

Computer Communications and Networks

K. G. Srinivasa
Siddesh G. M.
Srinidhi H.

Network Data Analytics

A Hands-On Approach for Application
Development



Computer Communications and Networks

Series editors

Jacek Rak, Department of Computer Communications, Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Gdansk, Poland

A. J. Sammes, Cyber Security Centre, Faculty of Technology, De Montford University, Leicester, UK

The **Computer Communications and Networks** series is a range of textbooks, monographs and handbooks. It sets out to provide students, researchers, and non-specialists alike with a sure grounding in current knowledge, together with comprehensible access to the latest developments in computer communications and networking.

Emphasis is placed on clear and explanatory styles that support a tutorial approach, so that even the most complex of topics is presented in a lucid and intelligible manner.

More information about this series at <http://www.springer.com/series/4198>

K. G. Srinivasa · Siddesh G. M.
Srinidhi H.

Network Data Analytics

A Hands-On Approach for Application
Development



Springer

K. G. Srinivasa
Department of Information Technology
Ch. Brahm Prakash Government
Engineering College
Jaffarpur, Delhi
India

Srinidhi H.
Department of Computer Science
and Engineering
Ramaiah Institute of Technology
Bangalore, Karnataka
India

Siddesh G. M.
Department of Information Science
and Engineering
Ramaiah Institute of Technology
Bangalore, Karnataka
India

ISSN 1617-7975 ISSN 2197-8433 (electronic)
Computer Communications and Networks
ISBN 978-3-319-77799-3 ISBN 978-3-319-77800-6 (eBook)
<https://doi.org/10.1007/978-3-319-77800-6>

Library of Congress Control Number: 2018937700

© Springer International Publishing AG 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer International Publishing AG
part of Springer Nature
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Foreword

In this book titled network data analytics, the authors have consolidated different tools and techniques used for Big data analytics. The book gives an insight into the various technologies along with related concepts used to analyze Big data. In these days, data continues to grow exponentially and much of the data is implicitly or explicitly imprecise. Data analysis seeks to analyze and predict noteworthy, unrecognized relations among different attributes in the entire dataset.

The Internet has reached across various technological platforms and millions of people in the world. With this spread, proliferating, new technologies and devices have emerged. One of such example is smartphones that have reached almost every person in the world. Other examples include smartwatch, smart homes equipped with sensing technologies for smart energy and water management. The Internet lies as the backbone for all of these devices and technologies. Cloud computing on the other hand also depends on the Internet for providing different types of services and infrastructures. The interconnection of devices and cloud equipped with Internet leads to new business cases and smart solutions leading the way to Internet of Things (IoT). Once the devices are interconnected with one another, the service needs to be delivered to the end user. A thing in IoT can be a device such as heat sensor or a cloud service that detects the heating level or the person itself requesting the service. Hence, network data analytics has become the need of the day.

In order to carry out data analytics, we need powerful and flexible computing software. However, the software available for data analytics is often proprietary and can be expensive. This book reviews Apache tools, which are open source and easy to use. After providing an overview of the background of data analytics, covering different types of analytics and the basics of using Hadoop as a tool, the book focuses on different Hadoop ecosystem tools, like Apache Flume, Apache Spark, Apache Storm, Apache Hive, R, and Python, which can be used for different types of analytics. It then examines the different machine learning techniques that are useful for data analytics, and how to visualize data with different graphs and charts.

Presenting data analytics from a practice-oriented viewpoint, the book discusses useful tools and approaches for data analytics, supported by concrete code examples. The book is a valuable reference resource for graduate students and

professionals in related fields, and is also of interest to general readers with an understanding of data analytics.

With the importance of data analytics in recent years, this textbook gives valuable directions to the application developers and researchers in this domain. The authors are well-known writers in the field of computer science and engineering and present state-of-the-art technology in the area of network data analytics.

Bangalore, India

Prof. L. M. Patnaik
Adjunct Professor and INSA Senior Scientist
National Institute of Advanced Studies
(www.lmpatnaik.in)

Preface

Recent advances in computing has lead to diverse applications in wide domains such as cyber-forensics, data science, business analytics, business intelligence, computer security, web technology, and Big data analytics. Data analytics refer to broad term where many of the areas such as cyber-physical systems (CPS), Internet of Things (IoT), Big data, machine learning, and data mining overlap among each other. However, there are subtle differences among these areas. Data analytics form as one of the key components of these wide areas in computing. For example, in IoT the data that are collected from various devices need to be analyzed for inferring the outcomes of it. Hence, data analytics need to be carried on it. There are various platforms available for data analytics. This book provides a brief overview of the term data analytics, different types of data, different types of analytics, and the analytical architecture.

About the Book

Data analytics lifecycle phases involve many steps such as variable selection, developing hypothesis, perform analytics, and visualization. The main tool/platform used for data analytics is Hadoop. Hadoop is an open-source project under Apache foundation and is used for processing large volume of data in a distributed manner. Hadoop and MapReduce programming are discussed in this book with real-life examples. Similarly, Apache Hive, a data warehouse solution under the Hadoop ecosystem, which provides an SQL kind of language called as HiveQL for processing the data, is detailed in this book. Apache Spark is one of the other platforms that helps in analyzing large massive datasets and more advantageous than Hadoop and Hive. The drawback of Hadoop is batch processing of data using MapReduce programming model which is eliminated in Apache Spark. Apache Pig is one of the Hadoop ecosystem tools that provide a workflow like MapReduce for analytics. As it is difficult for the developers to use different programming models, Apache Pig provides a workflow for analysis of large datasets. The book details about

programming with Apache Spark and Pig. Real-time processing of events plays an important role in data analytics. Recent advances in Internet have to lead to the rise of social media with massively large data generated in real time. Analysis of such data leads to interesting scenarios where some of the business decisions can be made. Apache Flume is one of the tools in the Hadoop ecosystem that provides a platform for real-time data analytics. Similarly, social media computing plays an important role in the fields of digital marketing and advertising. The companies collect reviews of different products from different social networking sites and infer decisions about it. For analysis of such real-time data, Apache Storm need to be used. The book explains fundamentals of Apache Flume and Storm.

Machine learning is one of the computing fields that have emerged over the years in various applications like text analytics, speech recognition, fraud detection in financial transactions, retail applications, and others. It forms as the basis step for analytics. The different types of machine learning techniques are regression, classification, clustering, and others. Regression is one of the machine learning techniques used for forecasting the future values. The main aim of regression analysis is to find the influence of a variable on the other set of variables in the dataset. It involves two types of variables, namely dependent and independent variables. Regression analysis is used for the applications like forecasting performance in cars, predicting the profit in stocks and others. Classification is one of the fundamental machine learning techniques that is used for differentiating the data into various classes. It is a supervised learning technique, wherein a classifier assigns label or class to a set of data points based on the data that are classified with labels. The process of classification starts with a training set of labeled data and assigned classes, and these are used to predict the class of unknown data points or new data. However, there are other analytical techniques that play an important role in data analysis such as clustering, association rule mining, random forest, principal component analysis, and logistic regression. Clustering is an analytical technique that groups similar objects into two or more groups using the different data attributes and features. Association rule mining is widely used in e-markets for enhancing the customer purchases by providing the suitable offers for the customers. Random forest is a classification technique that builds a number of decision trees based on the decision points in the classification. The book briefs the important concepts of machine learning and their implementation on real-time data.

Some of the application areas of data analytics that are included in the book are as follows:

- Most of the data generated appear in the form of text. Text analytics is the process of getting insights from the text by focusing on the small pieces of information in the text. The different applications of text analytics are sentimental analysis, spam classification, and automatic summarization.
- IoT has to be aligned to the data analytical platforms like Hadoop and Hive for developing applications. The different applications of IoT include smart grid, smart mobility, activity analysis, and air-quality analysis. Activity analysis consists of analyzing the typical activities of a person like walking, running, and

sleeping. Air-quality analysis involves analyzing the different components of the air such as CO₂, NO₂, and others for monitoring. In this chapter, firstly an overview of IoT and its components are highlighted followed by the case studies related to it.

- Data analytics include applications like object recognition, video surveillance, self-driving cars, and tracking objects. These applications are advanced analytical applications, and machine learning-based classification models cannot be used. Advanced classification models include neural networks as the basis for analytics. TensorFlow is one of the analytical tools that help in developing advanced analytical applications on image and video analytics. It helps to build neural networks with the required number of hidden layers for the model.
- Recommendation systems are advanced data analytical applications where the users are provided with various recommendations based on their preferences. They are used in various online retail sites for identifying the purchasing behavior of the users and ratings of different products.

Data visualization is one of the key aspects in a typical data analytical project. Data visualization for different categories of data varies, and different types of charts need to be created. The main aim of data visualization is to identify the patterns that are hidden behind the data. A visualization of data rather than the descriptive analytics provides a way for identifying the insights about the data. However, there are certain key principles that data visualization should adhere. Some of the key principles of data visualization are diversification, simplified, easy to understand, and comparison. The popular visualization tools used are Python, R, and Tableau. This book provides an overview data visualization.

Key Features

- Provides an in-depth introduction to data analytics that is easy to follow for both beginners and experts who want to get started with building data analytics-based applications.
- The chapters are created in such a way that users can jump to any concept they want to learn and do not need to follow the book linearly strictly.
- It covers all the topics required to be a data scientist.
- Provides numerous examples of real-life systems with a complete code.

Contents and Coverage

The chapter-wise scheme of the book is given here.

Chapter 1 provides a brief overview of the term data analytics, different types of data, different types of analytics, and the analytical architecture is first discussed. It

is later followed by the different phases that are involved in the lifecycle of the data analytics project and the interconnection of Big data and Hadoop ecosystem.

Chapter 2 introduces Hadoop and MapReduce programming aspects are discussed with examples. The highlights of the chapter include case studies like retail analytics and network log analytics in Hadoop with MapReduce.

Chapter 3 introduces Hive, and its architectural components are discussed first. Later, the chapter is followed with different kinds of operations that can be executed in Hive and examples on it. The chapter concludes with the network log and call log case studies with Hive.

Chapter 4 introduces Apache Spark that is discussed with its architectural elements and examples. The core elements of the Spark, text search, and retail analytics examples are discussed with Spark in this chapter.

Chapter 5 introduces Apache Pig that is discussed with an overview of architecture, data flow models, and Pig Latin functions with examples.

Chapter 6 introduces an overview of Apache Flume, and its architectural components with workflow are discussed. Later, the configuration of Flume with Twitter social network is discussed as an example for real-time analytics.

Chapter 7 introduces Apache Storm that is discussed with its architectural elements and examples. The configuration of Apache Storm with Twitter networking site is discussed as an example of collection and analysis of hash tags.

Chapter 8 introduces the basics of machine learning that are introduced with its key terminologies and its tasks. The different types of tasks that are involved in machine learning are data acquisition, data cleaning, data modeling, and data visualization. These tasks are discussed in this chapter with steps on getting started with machine learning.

Chapter 9 discusses the method for finding the best fit in regression using examples.

Chapter 10 presents classification methods, such as Naïve Bayes, decision tree, and support vector machines (SVM) that are discussed with examples. Naïve Bayes is a supervised classification method based on the conditional probability. Decision tree is a classification technique where the tree is built based on certain number of conditions. SVM is a classification technique used where there are multiple classes for classification.

Chapter 11 discusses association rule-based analytical techniques.

Chapter 12 presents different stages of text analytics which is first discussed and later followed by the case studies. Three case studies that are discussed as a part of this chapter are automatic summarization, spam classification, question classification, and sentimental analysis.

Chapter 13 provides an overview of IoT and its components followed by the case studies related to it. Two case studies, namely air-quality analysis and activity analysis, are discussed as a part of this chapter.

Chapter 14 provides an overview of TensorFlow. Image analytics with MNIST data is discussed first where the handwritten digits are recognized using a TensorFlow model. Later the case studies on spam classification and question

classification are revisited once again that were a part of machine learning. The main aim of the chapter is to discuss the TensorFlow and its applications in analytics.

Chapter 15 presents an overview of recommendation system followed by the matrix factorization method. A case study on the MovieLens dataset is discussed as an example for recommendation system with Apache Spark.

Chapter 16 gives an overview of data visualization with its importance and key steps for the visualization.

Chapter 17 provides the configuration of matplotlib and graphviz module that is discussed with small examples in Python.

Chapter 18 discusses the different types of charts and its visualization with examples in Python.

Chapter 19 presents the different types of data visualization charts with examples.

Delhi, India
Bangalore, India
Bangalore, India

K. G. Srinivasa
Siddesh G. M.
Srinidhi H.

Acknowledgements

We attribute our efforts for completing this book to all the people who have inspired us and shaped our careers. We thank our college management, colleagues, and students who encouraged us to work on this book.

Srinivasa would like to thank Prof. Vijay K. Minocha, Principal of Ch. Brahm Prakash Government Engineering College, New Delhi, who encouraged working on this book. His special thanks to his esteemed colleagues Prof. Harne, Dr. Athar Hussain, Pankaj Lathar, and Seema Rani for their valuable suggestions and continuous encouragement.

The authors would like to thank Dr. N. V. R. Naidu, Principal, Ramaiah Institute of Technology, for his valuable guidance and support. Special thanks to Dr. T. V. Suresh Kumar, Dr. Anita Kanavalli, Dr. B. P. Vijay Kumar, and Mr. Ramesh Naik for their continuous encouragement. The authors would like to acknowledge valuable inputs and suggestions by Sowmya B. J., Chetan Shetty, and Hanumantha Raju, Ramaiah Institute of Technology.

We are extremely grateful to our families, who graciously accepted our inability to attend to family chores during the course of writing this book, and especially for their extended warmth and encouragement. Without their support, we would not have been able to venture into writing this book.

We acknowledge the UCI machine learning repository as the datasets from the repository are adopted in the book for discussing the different concepts of data analytics.

Last but not the least, we express our heartfelt thanks to the editorial team at the Springer, who guided us through this project.

