve this problem. Standardization can be done by subtracting the mean from the value and dividing by the standard deviation. This is represented as follows:

Z=(value-mean)/standard deviation

In the second step, the covariance matrix is computed to find whether there exists redundant i.

VC dimension

VC dimension was originally given by *Vladimir Vapnik* and *Alexey Chervonenkis*. VC dimension may be defined as a measure of some of the features in terms of complexity, flexibility, richness or expressive power of the set of functions that may be learned using statistical binary classification algorithm.

Uses of VC dimension include the following:

- VC dimension is used in the statistical learning theory for the prediction of the probabilistic upper bound of the test error of a classification model.
- VC dimension is also used in sample complexity bounds. Sample complexity may be defined as the linear function of the VC dimension of the hypothesis space.
- VC dimension is used in computational geometry for the prediction of the complexity of approximation algorithms.

Conclusion

In this chapter, we learned about training versus testing, bounding the testing error, and VC dimension. In the next chapter, we will discuss how to detect bias and how to fix bias or achieve fairness in ML.

Questions

1. Explain the difference between training and testing in machine learning.
2. Explain bounding the testing error.

Chapter 11
Bias and Fairness in Machine Learning

Introduction

In machine learning and AI, future predictions are based on past observations and bias is based on prior information. Harmful biases occur because of human biases which are learnt by an algorithm from the training data. In the previous chapter, we discussed about training versus testing, bounding the testing error, and VC dimension. In the following chapter, we will discuss about bias and fairness.

Structure

Human biases are most found in data sets such as medical, educational, criminal, text, financial, etc. Human bias is little or no impacted on the weather data. This chapter will comprise the following topics:

- Introduction of bias
- How to detect bias?
- How to fix bias or achieve fairness in ML?

Objectives

In machine learning, algorithmic biases are referred to as unwarranted associations. Algorithmic biases are the bugs that can be harmful to the business and people. Following are the objectives of this chapter:

- Understanding how to identify bias.
- Understanding how to achieve fairness in ML.

Introduction

Bias is referred to as a disproportionate prejudice or inclination towards a particular thing or an idea. Bias may be found in the following different fields:

- Research
- Statistics
- Social sciences

How to detect bias?

Bias is one of the popular topics that one encounters while building AI-based models. Many uncommon and common biases may be found in the following stages of AI model development:

- Data collection
- Data preprocessing
- Data analysis
- Modeling

During data collection, biases may take place. This happens due to the occurrence of outliers and errors that happen while collecting data.

Biases that are found during the data collection process include the following:

- **Selection bias**: While preparing the sample data, selection of data must be done in a proper manner to avoid bias. For example, if the participants are students who are to undergo tests, then they may include the bias results.
- **The Framing Effect**: Survey questions are framed in such a manner that biases are avoided and displays positivity in sentences, else biases crop up.

- **Systematic bias**: It occurs because of faulty equipment. It leads to repeatable and consistent errors.
- **Response bias**: It occurs due to questions that are answered incorrectly by the participants.

During data preprocessing, the following steps may be undertaken:

1. Outlier detection
2. Missing values
3. Filtering data

Outliers lead to a disproportionate effect on many of the analyses that are conducted.

While dealing with the missing values, if all the missing values are replaced by the mean values, then it would mean being biased towards a particular group that is closer to the mean.

Biases may be found during the process of data analysis. Biases may be found using the following approaches:

- **Missing graphs**: Incorrect conclusions may be drawn from a distorted graph that provides incorrect information.
- **Confirmation bias**: It involves the tendency to focus and confirm information that is related to someone's preconceptions.

When performing data modelling, it is very important to detect biases. For example, Amazon created a hiring algorithm that showed gender bias by favoring men as high potential candidates. A model that has high variance focuses on training data and doesn't generalize well. Data always behaves in the same way in high bias. When we increase bias, variance decreases and vice versa. In supervised machine learning, training is performed on the input variables in such a manner that there is closeness between the predicted values and the actual values. Error refers to the difference between the actual and the predicted values. There are 3 types of errors in supervised machine learning:

- Bias error
- Variance error
- Noise

Bias and variance are reducible errors that we can minimize to a large extent. Noise is said to be an irreducible error that cannot be eliminated.

How to fix biases or achieve fairness in ML?

There are already many definitions of fairness as per literature and these cover the following elements:

- Equalized odds
- Unawareness
- Individual fairness
- Demographic parity
- Counterfactual fairness
- Predictive rate parity

We should avoid including sensitive attribute as one of the features in training data. There are many ways to mitigate biases. Some of these techniques include:

- Preprocessing
- In-processing
- Post-processing

The pre-processing approach takes place before the development of a model. Its main intent is to eliminate the underlying bias from the data set before modelling. This is one of the basic approaches of removing biases from the data. In-processing is the process of removing biases during the training phase. In post-processing, elimination of biases takes place after the training phase is over.

Conclusion

In this chapter, we learnt about biases, how to detect bias, and how to fix bias and achieve fairness.

Questions

1. Explain how we can detect bias.
2. Explain how we can fix bias.

Index