Contents

Part I Introduction to Data Analytics

1	Introduction to Data Analytics	3
1.1	What Is Data Analytics?	3
1.1.1	What Is Analytics?	3
1.1.2	What Is Data?	5
1.2	Data Analytical Architecture	8
1.3	Lifecycle of a Data Analytics Project	10
1.3.1	Objective Definition	12
1.3.2	Understanding Data and Requirements	14
1.3.3	Data Cleaning	17
1.3.4	Perform Analytics	19
1.3.5	Results Visualization	22
1.3.6	Put the Model into Operation	24
1.4	Big Data and Ecosystem	24
1.5	Exercises	27
	References	27
2	Hadoop	29
2.1	Introduction	29
2.2	Hadoop Architecture	30
2.2.1	NameNode	31
2.2.2	Data Node	31
2.3	Memory and Files in Hadoop	32
2.4	Replication in Hadoop	32
2.5	MapReduce	33
2.6	MapReduce and Hadoop Workflow	34
2.7	Installing Hadoop	36
2.8	MapReduce Examples	37
2.8.1	Word Count Example in Hadoop	37
2.8.2	Retail Analytics Using Hadoop	39

2.9	Call Log Analytics Using Hadoop	42
2.9.1	Call Log Analytics Random File Input	42
2.9.2	Map Function for Call Log Analytics Example in Hadoop	42
2.9.3	Reduce Function for Call Log Analytics Example in Hadoop	44
2.9.4	Execution of Call Log Analytics Example in Hadoop	45
2.10	Network Log Analytics Using Hadoop	48
2.10.1	Network Log Analytics Random File Input	48
2.10.2	Map Function for Network Log Analytics Example in Hadoop	49
2.10.3	Reduce Function for Network Log Analytics Example in Hadoop	50
2.10.4	Execution of Network Log Analytics Example in Hadoop	51
2.11	Exercises	52
	References	53
3	Apache Hive	55
3.1	Introduction	55
3.2	Hive Architecture	56
3.3	Installing Hive	58
3.4	Hive Data Types	59
3.5	Hive Data Description (DDL)	59
3.5.1	Creating Database	59
3.5.2	Using Database	60
3.5.3	Create Table	61
3.6	Hive Operations	62
3.6.1	Load the Data	62
3.6.2	Data Exploration Using Select	63
3.7	Hive Examples on Network Log	64
3.7.1	Case Study 1: Call Data Record	64
3.7.2	Case Study 2: Network Log	68
3.8	Exercises	71
	References	72
4	Apache Spark	73
4.1	Introduction	73
4.2	Working of Spark Application	73
4.3	Installing Spark	76
4.4	Resilient Distributed Datasets	77
4.5	RDD Representation and Job Scheduling	78

4.6	Spark Examples	78
4.6.1	Word Count with Caching	79
4.6.2	Estimation of Pi	80
4.6.3	Filter Log File	81
4.6.4	Retail Analytics Revisited in Spark	82
4.7	Exercises	83
	References	83
5	Pig	85
5.1	Introduction	85
5.2	Installing Pig	86
5.3	Architecture of Pig in Hadoop	86
5.4	Modes and Working of Pig	87
5.5	Dataflow Language	88
5.6	Pig Latin: Functions	89
5.7	Exercises	93
	References	94
6	Apache Flume	95
6.1	Introduction	95
6.2	Background	96
6.2.1	Log File/Events	96
6.2.2	Log File Processing Using Hadoop and Its Drawbacks	97
6.3	Flume Architecture	97
6.3.1	Log/Event Data Processing in Flume	99
6.3.2	Failure and Event Handling in Flume	100
6.4	Installing Apache Flume	101
6.5	Flume in Action	102
6.5.1	Naming of Source, Sink, and Channels	102
6.5.2	Defining Source	103
6.5.3	Defining Sink	103
6.5.4	Defining Channel	104
6.5.5	Binding Source and Sink to Channel	105
6.5.6	Flume Agent Start-up	105
6.5.7	Configuring Flume with Twitter	106
6.6	Exercises	106
	References	107
7	Storm	109
7.1	Introduction	109
7.2	Components of Storm	111
7.3	Storm Architecture	113
7.4	Installation of Apache Storm	114

7.5	Storm in Action	115
7.5.1	Spout Creation	116
7.5.2	Bolt Creation	117
7.6	Twitter Analysis Using Storm	117
7.6.1	Spout Creation	117
7.6.2	Bolt Creation: Identify Hashtags	119
7.6.3	Bolt Creation: To Count HashTags	120
7.6.4	Execution of Topology	121
7.7	Exercises	122
	References	123

Part II Machine Learning

8	Basics of Machine Learning	127
8.1	Introduction	127
8.2	Tasks in Machine Learning	128
8.3	Data Mining and Machine Learning	130
8.4	Machine Learning Terminology	131
8.4.1	Supervised Learning	132
8.4.2	Unsupervised Learning	132
8.4.3	Attributes	133
8.4.4	Model	133
8.4.5	Accuracy	134
8.4.6	Prediction	134
8.5	Getting Started with Machine Learning in Python	135
8.5.1	Getting Started with Scikit-Learn	136
8.6	Exercises	137
	References	138
9	Regression	139
9.1	Introduction	139
9.2	Simple Linear Regression	139
9.3	Linear Regression in Python Example	140
9.4	Polynomial Regression	142
9.5	Finding the Best Fit for Regression	147
9.6	Regression Case Study on Mtcars Dataset	151
9.7	Ridge Regression	153
9.8	Exercises	154
	References	154
10	Classification	155
10.1	Naïve Bayes	155
10.1.1	Introduction	155
10.1.2	Bayes Theorem	156
10.1.3	Naïve Bayes Classification in Python	157
10.1.4	Diagnostics for Improvement of Classifier	159
10.1.5	Case Study on Naïve Bayes Classification	161

10.2	Decision Tree	164
10.2.1	Basic Elements of Decision Tree	164
10.2.2	Decision Tree Classification Example in Python	165
10.2.3	Decision Tree Classification on Iris Dataset	168
10.3	Support Vector Machine (SVM)	169
10.3.1	Linear Classifiers	170
10.3.2	Non-linear Classifiers	171
10.3.3	SVM Case Study on Iris Dataset	171
10.4	Exercises	174
	References	175
11	Other Analytical Techniques	177
11.1	Clustering	177
11.1.1	K-Means	177
11.1.2	Elbow Curve Using Within Sum of Squares	178
11.1.3	Clustering Example	181
11.1.4	Clustering Case Study on Student Marks	184
11.2	Association Rule Mining	190
11.2.1	Association Learning Basics	190
11.2.2	Apriori Algorithm	191
11.2.3	Apriori Example	192
11.2.4	Association Rule Mining Case Study on Market Basket Analysis	194
11.3	Principal Component Analysis	195
11.3.1	Key Terms in PCA	196
11.3.2	Steps Involved in Implementing PCA	197
11.3.3	Principal Component Analysis in Python	197
11.3.4	PCA Visualization in Python	199
11.4	Random Forest	200
11.4.1	Classification Using Random Forest	201
11.4.2	Random Forest and Regression	202
11.4.3	Missing Value Replacement for Training Set in Random Forest	202
11.4.4	Random Forest Using Python	203
11.4.5	Visualization of Random Forest in Python	204
11.5	Logistic Regression	206
11.5.1	Logistic Regression in Python	207
11.5.2	Multinomial Logistic Regression	210
11.5.3	Multinomial Logistic Regression Using Iris Dataset	210
11.6	Handling Missing Values	213
11.6.1	Deleting Rows with Missing Values	213
11.6.2	Deleting Columns with Large Number of Missing Values	213

11.6.3	Averaging Techniques	213
11.6.4	Predictive Techniques	214
11.6.5	Handling Missing Values in Python	214
11.7	Exercises	215
	References	216

Part III Advanced Analytics

12	Text Analytics	219
12.1	Introduction to Text Analytics	219
12.2	Steps Involved in Text Analytics	219
12.3	Names, Numbers, and Stopwords Removal	221
12.4	Word Frequency Analysis	222
12.5	Generating Bag of Words	224
12.6	Word to Vector Model	227
12.7	Word Cloud Model	231
12.7.1	Word Cloud Model in Python	232
12.8	Automatic Summarization Case Study	234
12.8.1	Supervised and Unsupervised Summarization	235
12.8.2	Abstract and Extractive Summarization	235
12.8.3	Sentence Extraction	236
12.8.4	Auto Summarization in Python	236
12.9	Spam Classification Case Study	240
12.9.1	Spam Classification with Naïve Bayes in Python	240
12.9.2	Spam Word Cloud in Python	240
12.9.3	Spam Filter in Python	244
12.9.4	Spam Classification with SVM	247
12.10	Question Classification Case Study	249
12.10.1	Pre-processing	250
12.10.2	Creating Features	252
12.10.3	SVM Classifier	253
12.11	Sentimental Analysis	254
12.11.1	Introduction	254
12.11.2	Different Types of Sentimental Analysis	255
12.11.3	Sentimental Analysis in Python	256
12.12	Exercises	263
	References	263
13	Internet of Things and Analytics	265
13.1	Introduction to IoT	265
13.2	Components of IoT	267
13.3	Analytics and IoT	268
13.3.1	Big Data Analytics	269
13.3.2	Real-Time Analytics	270
13.3.3	Bringing Analytics to IoT	270

13.4	Air Quality Analysis	270
13.4.1	Normalize Air Quality Data for Regression	272
13.4.2	Regression Model for Air Quality Data	274
13.5	Activity Analysis	276
13.5.1	Activity Plots	277
13.5.2	Activity Count	279
13.6	Exercises	281
	References	281
14	Advanced Analytics with TensorFlow	283
14.1	Introduction to TensorFlow	283
14.2	Installing TensorFlow	284
14.3	TensorFlow Case Study Using MNIST Data	288
14.4	Spam Classification Revisited Using TensorFlow	291
14.4.1	Preprocessing	291
14.4.2	Creating Features	292
14.4.3	TensorFlow Classifier	293
14.5	Question Classification Revisited with TensorFlow	296
14.5.1	Preprocessing	297
14.5.2	Creating Features	298
14.5.3	TensorFlow Classifier	299
14.6	Exercises	301
	References	302
15	Recommendation Systems	303
15.1	Introduction	303
15.2	Types of Recommendation Systems	305
15.2.1	Content-Based Recommender Systems	305
15.2.2	Knowledge Based Recommendation Systems	305
15.2.3	Collaborative Filtering Systems	306
15.3	Collaborative Filtering Using Matrix Factorization	306
15.3.1	Matrix Factorization Example	307
15.3.2	Matrix Factorization in Python	308
15.4	Building Recommendation Systems Using Movielens 100k Dataset	310
15.4.1	Extracting the Features of Movielens 100k Dataset	310
15.4.2	Training the Model for Recommendation	312
15.4.3	Matrix Factorization for Building Recommendation on Movielens Dataset	313
15.4.4	Provide Recommendations	314
15.4.5	Analytics on the Results of 100k Movielens Data	314
15.5	Movie Recommendation Using Spark	315
15.6	Exercises	318
	References	318

Part IV Data Visualization

16 Introduction to Data Visualization	321
16.1 Introduction to Data Visualization	321
16.2 Importance of Data Visualization	322
16.3 Principles of Data Visualization	324
16.4 Popular Visualization Tools	327
16.5 Exercises	330
References	331
17 Getting Started with Visualization in Python	333
17.1 Matplotlib	333
17.2 Graphviz	334
17.3 Installing Matplotlib in Windows	334
17.4 Installing Matplotlib in Ubuntu	334
17.5 Small Examples with Matplotlib	335
17.6 Exercises	337
References	337
18 Visualization Charts	339
18.1 Comparison Charts	339
18.1.1 Comparison Bar Charts	340
18.1.2 Comparison Line Charts	341
18.1.3 Comparison Charts for Brand Analysis	342
18.2 Composition Charts	344
18.2.1 Composition Chart Stacked Bar	346
18.2.2 Composition Chart on Crime Analysis	347
18.3 Distribution Charts	348
18.3.1 Distribution Chart with Line Histogram Example . . .	350
18.3.2 Distribution Chart with Intelligent Persons Dataset	351
18.4 Relationship Charts	353
18.4.1 Relationship Chart with Scatter Plot	353
18.4.2 Relationship Chart with Bubble Plot	354
18.4.3 Relationship Chart Scatter Groups	355
18.4.4 Relationship Chart for Population Analysis	356
18.5 Exercises	358
References	358
19 Advanced Visualization	361
19.1 3D Bar Plot	361
19.2 3D Histogram	363
19.3 3D Contour Plot	364
19.3.1 Contour Plot for Housing Data	366
19.4 3D Scatter Plot	368
19.5 Gantt Plot	369

19.6	Box–Whisker Plot	370
19.6.1	Box Plot on Housing Data	373
19.7	Polar Plot	374
19.8	Dendrograms	375
19.9	Heatmaps	377
19.10	Pyramid Charts	379
19.11	Radar Plots	381
19.12	Exercises	382
	References	383
	Appendix: Installing Python	385
	Index	397

Abbreviations

BOW	Bag of words
CDR	Call data record
DDL	Data description language
DML	Data manipulation language
HDFS	Hadoop distributed file system
IMEI	International mobile equipment identity
IMSI	International mobile subscriber identity
MNIST	Modified National Institute of Standards and Technology database
PCA	Principal component analysis
RDD	Resilient distributed datasets
SVM	Support vector machines

Part I

Introduction to Data Analytics

Chapter 1

Introduction to Data Analytics



The smart era of computing has lead to the creation of data everywhere and constantly at increasing rate. The main sources of data increase are due to increased number of users of mobile phones and social media. Internet of Things (IoT) and sensors are also one of the main reasons for the increased amount of data. It becomes highly challenging to monitor, store, and analyze such data. Thus, data analytics plays an important role in the diverse fields of engineering. In this chapter, an overview of data analytics and its lifecycle with different perspectives are discussed.

1.1 What Is Data Analytics?

Data analytics is a broad term used in the field of computing. There are different areas in the computing field where data analytics form a central component. Some of the areas are CPS, Big data, IoT, and others. Before understanding the term data analytics, the two keywords ‘data’ and ‘analytics’ are dealt separately first, and then, the actual term ‘data analytics’ is discussed. Firstly, a brief overview of analytics is dealt in the following section.

1.1.1 What Is Analytics?

Over the past years, analytics has been helpful in various projects to come up with answers for various questions. Analytics helps in answering the following questions and not on the basis of guesses or institution.

- What happened?
- How or why did it happen?
- What is happening now?
- What is likely to happen next?

Analytics process consists of data which may be a simple file or a collection of data files stored in a database. As the data grow, database size needs to be increased or has to be stored in some other locations. Cloud computing and data warehouses help in this context for storing data and carry analysis on it. Analytics can be defined as the process of making inferences by the use of statistical methods (mean, median, mode), software systems (data mining, machine learning), visualization to identify the patterns in data. In simple, the terms analytics are the process of converting data into useful information. It is interdisciplinary and can be applied to any of the fields in science and engineering [1]. For example, in the field of genomics, it can be used to identify the genes responsible for a characteristic, and it can be used to weather patterns in atmospherical sciences. Analytics can be broadly classified into following three types [2].

- Descriptive analytics,
- Predictive analytics,
- Perspective analytics.

1.1.1.1 Descriptive Analytics

As the name suggests, descriptive analytics describe the data in a summarized way that can be interpreted in a simple manner. In most of the scenarios, it describes the past data or historical data. It is mainly used to see the influence of past data on the future outcomes. Majority of the statistical methods are used for descriptive analytics such as sum, average, and percentage.

The main aim of the descriptive analytics problem will be usually to find the count of a particular column data. Some of the scenarios where descriptive analytics performed are average sales per month, average number of particular products sold, total number of stocks that use the past historical data for analysis to gain insights such as operations, sales, financial transactions, and customers.

1.1.1.2 Predictive Analytics

Predictive analytics is a type of analytics that helps to provide the predictions on future outcomes and helps in understanding the future decision making. The foundations of the predictive analytics lie in probability analysis. It uses data from various sources and may perform ETL processing if needed and then identify the patterns and relationships among the data. Most common applications of predictive analytics include route prediction in GPS systems, forecasting the air quality, predicting the prices of the stock analysis, weather forecasting, etc. [3].

1.1.1.3 Prescriptive Analytics

Prescriptive analytics uses both descriptive analytics and predictive analytics to prescribe numbers of actions that are suggested for making the possible outcomes. Applications in prescriptive analytics are related to energy, health care, etc. The main aim of prescriptive analytics is to suggest actions for decision. For example, in healthcare systems, it can help in the improvement of patient care systems by addressing the issues related to patient. The issues can be addressed by information processing of patients data like condition of rooms and medicine quality and suggest changes based on this information.

In a GPS system, the prescriptive analytics provides information on the real-time traffic and suggests taking an alternative route to reach the destination quickly. Though it seems to be an overlap between the predictive and prescriptive analytics, real-time data are involved in the case of prescriptive analytics compared to predictive analytics scenario. Table 1.1 highlights the different types of analytics with their summary on its description.

1.1.2 What Is Data?

A data can be any factual information that exists in various formats such as binary, text, audio, video, Web, image. Unless and until analytics is carried out on data, any valuable information cannot be gained. Computing that is driven by data has changed this world in terms of interaction with devices and humans. In day-to-day life, humans are most dependent on the devices and the inferences drawn from them. For example, people use GPS navigation services for daily commuting purpose. Such type of services is possible through analytics of various types of data

Table 1.1 Summary on different types of analytics

Type of Analytics	Description
Descriptive analytics	This type of analytics describes the information about the data using simple statistics. For example, in the case of population estimation, it provides the information on the number of people belonging to a particular age and addresses the issues related to youth and seniors separately
Predictive analytics	This type of analytics provides valuable information of the data using advanced statistical methods, machine learning techniques and builds a model for future predictions. The key relationships are identified among the features of the data for predictive analysis. For example, regression analysis is performed to see whether a customer subscribes to an insurance policy or not depending on the age, income, gender, and country he/she belongs to
Prescriptive analytics	In this type of analytics, management and decision-making process strategies are employed to gain valuable insights from the data. For example, specific advertising of the target customers can be handled with linear programming and operation research methods

involved such as different points in a city and connecting roads between them. When carrying out analytics, three types of data that can be classified are [4]:

- Categorical data,
- Numerical data,
- Ordinal data.

1.1.2.1 Categorical Data

Categorical data refers to the characteristics present in the data. For example, it can be gender, type of movie, martial status, etc. The categorical data are often represented by values such as 0 or 1. The value represented by the categorical data does not have any meaning. So, the alternative way of expressing the categorical data can also be Yes/No. For example, if gender is present in the dataset, it can be classified as 0 (female) and 1 (male), but Yes/No might not be suitable for analysis. But, in the case of martial status, Yes (Married)/No (Unmarried) and 0 (Unmarried)/1 (Married) can be used for analytical purposes.

Table 1.2 shows the example of categorical data for a network that consists of IP addresses and the class it belongs to. Here, the classes considered are Ipv4 and Ipv6. As it can be seen from Table 1.2, the addresses cannot be used directly for the classification. However, the format of an IP address indicates whether it belongs to the Ipv4 class or Ipv6 class. In this case, the Ipv4 address is identified as the class 0 and Ipv6 address is identified as class 1.

1.1.2.2 Numerical Data

Numerical data often refers to the data that can be measured. For example, it can be person's weight, height, blood pressure, body mass index (BMI), etc. In terms of financial analysis, it can be number of stock shares owned by a person, maximum profit obtained by selling a share. In network analysis, the examples of numerical data include download speed, upload speed, bandwidth, packet size, number of ports in a router, number of hosts in a network.

Table 1.2 Categorical data

IP Address	Class	Modified class
172.16.254.1	Ipv4	0
2001:0db8:85a3:0000:0000:8a2e:0370:7334	Ipv6	1
172.14.251.1	Ipv4	0
172.16.245.2	Ipv4	0
2001:0db6:85a3:0000:0000:8a3e:0270:7324	Ipv6	1

Figure 1.1 shows the numerical data on the bandwidth of a particular network. It represents the hosts in a network and the bandwidth for each host. For example, the host 1 is assigned 1 Mbps, host 2 is assigned 2 Mbps, and so on. In this way, the numerical data can be present in a dataset.

1.1.2.3 Ordinal Data

Ordinal data is a mix of numerical and categorical data. It consists of both numerical and categorical values, for example, the rating of movie on a scale of 1 to 5 (numerical) where 1 being the lowest and 5 being the highest (categorical). During analytics, ordinal data are often analyzed as categorical data. The main difference between the categorical data and ordinal data is the value assigned to ordinal data has a numerical meaning. For example, if the rating of a movie by a customer is 3, then it is a good movie, but if the rating is 5, then the movie is excellent as seen in Fig. 1.2.

Fig. 1.1 Numerical data representation

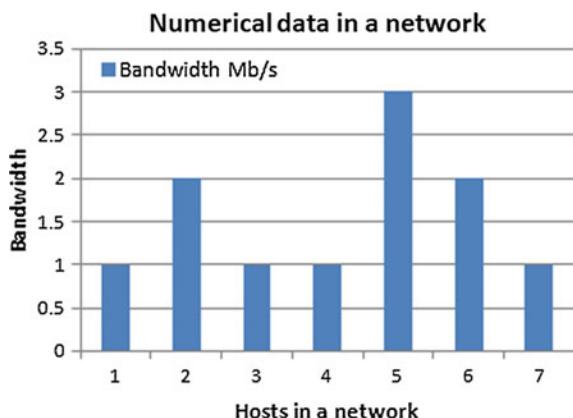
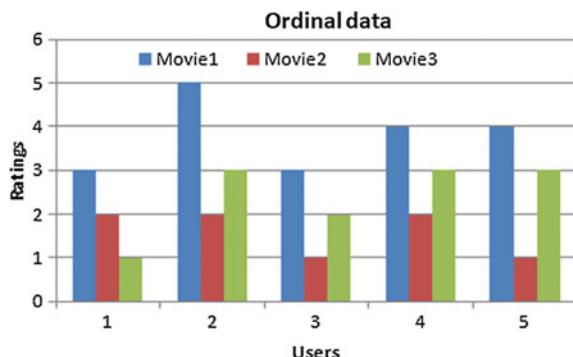


Fig. 1.2 Ordinal data representation



1.2 Data Analytical Architecture

The architecture or the frameworks needed for data analytics need to be flexible for data scientists/application developers for loading data, carrying analytics, and decision making. In any architecture of data analytics, the essential elements include data sources, tools for analysis, and reporting. For data analytics projects, different sources of data are needed for analysis. The current analytical architecture consists of typical data warehouses and reporting tools for analysis [2]. A typical analytical architecture is as shown in Fig. 1.3. The main essential components of a typical analytical architecture are,

- Data sources,
- Data warehouse,
- Dashboards,
- Users and applications.

Initially, the data sources are gathered from each source which may be in the form of numerical, ordinal, or categorical data. These data can be from different database sources such as customer, inventory, accounting, reporting, sales. Essentially, all the data are stored typically in a data warehouse that can be used by all the users and applications for reporting. The data from the data warehouse can be extracted using various ETL (extract, transform, load) tools. These ETL tools gather the data first in the original format and transform into the required format for analysis.

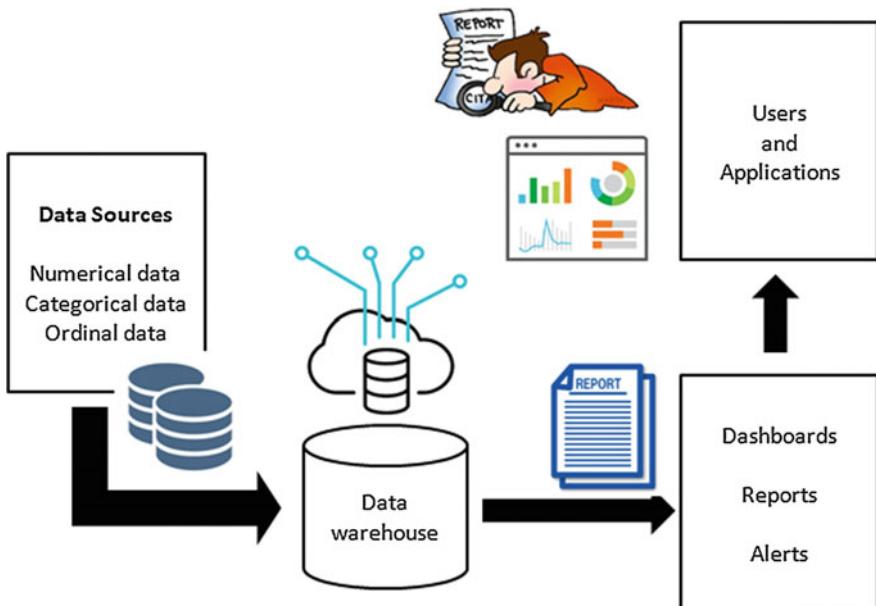


Fig. 1.3 Typical data analytical architecture

Once the data are available for analysis, typical SQL or relational database systems are used for extracting meaningful insights about the data. For example, a typical SQL query to gather the sales from month January to March can be as follows:

```
>>Select sales.data from sales where month= {January, March}
```

The reports, dashboards, and alerts at the last level of the data analytical architecture are notified to the users and applications. The applications update their respective dashboards according the analysis carried out by the previous phases of analytics. The user notifications pushed to their respective laptops/tabs/smartphone applications alert the user on the results of analysis. The users may or not take actions based on the feedback of the alert. In this way, the data analytical architecture helps in carrying out the analysis and decision-making process. However, there are typical challenges that the current analytical architecture faces and are discussed as follows:

- The data sources may not be limited to a particular number or format. So, if the data sources are coming from varied background, then the question arises of handling large volume of data. A typical data warehouse solution may not be suitable solution in such cases. The centralization of data storage with the help of a data warehouse may lead to failure and backup issues. It does not allow the data scientists/application developers to explore the data in iterative phases since it will be controlled by a centralized point.
- Some of the critical data that are essentially needed by different applications for reporting purpose and may be 24/7 operational. In such cases, the centralized point of data warehouse fails.
- One way to overcome the centralized structure can be local data warehouse solutions for each of the different sources of data accumulated. The main drawback of this method is ‘data schema’ needs to be changed for each of the new data source that comes.

The drawbacks of the current analytical architecture can be reduced by the introduction of new distributed file systems/databases as shown in Fig. 1.4. It can be seen from Fig. 1.4 that the centralized store is replaced by a distributed file systems like Hadoop, Hive, HBase for managing different sources. These types of file systems allow the typical data analytical projects to be carried out in a smoother way. The users and applications need not wait for the traditional reporting tools for reports rather can directly fetch the data from the distributed data stores and carry out analysis.

In the modern-day analytical architecture, the traditional database systems are replaced with distributed file systems for analysis. The examples of the distributed file systems are Hadoop and Spark. Hadoop is a distributed file system that helps in storage and analysis of large volume of data. Spark is an extension of Hadoop that helps in storing and analysis of large volume of data in the form of resilient distributed datasets.

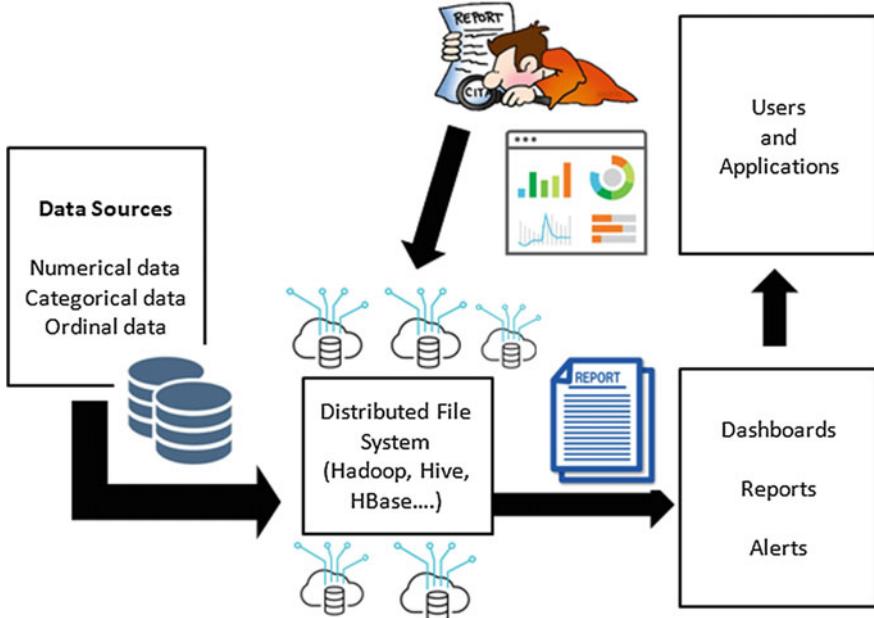


Fig. 1.4 Modern-day data analytical architecture

The main advantage of such distributed platforms for analysis is they have the capabilities like scalability, fault tolerance, and replication. The backup of data is maintained by the distributed platform Hadoop automatically and thus preventing the data loss. In this way, the current analytical architecture is modified for analysis. Similarly, on the visualization side, the traditional visual charts in sheets are replaced with new visualizations that include a dashboard for everything.

The visualization in the current analytical architecture is provided with dashboards where the entire scenario of the data sources and analytical process is available. Small minor changes can be introduced from the visualization side and feed into the file system for getting the data. However, it should not affect the analytical process. So far, the different types of data and analytics are discussed with the traditional architecture of the analytics. However, data analytics project involves different types of phases where in each of the phases a particular work is carried out. In the next section, the lifecycle of an analytics project is discussed with each phase and its components.

1.3 Lifecycle of a Data Analytics Project

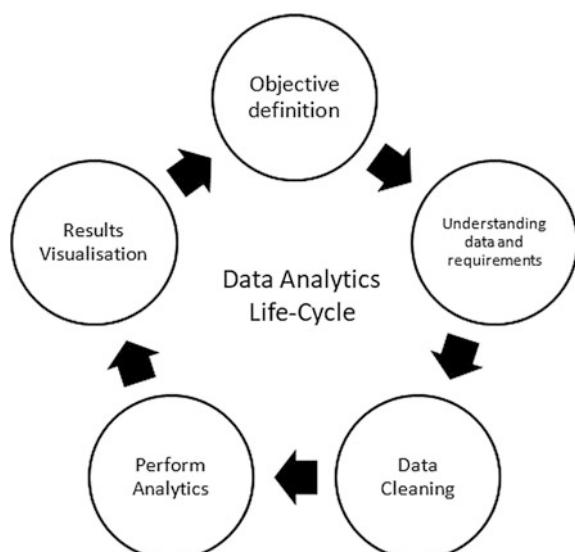
The projects that involve data analytics differ from the traditional projects of image processing and database management systems. More exploration is needed in the case of data analytical projects. A process or a lifecycle that governs the different

activities of the project is essentially needed but not to hinder the process of exploration [5]. Initially, the problems that appear large are further broken down into smaller pieces so that it can be easily addressed. If a process is governed with time and phases, then it helps in repeating the same for smaller pieces of the problem easily. Hence, the lifecycle of a data analytics project needs to be designed carefully.

Before discussing the aspects of the lifecycle of the data analytics, the common mistakes that are made are quickly doing the data collection and analysis. The scope of the work should be clearly defined with understanding the requirements and in a timely manner. In some of the scenarios, it may happen during data analysis phase that the objective defined earlier during the lifecycle may not be suitable and the project might need to be started once again or canceled. Thus, a well-defined process or lifecycle that provides flexibility of addition of methods for analysis for different parts of the project is needed. In this chapter, data analytics lifecycle is presented with five key phases, namely objective definition, understanding data and requirements, data cleaning, perform analytics, and results visualization.

The phases of the lifecycle of data analytics project are as shown in Fig. 1.5. The phases are defined for the problems related to Big data and IoT. The phases of the project are defined in an iterative manner because earlier phases in the project can be reiterated further if needed. Each of the phases of the lifecycle and its substeps are presented in the upcoming sections.

Fig. 1.5 Data analytics lifecycle



1.3.1 Objective Definition

It is the first phase of the data analytics lifecycle where the main aim/objective of the project is defined. In defining the objective of a data analytics project, the following steps are followed as shown in Fig. 1.6.

1.3.1.1 Domain Identification

Before defining the objective of the analytics project, the domain area of the problem that needs to be solved needs to be carefully studied. In the team of the project, data scientists/application developers have knowledge on quantitative and computational aspects in broader areas. For example, in the team, a member who has well-versed knowledge in applied statistical learning or life sciences adds a value in understanding the domain of the problem. The models that need to be developed in the performing analytics can be better understood with the help of this knowledge. Thus, this step helps in identifying the resources needed for the project depending on the domain and helps in maintaining the balance for the project.

1.3.1.2 Needed Resources

The resources that are needed for the project need to be estimated that include tools, technology, staff, and systems. The estimation of resources is important for the later operationalize phase of analytics. It helps in identifying the gaps between the tools,

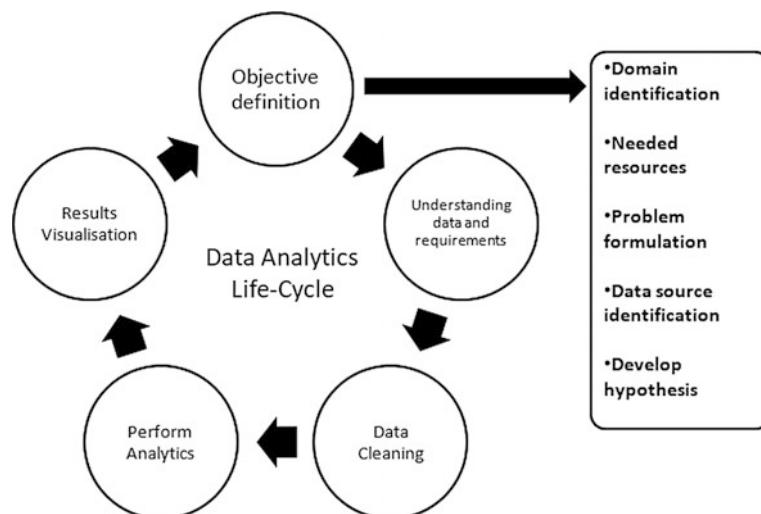


Fig. 1.6 Objective definition in data analytics lifecycle

technology, and skills of the people involved in the analytics project. For example, if there is a project estimated for a longer period of time, then the resources for the project need to be equipped with the skills that are later needed in the future. The resource estimation helps to answer some of the questions like what type of advanced skills are needed? and will it be cultivated with the existing expertise in the team? and thus facilitate long run of the projects.

In addition to the computer and technological skills, the data available for analysis also need to be looked upon. Data act as a key resource in the analytics lifecycle. The data available may be too less for analysis, and thus, the team might need to look for other sources of data or transform the existing data itself. Data inventory is also one of the key resources that need to be looked upon for analysis.

Once taking into account of the resources needed such as tools, technology, people, and data, further negotiations need to be made. Project managers and other key stakeholders negotiate on the resources needed and available for the analytics lifecycle. In this way, the resources that are needed for the analytics lifecycle need to be estimated.

1.3.1.3 Problem Formulation

The key step in the objective definition step of the analytics lifecycle step is the problem formulation. In this step, the problem that needs to be solved has to be clearly framed for solving. It is also called as framing the problem statement. The problem statement is framed and shared among key stakeholders. The stakeholders might give different opinions about the problem statement as seen from their perspectives. The solutions can also vary with different types of views about the problem statement. Thus, the team needs to articulate the situation of solving the problem and assess the challenges of solving the problem.

The main aim of framing the problem statement is to come up with the objectives of the problem and identify the objectives that need to be met. The success criteria can also be estimated as a part of this process to see how many objectives can be met. Based on the success criteria, the team will know the outcome of the project and is it good enough to meet the needs. The best practice is to share the success criteria among the key stakeholders like sponsors in projecting the expectations that can align with the clients.

Along with the success criteria, it is also a good practice to assess the failure criteria of the projects. The best-case scenario of aligning the success criteria assumes that the project is carried out in a planned approach and the team achieves goals and objectives of the project as planned. However, there are certain difficult tasks and outcomes that a project faces that are not planned. Thus, the failure criteria for the analytics lifecycle also need to be planned. It helps the team in understanding when to stop and see if the limited results can be obtained. In this way, establishing both success criteria and failure criteria will help the team to align to the sponsorship goals as well.

1.3.1.4 Data Source Identification

Once the problem formulation is done with clear objectives defined, the necessary data sources that are needed for analytics should be identified. The team should try to collect aggregated data for analysis depending on the volume and variety of the data. The necessary steps that should be carried out as a part of data source identification process are listed as follows:

- **Identify data sources:** The data sources that are needed for analytics are listed first to see the available data sources. The data sources can be categorized as aggregated data sources, offline sources, and any online-based sources.
- **Check raw data:** If some raw data are available like image and video and if any analytics is based on that, it needs to be curated and transformed into required format. Such data need to be evaluated first and then see if it is appropriate for analytics.

1.3.1.5 Develop Hypothesis

A hypothesis is a formal statement of the problem for analytics. Formulating hypothesis gives an idea of the potential data sources that can be used for analytics. For example, the hypothesis can be '*Gaining insights from customer feedback and improving the product marketing.*' In this example, the hypothesis clearly states that customer feedback analytics is carried out for improving the product marketing. The data source is the customer feedback in the form of either ratings or reviews.

A set of hypothesis can be generated based on the initial hypothesis. The team can derive multiple hypotheses using the initial one and have a better idea of the problem. For example if the initial hypothesis is '*Improving user experience in company website,*' a set of other hypothesis that can be derived from it as follows:

Improving user experience for payment in the company website.

Improving user experience for adding products to cart in the company website.

Hypothesis can be developed by different stakeholders of the team from their perspectives, and the final hypothesis can be developed by negotiations. Formulation of hypothesis in a typical data analytics project is usually carried out by collecting multiple views. These views help in expanding the initial hypothesis and deriving the final set of hypothesis. The final set of hypothesis can be the objectives of the project, and easily the deliverables can be formulated. In this way, the formulation of hypothesis can be carried out.

1.3.2 Understanding Data and Requirements

The objective of the problem formulated in the first phase of analytics needs to be solved in the next phases by first identifying the data features and requirements to

achieve the objective. In this phase of analytics, the attributes or the features of the data are extracted from the dataset and analyzed. Once the features are known, the necessary requirements for using the features of the data are estimated in this phase. The important substeps of this phase are *identifying data features* and *identifying data structures* required as shown in Fig. 1.7. These substeps are discussed with examples as below.

1.3.2.1 Identify Data Features

A critical step in the analytics lifecycle is to understand the data and get familiar with the domain of the analytics to be performed. For this purpose, the different features in the data need to be analyzed clearly. It provides the context of the data to be analyzed with expected outcomes and final results. For example, consider a scenario where the retail analytics need to be carried out for an online shopping site. The data features that are available in this context are *products shipped*, *product survey*, *social media sentiment*. For these features, a dataset inventory needs to be prepared that gives the idea of accessibility of the data. Table 1.3 summarizes the data inventory for the scenario considered. In Table 1.3, it can be observed that social media sentiment data need to be collected from the outside sources and other data sources are marked for the respective categories. In this way, the data inventory set can be created for data sources.

One of the other important tasks of identifying the data features is to identify whether it is categorical or ordinal or nominal. These types of data are discussed in the earlier sections of the book. For example, in the scenario of retail analytics

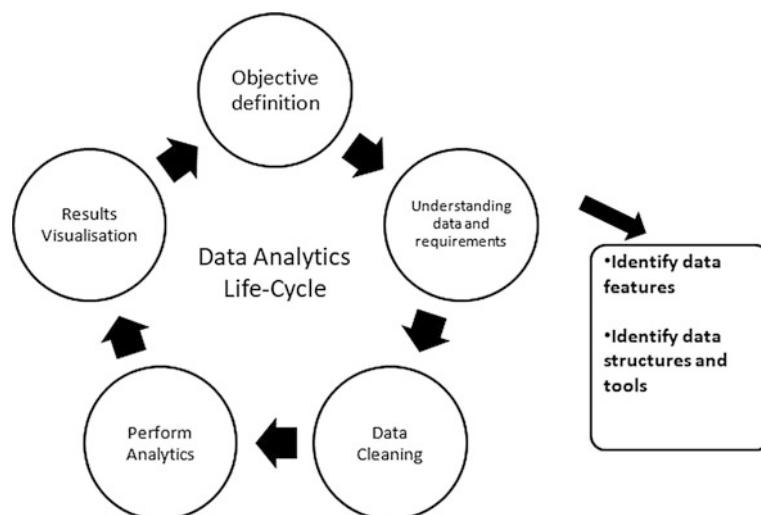


Fig. 1.7 Understanding data and requirements in data analytics lifecycle

Table 1.3 Data inventory set for retail analytics

Dataset	Available and accessible	Available but not accessible	Obtain from outside sources
Products shipped	Yes		
Product survey		Yes	
Social media sentiment			Yes

considered, *products shipped* feature is numerical data that give the number of products shipped (25, 43). Another feature, *product survey*, is categorical in nature that may say *satisfied*, *not satisfied*. In this way, the different data features in the dataset can be identified. This phase of analytics can be more understood in machine learning techniques that are covered in part 2 of this book.

1.3.2.2 Identify Data Structures and Tools

The data features identified in the previous step for understanding the context of analytics to be performed have to be used with necessary data structures and tools. Here, the data structures can be a *csv file*, *text file*, *matrix*, *json file*, etc. These types of file structures or data structures are normally used for analytics. For example, in the case of retail analytics online, the data feature *products shipped* can be from various departments in a csv file. Similarly, the *product survey* can be in the form of text file with questions and answers from the customers. The data feature, *social media sentiment*, that needs to be collected from outside data sources can be in json format. In this way, the different data features, the data structures in which the data are present, need to be identified.

However, once the structure in which the data features are present may not be enough for analytics. The format needs to be converted to other formats and is taken care by data conditioning phase in the later phases of analytics. The tools that are needed for fetching the data need to be identified. For example, Python programming and its modules can be used in reading csv data, text data, and json data. The *pandas* module in Python is used for reading the csv files and other formats. The various tools that help in understanding of data that are present in various formats are:

- **Python:** It is one of the popular scripting language that helps in reading various data formats such as csv, text, json using the modules like numpy, pandas, and matplotlib. The advantage of Python language is it is easy to program and build various analytical models [6].
- **R:** It is a powerful statistical tool that helps in reading various formats of files and performs statistical functions such as correlation, mean, variance, covariance.

- **Apache Hadoop:** It is a distributed file system that helps in storing large volume of information. Most of the information that is in raw format can be saved in Hadoop and then use other tools for transforming the data. The data are stored in a distributed manner with NameNode and datanodes. The main advantage of Hadoop solution is it can be configured with commodity hardware [7].
- **Apache Hive:** It is a data warehouse solution on top of Hadoop that helps in storing the data in tabular format for analysis. The files that are stored in Hadoop can be viewed in Hive for simple analytics. However, advanced analytics can also be performed with the help of Hive functions.
- **Apache Flume:** It is a workflow platform that helps in retrieving data from multiple platforms into Hadoop using various workflow connections.

1.3.3 Data Cleaning

The data sources and the data structures identified in the initial phases of analytics are used to gather the data from different sources. However, all of these data might exist in different formats and need to be converted to a suitable form for analytics. For example, in carrying out an image analytics project, the image data in pixels cannot be directly used for analytics. It may need to be converted into a suitable format for analysis. In this phase of analytics, the data are subject to conditions for analysis. The substeps of this phase are discussed as follows as shown in Fig. 1.8.

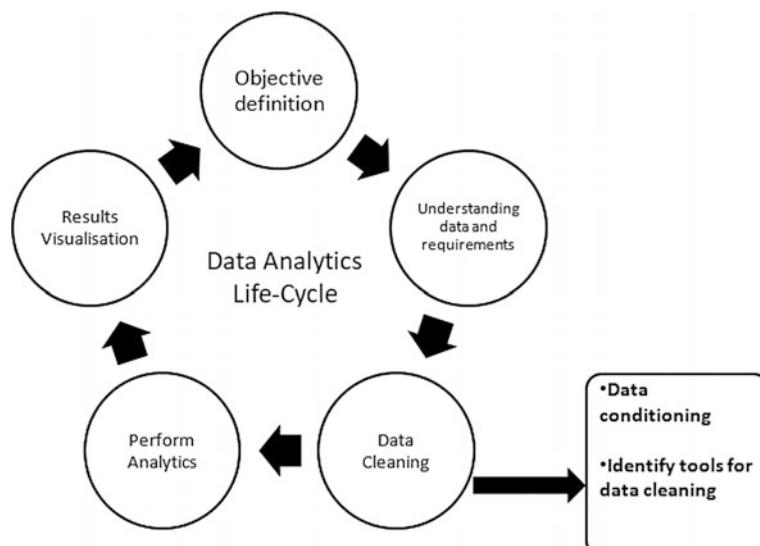


Fig. 1.8 Data cleaning in data analytics lifecycle

1.3.3.1 Data Conditioning

Data conditioning refers to the process of normalizing and transformation of data required for analytics. It is one of the critical steps of analytical lifecycle where the data are subjected to various phases of normalization and cleaning. It facilitates the developers for further analysis phases of analytics lifecycle. Many operations on the dataset like normalization of categorical data and nominal scale analysis are done as a preprocessing step in data conditioning phase of analytics (Fig. 1.9).

It is a cautious phase in the analytics lifecycle as most of the stakeholders in the project are dependent on the format of the data preprocessed in data conditioning phase. A consent needs to be taken by the important stakeholders of the team for data conditioning. For example, consider the scenario of *building recommendation system by analyzing the user ratings and feedback*. In this case, the data might contain good, bad, and neutral ratings. The data from all the three ratings need to be considered. However, some of the users might have purchased the product and not given the rating. In this case, the team must decide what should be valued in this case? One of the solutions can be ignoring such tuples of data. The problem that occurs in such cases is if a tuple is ignored for a user who has purchased few products but not given rating for a particular product, then the recommendations might not be proper. Thus, some of the questions that can be formulated for data conditioning are listed as follows:

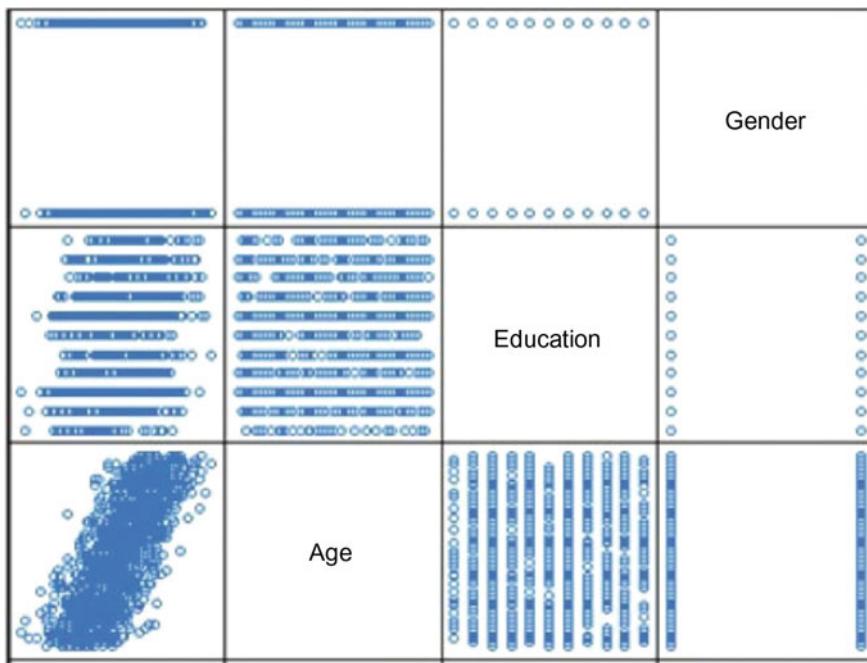


Fig. 1.9 Sample scatter plot for data exploration

- Which are the data sources and target variables in it?
- What is the consistency of data? The degree to which the consistent and inconsistent values are present in the data.
- What is the type of the data columns? The values in the column must match the expected type of column. For example, if student grades are assigned with values 1–10 or A–E, can there be a mixture of values or only one?
- Are there any system errors in generated data? In network analytics, most of the data are system generated. In such cases, there might be errors in data related to IP addresses, hosts, and router IDs. The data need to be reviewed to see if they match the standard formats as prescribed.

1.3.3.2 Tools for Data Cleaning

The data conditioning can be done with the help of some tools as listed below. These tools provide a basis for cleaning the data as required for analytics.

Hadoop

Hadoop is one of the basic tools that can be used for data cleaning. The examples where Hadoop is used as a tool are parsing of web traffic, GPS data, genomic analysis and aggregating all the unstructured data. Since, it supports any format of data to be stored in a distributed manner it can be used as one of the tools for data cleaning. However, MapReduce programming need to be used for retrieving and processing of data in Hadoop. The programming knowledge of MapReduce is essential for data cleaning using Hadoop.

Alpine Miner

It is one of the tools that can be used for data cleaning in analytics lifecycle and provides GUI interface [8]. The GUI allows the developer to create different workflows to perform analytical events like top 100 customers, top 100 posts etc. It is based on SQL and thus easily one can use tool for data conditioning in the analytics lifecycle.

1.3.4 Perform Analytics

The fourth phase of analytics lifecycle involves performing analytics. Once the data are gathered from different sources and data requirements are understood subjected to the treatment for analysis, analytics process can be carried out. In this phase, the different substeps involved are listed as follows as shown in Fig. 1.10.

1.3.4.1 Variable Selection Using Data Exploration

The data that are explored in the second phase of analytics for understanding the data and requirements involve checking only quality of data and its sources. The

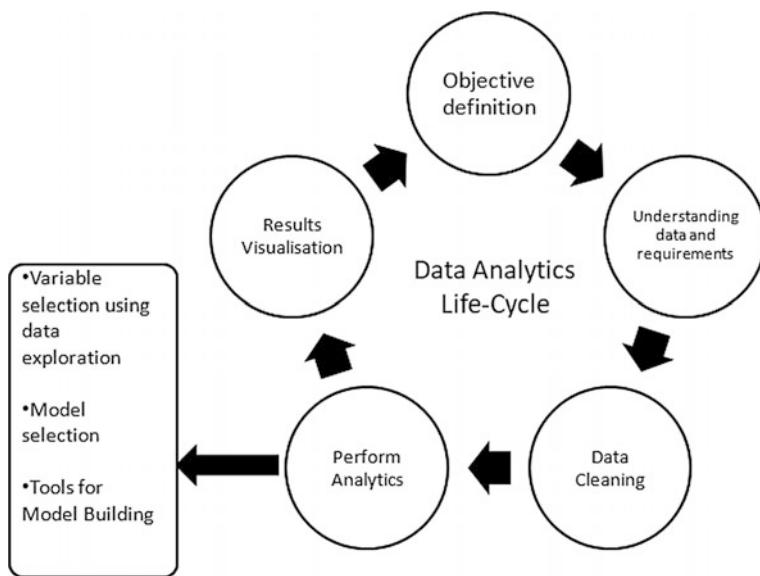


Fig. 1.10 Perform analytics in data analytics lifecycle

objective of the exploration is extended further for analysis by identifying the variables for model building. The dataset contains many variables, and only some of the variables are needed for analytics. In this context, the variables that are needed for analysis and the correlation among them need to be identified.

One of the ways to do variable selection is to use small visualizations for the dataset and see the variables that can be selected. For example, if we are determined to predict the forecasting values within a dataset, a scatter plot would be helpful for identifying the variables. A scatter plot is as shown in the figure for a dataset. The dataset consists of income, age, gender, and education as the variables. If the objective is to forecast the income depending on other variables like age, education, the scatter plot can be used. It can be observed from Fig. 1.9 that scatter plot is that there is a high correlation between the education and the income but not with gender. Thus, in this way with visualizations, the variables needed for analytics and their correlation can be identified.

The different stakeholders in the team can also have a hunch of the variables to be used in the dataset for analytics. The hypothesis formulated in the phase 1 might have implications on the variable selection as well. However, they should not be dependent on each other. Although the stakeholders in the team have a good knowledge on the domain of the analytics, there can be mere consequences on the hypothesis formulated. The hypothesis may need to be accepted or rejected as one the basis of the data variables selected. There can be cases where the correlation computed might be wrong and thus may lead to rejection of hypothesis. The cautious of choosing the variables will be through exploration and visualizations.

The key to this step for performing analytics is to choose the correct variable for prediction rather than exploring each and individual variable for analysis. It may be an iterative phase where the essential variables required for analysis need to be identified. For example, if the model is regression analysis, then the correct-independent variable needs to be identified for the other dependent variables. In this way, the variables needed for analysis have to be explored so that models selected in the next step need not be changed in the next step.

1.3.4.2 Model Selection

Depending on the goal of the project, a model is selected from a list of techniques for carrying out analytics. The main goal of this step is to choose a model or an analytical technique to achieve the hypothesis formulated in the phase one of the analytics lifecycle. A set of rules and conditions are designed for the model to analyze the data in real time and act according to it. Generally, in machine learning, these sets of rules are combined into models such as classification, clustering, regression. Given a list of models for analytics, the team can select one or two viable models for carrying out analytics to a given problem. The details of model building like regression, clustering, SVM are discussed in detail in part 2 of this book.

Model selection is not limited to only machine learning and data mining models. It can also consider additionally other types of data like unstructured and semi-structured data. For example, MapReduce models can be used for unstructured data analytics. A documentation of choosing and constructing preliminary models in the model selection phase helps to achieve the objectives of the project. Once the team has selected the model, it can be built using various options.

In the phase of **model building**, the dataset is split into training and testing dataset first. This helps in developing an analytical model for the training set first and testing the same model on the testing data taken. The training and the test datasets should be sufficiently robust for building the analytical models. The model built is fit into the training data, and the same model is evaluated for the accuracy on the test data. The different modeling techniques used for model building may be complex, but if the initial phases of the analytics lifecycle are completed with no ambiguities, it will take short time for building the model.

The model built has to meet the objectives in phase 1 of the analytics lifecycle. Some of the questions are listed below for which the answers need to be drawn for meeting the objectives of the project.

- Is the model accurate and valid for the given data?
- Does the output behavior of the model appropriate for objectives designed earlier?
- Do the parameter values selected for the model appropriate for the context of the project considered?
- Does the model sufficient for achieving the overall goal of the project?

- Are the data inputs considered enough?
- Do the data inputs have to be transformed once again to improve the model?
- Does one or models need to be constructed once again to see if they make sense about the context of the project?
- Does the model run according to the infrastructure requirements?

Once the model selection and model building are done for the project considered, the analytics lifecycle further moves to visualization phase. The phase is discussed in the next section.

1.3.4.3 Tools for Model Building

For model building, there are various tools. Some of the open-source tools are listed as below:

- **Python:** It supports model building methods such as classification, regression, clustering with the modules like scikit-learn, pandas, numpy, and matplotlib.
- **Weka:** It is a data analytical package available in Java environment and supports various analytical models.
- **R:** It is statistical software, with the support of various data analytical models as well.
- **MADlib:** It is SQL-based open-source software that can be used for building analytical models as well.

The tools in the model building phase and in the phase 2 for understanding the data requirements overlap with each other as there has to be consistency among the data gathered and model selection. Since the model will be using the same data gathered in the phase 2 of the analytics lifecycle, the same tool can also be used in model selection and building.

1.3.5 Results Visualization

In the analytics lifecycle, the last phase is results visualization and communication. This is the phase where the actual results of success criteria and the failure criteria defined in the phase 1 of analytics can be reviewed. The models that are built in the previous phase of analytics lifecycle need to be visualized for better clarity over the analysis phase. The different substeps of the results visualization are listed as follows.

1.3.5.1 Identify the Type of Chart

The model used during the initial phase of the analytics lifecycle can be visualized with different types of charts. The generic representation of these charts are

relationship charts, composition charts, comparison charts, and distribution charts. These charts are selected based on the model used for analytics lifecycle. The different types of chart are discussed in detail in part 4 of the book.

The results visualization should match the model characteristics and its estimated results. For example, if regression is the model used for analytics, the model should fit the data points as shown in Fig. 1.11. In this way, different types of charts can be used for different models used for analytics lifecycle. However, instead of charts, tables can also be used to communicate the results of the analytical results. For example, in the case of classification a confusion matrix in tabular format can be used as a means for analysis. It is as shown in Table 1.4 where 25 data points are classified exactly as ‘cat’ and 34 of them are not classified as ‘cat.’ Hence, for each model, the results visualization can be accomplished with appropriate charts and tables.

1.3.5.2 Aggregation of Charts

If there are multiple models considered for the analytics lifecycle, then the visualizations can be different for each model. In such cases, the charts obtained for these models need to be aggregated as a single chart to get an overall view of the results. For example, if there is a scenario of *forecasting the values of clothing in an online retail*, then both regression and clustering models are used for analytics.

In such cases, the regression model fit and the different classes of clustering need to be aggregated for visualization. The team of the analytics project might be split into offshore and onshore. So, in order to have a common understanding among the key stakeholders in the analytics project, the charts have to be aggregated into a single one. Hence, in this way, the charts can be aggregated to interpret the results of the graphs in the chart.

Fig. 1.11 Regression chart

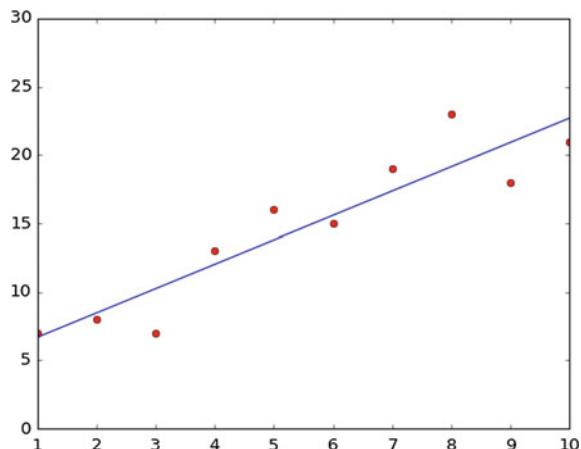


Table 1.4 Confusion matrix visualization

	True	False
Cat	25	34
Dog	23	45

1.3.5.3 Report Writing

The final step of analytics project is report writing where the collected results are matched with the objectives defined for the project. In report writing, right from the phase one to the last phase all the necessary results, graphs, and tables are included. The report should be written in such a way that the models used in the analytics project are well understood and can be used for others as well.

In the first chapters of the report, the objectives and hypothesis of the project can be clearly stated. It should include the necessary motives for carrying out analytics in a specific domain. The different stakeholders considered for the analytics project have to be listed down with their roles. The chapter can end with a suitable hypothesis that needs to be modeled for getting insights.

The later chapters in the report can include the data sources, infrastructure requirements, variables included for data analysis, model selection, model building procedure, and the visualization of results. In the report, necessary figures and tables can be included in the sections such as model building and visualization for proper communication of results. The best practice is to document the findings of results in each chapter for avoiding the ambiguities encountered in the report.

1.3.6 Put the Model into Operation

The report prepared can be reviewed by a team of experts to evaluate the correctness of the model and the objectives of the project. Depending on the review by the experts, the model can then be put into operationalize phase for the continuous analytics. In the process of reviewing, the team should be in a situation to reiterate the phases of analytics for minor changes. (Fig. 1.12)

1.4 Big Data and Ecosystem

With the rise of Internet and computing, the data volume increased day-by-day and has reached from few gigabytes to petabytes of data. Similarly, the formats of the data generated typically from the traditional relational database systems (RDBMS) have moved to new social media formats (image, video, etc.). The term ‘Big data’ is related to all of these. Big data refers to three main characteristics, namely volume, variety, and velocity [2]. Here, volume refers to the amount of the data that are generated for a period of time. The amount of the data can be 1 GB, 1 TB, 1 PB, and

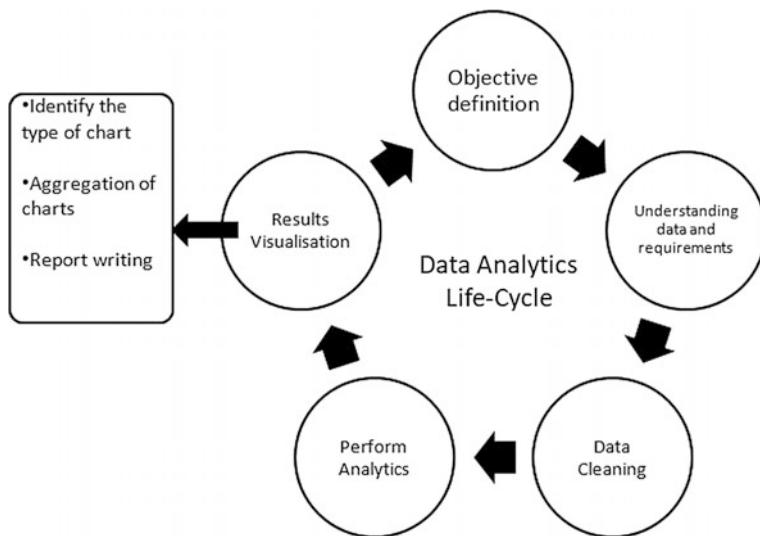


Fig. 1.12 Results visualization in data analytics lifecycle

so on. The volume depends on the variety of the data generated. Here, the variety refers to the different structures of data such as text, image, video, audio, sensor data, computer data. These types of data vary in different formats. For example, an image can be saved in jpeg, png, tiff, etc. The velocity of the data refers to the rate at which the data are being generated for a certain period of time. The period of time can be 1 s, 1 min, 1 h, and so on. For example, there can be 1 lakh images shared in 1 h, 2 million images shared in 1 day, and so on. With such different types of data generated, technologies need to be advanced in gathering such data to gain insights. A stand-alone technology cannot help in these situations and rather requires an ecosystem for supporting wide variety of data. The ecosystem of Big data should be able to accommodate data stores, learning methods, and smart applications. In this section, the ecosystem of Big data and its components are discussed.

The ecosystem of Big data comprises of elements like data sources, data acquisition systems, data processing, and smart applications as shown in Fig. 1.13. These components may differ on the domain of the analytics to be performed [5]. Figure 1.13 represents the abstract view of the ecosystem with subcomponents in it. The subcomponents are listed as follows:

Data sources: The different types of sources that are present in the Big data ecosystem are sensors, location trackers, transaction logs, social media, email, and messaging. These are the essential sources that mark the analytics lifecycle phase. The formats of the data produced from each sources differ and have to be gathered by transformation. For example, the sensors and location trackers might produce the data in structured format, whereas the format of the data generated in social media is unstructured. Thus, different sources of data that generate different formats of the data need to be aggregated in a form that can be stored in data acquisition systems.

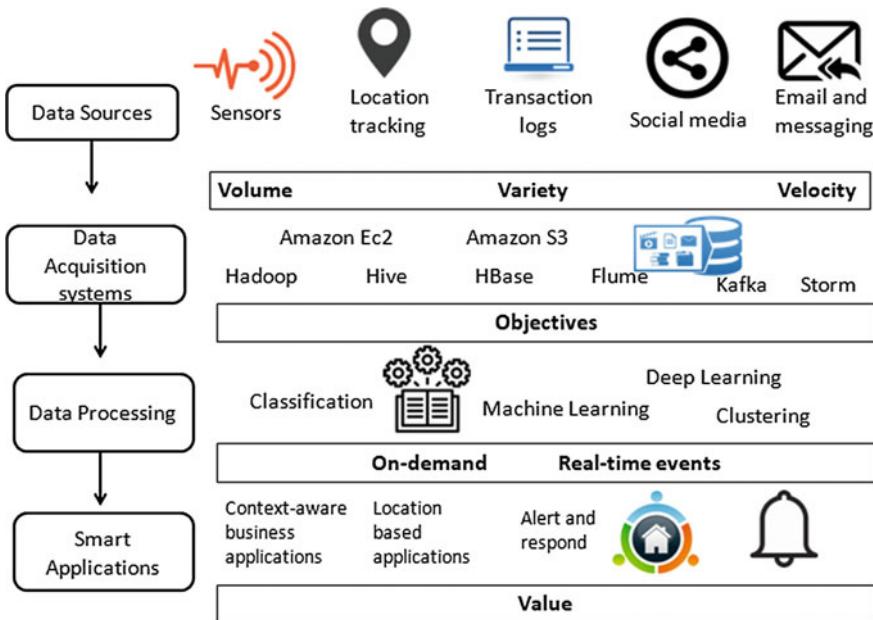


Fig. 1.13 Big data ecosystem

Data acquisition systems: The data generated from the sources can be acquired through systems like Hadoop, Hive, HBase, Amazon Ec2, Amazon S3, Kafka, Storm. The example of the systems represented here belongs to the Hadoop ecosystem. Though there are multiple platforms that are available for data acquisition, these are in majority used for analytical applications. The main advantage of Hadoop-based systems is they are based on distributed computing. These systems support to view the data in different formats. For example, a text file stored in Hadoop can be viewed as a table in Hive. In this way, data acquisition systems are used for storing the data gathered from different sources.

Data processing: The data stored in the systems can be analyzed with techniques like machine learning, deep learning, classification, clustering, regression, and others. The data acquisition systems must support data analysis in this context further. The systems considered in the previous step support data analysis through various programming models. For example, Hadoop and Amazon Ec2 provide MapReduce programming model for analysis. Hive provides SQL support for analysis. Kafka and Storm supports real-time analysis of data that are gathered from different sources through spouts and bolts. However, machine learning techniques like classification, regression, clustering are supported by Apache Spark using MLlib module.

Smart Applications: The ecosystem of Big data with data acquisition and data analysis leads to smart applications. The examples of smart applications include context-aware applications, alerts and notifications, genomic analysis, climate

analysis, location-based applications, retail analytics, fitness applications. These applications are based on the models created in the data analysis phase. Thus, smart applications lie at the last line of the ecosystem. As the ecosystem develops with newer technologies and data acquisition systems, smart applications can wear out or get updated with new features.

The Big data ecosystem can be correlated with analytics lifecycle in the following way. The data sources can be mapped to phase 1, i.e., identifying the objectives. The data acquisition systems can be mapped to phase 2, i.e., understanding requirements. The data processing can be mapped to phase 3 model building, and smart applications can be mapped to phase 4, perform analytics. The phase 5 of the analytics lifecycle, i.e., visualization, is not listed in the Big data ecosystem as it depends on the domain considered for analytics. In this way, the Big data ecosystem mapping can be done with the analytics lifecycle.

1.5 Exercises

1. What are the different types of data sources for structured, semi-structured, and unstructured data?
2. Describe the key users in a data analytics project.
3. What are the objectives of a typical data analytics project? Explain with scenarios.
4. What is Hypothesis testing? Explain the different types of hypothesis that can be drawn for a data analytics project for the following scenario. ‘A company wants to study the positives and negatives of a product.’
5. Describe the key considerations in selecting the variables for data analytics.
6. Discuss the different types of ordinal data in a typical computer network environment.
7. Is text analytics a descriptive analytics? Infer with an example.

References

1. Miner, G., Elder, J., IV, & Hill, T. (2012). *Practical text mining and statistical analysis for non-structured text data applications*. Cambridge: Academic Press.
2. Tsai, C. W., Lai, C. F., Chao, H. C., & Vasilakos, A. V. (2015). Big data analytics: A survey. *Journal of Big Data*, 2(1), 21.
3. Kambatla, K., Kollias, G., Kumar, V., & Grama, A. (2014). Trends in big data analytics. *Journal of Parallel and Distributed Computing*, 74(7), 2561–2573.
4. Analytics, D. N. T. D., & Mining, L. B. I. P. (2015). *Machine learning and knowledge discovery in databases*. Lecture Notes in Computer Science, 9286.

5. Demchenko, Y., Ngo, C., & Membrey, P. (2013). Architecture framework and components for the big data ecosystem. *Journal of System and Network Engineering*, 1–31.
6. Python, <https://www.python.org/>.
7. Hadoop, <http://hadoop.apache.org/>.
8. Alpine Miner, <http://datascienceseries.com/partners/partners/alpine-data-labs>.

Chapter 2

Hadoop



2.1 Introduction

Before the ‘Big data’ came into picture, the options for processing huge volumes of data were less. Some of the databases were used for such analysis purposes. These options were expensive with hardware setups, and expertise was needed for analysis. A customized nature of systems was needed for specific applications. It was a great difficulty in handling the large data with such systems. On the other hand, the system in some cases was not a good fit to the problem considered for solving. Most of the companies started looking for solutions that are smart to handle such large data.

As the decades of computing grew, data generation also gradually and so does the need to process such data also grew. Since the key for business requirements was the data, it was a high time for the major changes in the architectural designs of data processing. The alternative options to the high-end hardware systems were looked out. Mainframe computing was most used for data processing of large volumes. The first change was to cut down the cost of using these high-end systems for data processing. The use of commodity hardware could bring this change and lower the expenses for data analysis.

Google started to think in this line as it deals with huge volumes of data. In 2003 and 2004, it came up with new project Google File System (GFS) and MapReduce. GFS provided way for multiple processing of data in a large-scale and efficient manner. The main aim of the GFS was to provide a platform that is not suited for a single domain but in general to all the data processing problems. Google used the commodity hardware for processing using the principle that hardware is prone to failures and software needs to handle it. In this way, the change in the architectural design of the data processing systems was initiated.

In the mean time, an open-source project Nutch based on Web crawler was carried out by Doug cutting [1]. The GFS and MapReduce elements were introduced in Nutch and were renamed as Hadoop under the Apache software

foundation. The implementations of the GFS and MapReduce elements were introduced in Nutch and renamed it as Hadoop under the Apache software foundation [2]. In this way, the major architectural design was initiated by Google and Doug cutting for large-scale data processing. In this chapter, the common building blocks of Hadoop, its architecture, MapReduce programming model, examples on MapReduce, and case studies on MapReduce are presented. The major highlights of the Hadoop are listed as follows:

- Hadoop is designed to run on low-commodity servers and clusters.
- It supports scalability by allowing the addition of more servers and thus helping in the scale-up architecture.
- It helps in identifying the work-around failures.
- It manages the cluster with features like load-balancing and fault tolerance.

2.2 Hadoop Architecture

Hadoop is an open-source platform for processing large datasets. It is based on the distributed computing platform. The essential components of Hadoop are *MapReduce* and Hadoop distributed file system (HDFS) [3]. The basic architecture of Hadoop is as shown in Fig. 2.1. In this section, the architecture of the Hadoop is discussed.

The architecture shown in Fig. 2.1 is a basic level of architecture of Hadoop. However, the high-level of architecture of Hadoop is discussed in the upcoming sections of this chapter. The architecture is based on the traditional master-slave architecture with two essential components *NameNode* and *data node*. The overall architecture is called as Hadoop distributed file system (HDFS) because it allows data nodes to be configured in a distributed manner.

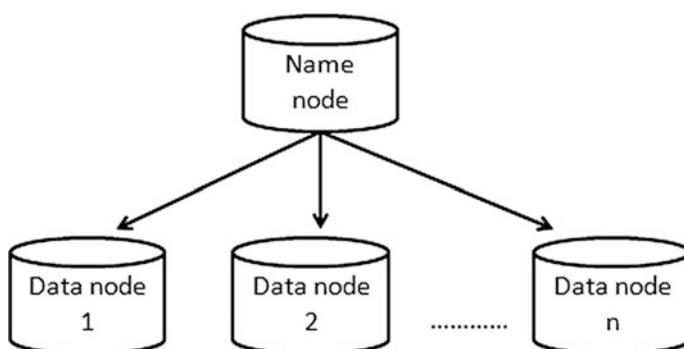


Fig. 2.1 Hadoop architecture

2.2.1 NameNode

The NameNode in Hadoop acts as the master that contains the metadata of the files stored in HDFS. It is the main mode in Hadoop that gives the overall information of the cluster [4]. It maintains the data such as configuration of the cluster, number of data nodes, number of MapReduce jobs being run, MapReduce jobs completed, memory in the cluster. It is a single point of contact for knowing the cluster configuration. Once Hadoop is installed, `localhost:8080` will act as the portal to know the information of the cluster using NameNode. A snapshot of the NameNode example is as shown in Fig. 2.2.

It can be observed from Fig. 2.2 that the configured capacity for the Hadoop file system is around 14 GB along with other memory configurations. In this way, the cluster configuration can be seen within a NameNode.

2.2.2 Data Node

The data are stored in the data nodes in Hadoop. It acts as slaves for the master NameNode. The NameNode assigns the computation to be done by the slaves, i.e., data nodes. The computation of MapReduce jobs takes place in the data nodes. The data nodes are divided into regions further for data processing. For example, if there are three data nodes, there can be three regions or more than that for the data nodes. These regions are used for replication of the data and fault tolerance.

The main function of the data nodes in Hadoop is to perform the map or reduce tasks assigned by the NameNode and maintain fault-tolerant architecture. The fault tolerance is provided by Hadoop through data node communication. A heartbeat message is sent from the data nodes to the NameNode for maintaining the ecosystem of the Hadoop.

Cluster Summary

```

Security is OFF
75 files and directories, 26 blocks = 101 total.
Heap Memory used 8.77 MB is 43% of Committed Heap Memory 20.03 MB. Max Heap Memory is 193.38 MB.
Non Heap Memory used 22.63 MB is 99% of Committed Non Heap Memory 22.69 MB. Max Non Heap Memory is 96 MB.
Configured Capacity : 14.08 GB
DFS Used : 204 MB
Non DFS Used : 7.16 GB
DFS Remaining : 6.72 GB
DFS Used% : 1.41 %
DFS Remaining% : 47.72 %
Block Pool Used : 204 MB
Block Pool Used% : 1.41 %
DataNodes usages : Min % Median % Max % stdev %
                   1.41 % 1.41 % 1.41 % 0 %
Live Nodes : 1 (Decommissioned: 0)
Dead Nodes : 0 (Decommissioned: 0)
Decommissioning Nodes : 0

```

Fig. 2.2 Hadoop cluster with NameNode

2.3 Memory and Files in Hadoop

The files are stored in a distributed manner in Hadoop. The NameNode in Hadoop takes care of distributing the files in Hadoop to the data nodes. When the client submits the file in Hadoop to be stored, this file is treated as *blocks* by the NameNode in Hadoop [5]. These blocks are stored in data nodes, where each size of the block is 64 MB. For example, if there is a large file of size say 1 GB, the file is divided into as many blocks by the NameNode where the size of each block is 64 MB. The division of the file into blocks is automatically taken care by the NameNode of Hadoop, and the user need not be focused on this. The blocks stored in the data nodes are as shown in Fig. 2.3. It can be observed in Fig. 2.3 that the blocks are stored in the data nodes of Hadoop and size of each block is 64 MB.

2.4 Replication in Hadoop

Hadoop provides fault-tolerant architecture by replication of files in the distributed file system. The default replication factor is 3 in Hadoop; i.e., the number of replicas of the files in Hadoop is 3 [6]. The blocks in the data nodes are replicated by a factor of 3 where one replica is stored in the same data node where the original file resides. The second replica is stored in another node within the same cluster. The third replica is stored in another node within the same cluster or in another cluster if it is available. The replication of the blocks in Hadoop is as shown in Fig. 2.4. It can be seen that the data blocks 1, 2, 3 are replicated in other nodes in the cluster.

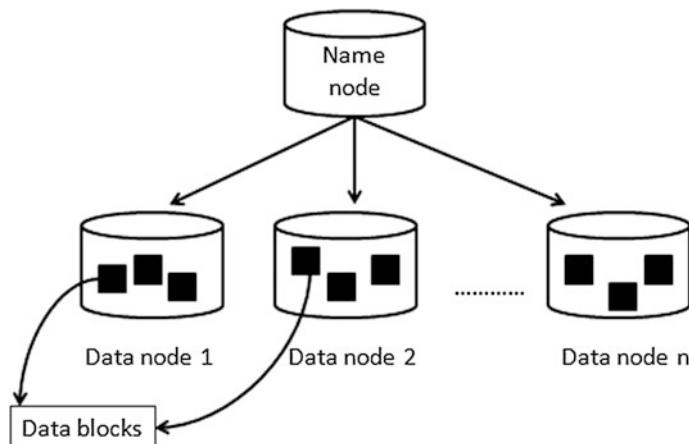


Fig. 2.3 Blocks in data nodes of Hadoop

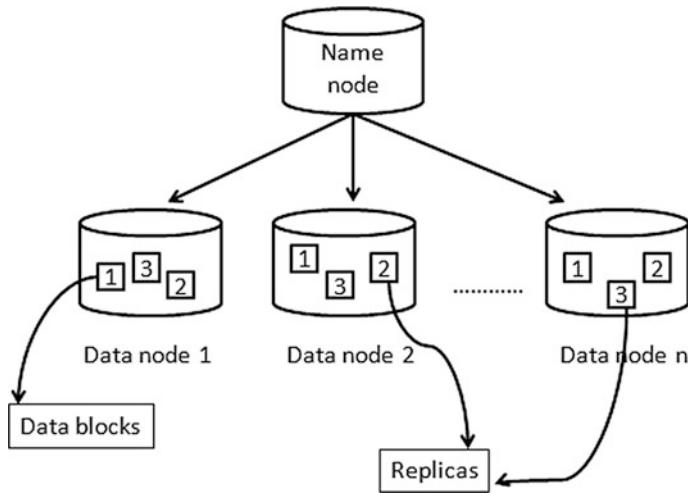


Fig. 2.4 Replication in Hadoop

2.5 MapReduce

MapReduce is a programming model/framework that allows to execute tasks in parallel with Hadoop. The files stored in the Hadoop can be analyzed with MapReduce tasks [7]. It consists of two tasks, namely MapReduce. The basic framework of the MapReduce programming model is as shown in Fig. 2.5 [8].

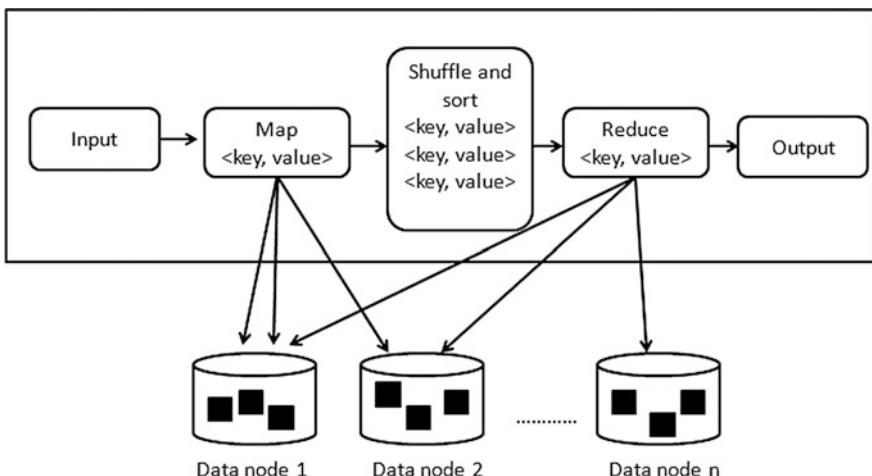


Fig. 2.5 MapReduce framework

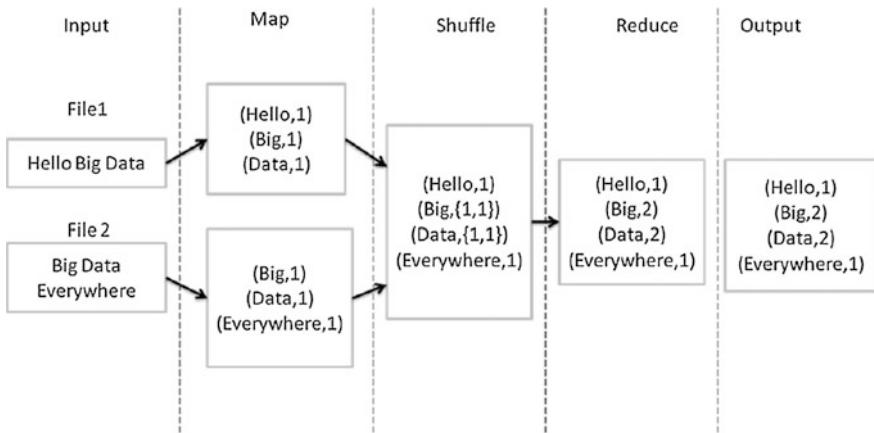


Fig. 2.6 MapReduce on word count

Map function takes the input, and reduce function will sum up the values to get the output. The $\langle\text{key},\text{value}\rangle$ pairs are obtained from the input based on the delimiter used. For example, in a csv file ‘,(comma)’ is used as a delimiter to separate the input for MapReduce tasks. These $\langle\text{key},\text{value}\rangle$ pairs will be shuffled by assigning a value to each key obtained from the input. The sort phase is used to sort the $\langle\text{key},\text{value}\rangle$ pairs first based on the keys. Finally, reduce phase sums up all the values corresponding to a key to produce the output.

The data nodes in the Hadoop are assigned MapReduce tasks from the NameNode. They can execute simultaneously both MapReduce tasks. The parallelism is exploited in the Hadoop by assigning one of the nodes map task and the other with reduce task. Once the node assigned with map task is finished, it can take the other map tasks for further processing [9, 10].

A small example of MapReduce is illustrated in Fig. 2.6. In Fig. 2.6, the input is split based on the delimiter ‘,’ to count the number of words in the file. The map function takes the $\langle\text{key},\text{value}\rangle$ pairs as $\langle\text{word}, 1\rangle$, where initially all the values are assigned as 1 for all the keys. In the sort phase, the values corresponding to a key are sorted. Finally, in the reduce phase, the values are added to get the count of each word. In this way, MapReduce functions can be used for analytics. More examples are discussed in the upcoming sections of this chapter.

2.6 MapReduce and Hadoop Workflow

In the previous sections, Hadoop and MapReduce were dealt separately and their architectures/frameworks were discussed. In this section, the working of a MapReduce program along with Hadoop is presented. When a MapReduce program is run on Hadoop, it essentially consists of entities, HDFS, client, JobTracker, and the TaskTracker [9].

- HDFS: It is the default file system used for running the MapReduce program. The file on which the MapReduce program would be run is copied into the HDFS first, and then, the MapReduce program is executed.
- JobTracker: It runs on the NameNode to keep track of the jobs being run on the cluster. It is used to manage different MapReduce jobs on the cluster. Once the Hadoop is installed, the Web portal of the NameNode can be accessed to see the status of the jobs in the cluster using JobTracker as shown in Fig. 2.7.
- TaskTracker: It runs on each data node of the cluster and executes the instructions according to the metadata of the NameNode. The MapReduce task assigned by the JobTracker to the data nodes is stored in the queue of the TaskTrackers for each data node.

When a MapReduce program is executed on Hadoop, the following steps are followed as shown in Fig. 2.8.

- Client that needs to run a MapReduce task submits the job to JobTracker running on the Namenode of the Hadoop cluster.
- The JobTracker generates and returns a job id for the submitted MapReduce task to the client. This id is used by the client or the Namenode to stop or kill the job if needed.
- The job resources such as the required jar files, metadata files, input files to the MapReduce tasks are copied from the client to the HDFS that can be accessed by the Namenode as well as Datanodes for processing.
- The JobTracker has now scheduled the job to the Tasktracker running on different Datanodes.
- The Tasktracker runs either the map tasks or reduce tasks as assigned by the JobTracker. Once the job is finished, the results are returned to the JobTracker. It keeps sending the heartbeat messages to the JobTracker indicating the Datanode is up and running.
- The JobTracker collects the final result from all the Datanodes and returns to the client in a prescribed format.

Cluster Summary (Heap Size is 15.56 MB/193.38 MB)

Running Map Tasks	Running Reduce Tasks	Total Submissions	Nodes	Occupied Map Slots	Occupied Reduce Slots	Reserved Map Slots	Reserved Reduce Slots	Map Task Capacity	Reduce Task Capacity	Avg. Tasks/Node	Blacklisted Nodes	Excluded Nodes
0	0	0	1	0	0	0	0	2	2	4.00	0	0

Scheduling Information

Queue Name	State	Scheduling Information
default	running	N/A

Filter (Jobid, Priority, User, Name)
Example: 'user:smith 3200' will filter by 'smith' only in the user field and '3200' in all fields

Running Jobs

none

Fig. 2.7 JobTracker in Hadoop

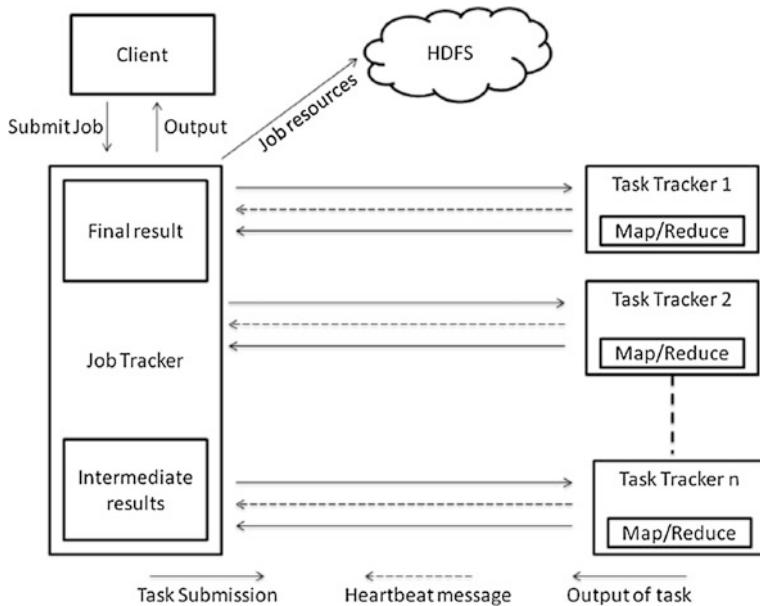


Fig. 2.8 MapReduce job execution in Hadoop

2.7 Installing Hadoop

In this section, the installation of Hadoop is discussed on Ubuntu platform. The prerequisites for Hadoop installation are:

- Java
- Open SSH

Open SSH access is required for Hadoop configuration because the file system needs to be accessed for all the users. The following steps show the installation of Hadoop.

- Download the Hadoop from the <http://hadoop.apache.org/#Download+Hadoop>;
- Extract the file into a folder;
- Update the bash using the following commands.

```

export HADOOP_HOME=.../hadoop
export JAVA_HOME=/usr/lib/jvm/java-6-sun
export PATH=$PATH:$HADOOP_HOME/bin

```

- Update the Java installation path in hadoop-env.sh file
`export JAVA_HOME=/usr/lib/j2sdk1.5-sun`
- Add the following snippets between the <configuration> ... </configuration> tags in the respective configuration conf/hdfs-site.xml file.

```
<property>
<name>hadoop.tmp.dir</name>
<value>/home/user/hadoop_tmp</value>
<description>A base for other temporary directories
.</description>
</property>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:54310</value>
<description>The name of the default file system. A URI whose scheme and
authority determine the FileSystem implementation. The uri's scheme determines
the config property (fs.SCHEME.impl) naming the FileSystem implementation
class. The uri's authority is used to determine the host, port, etc. for a filesystem.
.</description>
</property>
```

2.8 MapReduce Examples

In any programming language, the first program that is executed is *Hello world*. For MapReduce programming, the *hello world* program is *word count* example. The word count programs count the number of words in a file using MapReduce function. In this section, MapReduce example on the word count is discussed.

2.8.1 Word Count Example in Hadoop

To run the word count program in Hadoop, three essential things are needed. One is the input file consisting of random text and the map, reduce functions. For this example, the input file *word_count.txt* consists of the random text as follows.

```
Hello world
This is Hadoop
Hello again
This example illustrates the use of MapReduce in a hadoop environment
we count the number of words in the file
```

2.8.1.1 Map Function for Word Count Example in Hadoop

The following code shows the map function used for the retail analytics example in Hadoop. Each line is taken as the input and converted into lower case first. Each

word in the sentence is separated based on the delimiter as ‘space.’ Each word tokenized as input from the split() method is then passed to the reduce function.

```
import sys
for line in sys.stdin:
    words = line.strip().lower().split()
    for word in words:
        print(word)
```

2.8.1.2 Reduce Function for Word Count Example in Hadoop

The following code demonstrates the reduce function for the word count example in Hadoop. The words passed from the map function are collected first by the reduce function. The word is compared with current word stored. If the word is encountered for the first time, then the count of the particular word is initialized to 1. Otherwise, if the word is encountered for the second time, then the count of the word is incremented. The incremented count of the word is suffixed with the word to get the count of the words in the file.

```
import sys
cur_count = 0
cur_word = None
for line in sys.stdin:
    word = line.strip().lower()
    if cur_word != None and cur_word != word :
        print("{0}\t{1}".format(cur_word,cur_count))
        cur_count = 0
    cur_word = word
    cur_count = cur_count + 1
    if cur_word != None :
        print("{0}\t{1}".format(cur_word,cur_count))
```

2.8.1.3 Execution of Word Count in Hadoop

To execute the word count in Hadoop using the MapReduce functions, the following command is used. The <input_path> refers to the path of the input file that resides in the Hadoop. The <output_path> refers to the path of the output in HDFS where the output is written. For the example considered in the word count input file, the output of the word count is as shown below.

```
hs word_count_map.py word_count_reduce.py <input path> <output path>

a      1
again   1
count   1
environment      1
example  1
file    1
hadoop  2
hello   2
illustrates      1
in     2
is     1
MapReduce 1
number  1
of     2
the    3
this   2
use    1
we    1
words  1
world  1
```

2.8.2 *Retail Analytics Using Hadoop*

In the previous section, a small example on the word count in Hadoop was discussed. Though it seems simple for doing programming in MapReduce, for advanced problems it needs more coding. In this section, a small example on the retail analysis of an online shopping platform is considered.

2.8.2.1 **Retail Analytics Random File Input**

For the case study considered here, the input file consists of the following data in a csv format. It consists of the fields *order_id*, *date*, *timestamp*, *location_store*, *department*, *amount*, and *type of purchase*. Only few rows are shown below. The actual dataset considered for the analysis consists of 200 instances of purchases. The file is initially copied into the HDFS using the command

```
hadoop fs -copyFromLocal ....sales.csv <hadoop-path>
```

```
2809526,05-09-2012,15:55,Laredo,Books,381.96      Visa
2948163,18-09-2012,09:10,Spokane      Pet,Supplies
96.07,Discover
1264295,22-04-2012,13:48,Boise,Baby,65.54,Discover
954786,26-03-2012,11:20,Los      Angeles,Children's Clothing,42.61,
```

```
MasterCard
849639,16-03-2012,17:52,Indianapolis,Garden,349.35,Cash
3707940,23-11-2012,17:53,Spokane,Men'sClothin, 353,Amex
```

The analytics considered for this file is total profit made by individual department, i.e., supplies, children's clothing, men's clothing. In the next section, the MapReduce functions for the analytics are discussed.

2.8.2.2 Map Function for Retail Analytics Example in Hadoop

The following code shows the map function shows the retail analytics example in Hadoop. The dataset considered has 7 data features, and thus, initially the length of the data is checked to see whether the data fetched is correct or not. The aim of the analysis is to calculate the total amount for each department in the sales. Thus, data [4] fetch the department, and data [5] fetch the amount purchased by the customer for that department. The key-value pairs (product, sales) where product represents the department from the input file are passed to the reduce function for calculating the total sales for the department.

```
import sys
for line in sys.stdin:
    data = line.strip().lower().split(',')
    if len(data) != 7:
        continue
    print(data[4]+'\t'+data[5])
```

2.8.2.3 Reduce Function for Retail Analytics Example in Hadoop

The following code shows the reduce function for retail analytics example in Hadoop. The pairs (product, sales) are collected from the map function to calculate the overall sales for that product. Initially, the current sales *cur_sales* are initialized to zero. The product from the pair (product, sales) is checked to see whether it is already there to increment the current total revenue of that product. In this way, the reduce function is used to calculate the total revenue for each product.

```
import sys
cur_sales = 0
cur_product = None
for line in sys.stdin:
    data = line.strip().lower().split('\t')
    if len(data) != 2:
        continue
```

```

product, revenue = data
if(cur_product != None and cur_product != product):
    print('{0}\t{1}'.format(cur_product,cur_sales))
    cur_sales = 0
cur_product = product
cur_sales = cur_sales + float(revenue)
if( cur_product != None):
    print("{0}\t{1}".format(cur_product,cur_sales))

```

2.8.2.4 Execution of Retail Analytics Example in Hadoop

The MapReduce functions can be executed in the following way to see the overall profit of each department. The *<input path>* refers to the path of the data file stored in the HDFS. The *<output path>* refers to the path of the output where the results of the analytics are stored. The output of the execution is as shown. It can be seen that the product *baby* has a total revenue of 1953.47, *cameras* have a total revenue of 4089.83, and so on.

```
hs retail_map.py retail_reduce.py<input path><output path>
```

```

baby 1953.47
books      2277.980000000005
cameras    4089.83
cds       2542.15
children's clothing     3274.190000000005
computers   2858.42
consumer electronics  1475.560000000002
crafts      2749.55
dvds       3702.71
garden      1570.620000000001
health and beauty 2737.37
men's clothing    3435.220000000003
music       2342.039999999995
pet supplies     3825.66
sporting goods   2161.069999999997
toys        2582.05
video games      3405.069999999993
women's clothing 2548.58

```

In this way, MapReduce programming can be used to carry out retail analytics. The example demonstrated in this section can be further extended to cases like overall sales at a particular location, overall sales based on the type of transaction,

overall sales of 2 departments in a particular region. These types of analysis help in understanding the customer behavior, products demand, and availability.

2.9 Call Log Analytics Using Hadoop

The call log records are used by the telecom operators for billing, calculating the customer response, network lags, etc. In this section, call log analytics is discussed with MapReduce as the example. The main aim is to analyze the call log records using MapReduce programming model.

2.9.1 Call Log Analytics Random File Input

The input file considered for the call log analytics is as shown below. It consists of the features *subscriber_phone_number*, *cell_id*, *timestamp*, *call_duration*, *phone_id*, *status*, and *type_of_call*. Some of the rows of the file are as shown below. The dataset consists of 50 instances, and the dataset is initially copied into the HDFS using the following command. The *output_path* refers to the path of the output file where analysis results need to be stored.

```
hadoop fs -copyFromLocal <input_path>..../call_log <output_path>
```

98869801	78859801	'12:05:02'	15	1	'busy'	'voice'
98869802	78859802	'10:06:38'	12	1	'busy'	'voice'
98869803	78859803	'12:05:02'	15	1	'busy'	'voice'
98869805	78859805	'12:05:02'	15	1	'busy'	'voice'
98869804	78859805	'12:05:02'	15	1	'busy'	'voice'

The different types of analysis that can be carried out on the call data record can be:

- Call records that have particular duration;
- Call records that occurred during a particular time;
- Call records of a specific type and a particular duration.

2.9.2 Map Function for Call Log Analytics Example in Hadoop

The map() function for different types of analysis is as shown in the following code. The data are read from the csv file that contains seven features. The delimiter used is ','. Once the data are read, the appropriate columns are selected for analysis. Here, five different types of analysis are considered that are listed as follows.

- Call data records which have call duration of at least 15 min;
- Call data records which have call type as ‘voice’ and call duration less than 10 min;
- Call data records which have call type as ‘sms’ and call duration more than 10 min;
- Call data records that are before 12 p.m. and call type as ‘voice’;
- Call data records that are between 12 p.m. and 1 p.m. and call type as ‘sms’.

```
import sys
for line in sys.stdin:
    data = line.strip().split(',')
    if len(data) != 7:
        continue

#Analysis 1: Select the rows which has call duration of at least 15 mins
if( int(data[3]) >= 15):
    for item in data:
        print(item,end='\t')
    print()

#Analysis 2: Select the rows which has call type as 'voice' and call
duration less than 10 mins
if( data[6] == '\'voice\''):
    for item in data:
        print(item,end='\t')
    print()

#Analysis 3: Select the rows which has call type as 'sms' and call
duration more than 10 min
if( data[6] == '\'sms\''):
    for item in data:
        print(item,end='\t')
    print()

#Analysis 4:
Select the calls that are before 12 pm and call type as 'voice'.
if( data[6] == '\'voice\''):
    for item in data:
        print(item,end='\t')
    print()
```

#Analysis 5: Select the call that are between 12 pm-1 pm and call type as ‘sms’.

```
if( data[6] == '\`sms\`' ):
    for item in data:
        print(item,end='\t')
    print()
```

The appropriate data features are selected for each analysis. For example, data [3] represent the *call_duration*, data [6] represent the *call_type*, and so on. The mapper function for each analysis needs to be saved in a separate file for further execution. All the analysis cannot be carried out with a single file. The execution steps are discussed in further sections.

2.9.3 Reduce Function for Call Log Analytics Example in Hadoop

The following code demonstrates the reduce function of the call log analytics. In each of the reduce functions for each of the map functions considered in the previous section, the appropriate columns in the data are selected to get the output. The code is similar to the map functions, and not much changes are needed to the reduce functions. For example, for the analysis case 4, the timestamp is first split based on the delimiter ‘:’. In this way, the reduce function is used for carrying out the analysis in each case.

```
import sys
for line in sys.stdin:
    data = line.strip().split('\t')
    if len(data) != 7:
        continue

#Analysis 1 : Select the rows which has call duration of at least 15 mins
print(line,end='')

#Analysis 2: Select the rows which has call type as 'voice' and call
duration less than 10 mins
if( int(data[3]) < 10):
    for item in data:
        print(item,end='\t')
    print()
```

```

#Analysis 3: Select the rows which has call type as 'sms' and call duration more than 10 min
if( int(data[3]) >= 10):
    for item in data:
        print(item,end='\t')
    print()

#Analysis 4:Select the calls that are before 12 pm and call type as 'voice'.
time = data[2].strip('\"').split(':')
if( int(time[0]) >= 12):
    for item in data:
        print(item,end='\t')
    print()

#Analysis 5: Select the call that are between 12 pm-1 pm and call type as 'sms'.
if data[2].startswith('\'12:'):
    for item in data:
        print(item,end='\t')
    print()

```

2.9.4 Execution of Call Log Analytics Example in Hadoop

The MapReduce functions for call log analytics in each analysis case need to be stored separately for execution. For example, the analysis 1 case can be stored as map_analysis1 and reduce_analysis1. The execution of the MapReduce can be done using the following command. The *<input_path>* refers to the csv file of the input path. Here, the execution refers to the case of the analysis 1 where the call data records that are less than 15 min are selected. The output of the analysis 1 case is as shown, where the call records of less than 15 min are selected.

```
hs  call_log_map_analysis1.py  call_log_reduce_analysis1.py  <input_path>
<output_path>
```

Analysis 1 output

98869801	78859801	'12:05:02'	15	1	'busy'	'voice'
98869803	78859803	'12:05:02'	15	1	'busy'	'voice'
98869805	78859805	'12:05:02'	15	1	'busy'	'voice'
98869804	78859805	'12:05:02'	15	1	'busy'	'voice'
98869806	78859806	'01:05:02'	15	1	'busy'	'voice'
98869807	78859807	'01:05:02'	15	1	'busy'	'voice'
98869808	78859808	'12:05:02'	15	1	'busy'	'voice'
98869809	78859809	'12:05:02'	15	1	'busy'	'voice'
98869810	78859810	'12:05:02'	15	1	'busy'	'voice'

98869812	78859812	'11:05:02'	15	1	'busy'	'sms'
98869811	78859811	'10:05:02'	15	1	'busy'	'sms'
98869813	78859813	'11:05:02'	15	1	'busy'	'sms'
98869814	78859814	'11:05:02'	15	1	'busy'	'sms'
98869815	78859815	'11:05:02'	15	1	'busy'	'sms'
98869816	78859816	'11:05:02'	15	1	'busy'	'sms'
98869817	78859817	'11:05:02'	15	1	'busy'	'voice'
98869818	78859818	'11:05:02'	15	1	'busy'	'voice'
98869819	78859819	'11:05:02'	15	1	'busy'	'voice'
98869820	78859820	'11:05:02'	15	1	'busy'	'voice'

Similar to the analysis 1 case, the other cases are also executed using the same approach as follows.

```
hs  call_log_map_analysis2.py  call_log_reduce_analysis2.py  <input_path>
<output_path>
```

Analysis 2 output

98869821	78859821	'5:05:02'	0	2	'failed'	'voice'
98869822	78859822	'5:05:02'	0	2	'failed'	'voice'
98869823	78859823	'5:05:02'	0	2	'failed'	'voice'
98869824	78859824	'5:05:02'	0	2	'failed'	'voice'
98869825	78859825	'4:05:02'	0	2	'failed'	'voice'
98869826	78859826	'5:05:02'	0	2	'failed'	'voice'
98869827	78859827	'5:05:02'	0	2	'failed'	'voice'
98869828	78859828	'4:05:02'	0	2	'failed'	'voice'
98869829	78859819	'6:05:02'	0	2	'failed'	'voice'
98869830	78859830	'5:05:02'	0	2	'failed'	'voice'
98869838	78859838	'14:05:02'	0	2	'failed'	'voice'
98869839	78859839	'16:05:02'	0	2	'failed'	'voice'
98869840	78859840	'15:05:02'	0	2	'failed'	'voice'
98869848	78859848	'14:05:02'	0	2	'failed'	'voice'
98869849	78859849	'16:05:02'	0	2	'failed'	'voice'
98869850	78859850	'15:05:02'	0	2	'failed'	'voice'

Analysis 3 execution

```
hs  call_log_map_analysis3.py  call_log_reduce_analysis3.py  <input_path>
<output_path>
```

Analysis 3 output

98869812	78859812	'11:05:02'	15	1	'busy'	'sms'
98869811	78859811	'10:05:02'	15	1	'busy'	'sms'
98869813	78859813	'11:05:02'	15	1	'busy'	'sms'
98869814	78859814	'11:05:02'	15	1	'busy'	'sms'
98869815	78859815	'11:05:02'	15	1	'busy'	'sms'
98869816	78859816	'11:05:02'	15	1	'busy'	'sms'

Analysis 4 execution

```
hs  call_log_map_analysis2.py  call_log_reduce_analysis2.py  <input_path>
<output_path>
```

Analysis 4 output

98869801	78859801	'12:05:02'	15	1	'busy'	'voice'
98869803	78859803	'12:05:02'	15	1	'busy'	'voice'
98869805	78859805	'12:05:02'	15	1	'busy'	'voice'
98869804	78859805	'12:05:02'	15	1	'busy'	'voice'
98869808	78859808	'12:05:02'	15	1	'busy'	'voice'
98869809	78859809	'12:05:02'	15	1	'busy'	'voice'
98869810	78859810	'12:05:02'	15	1	'busy'	'voice'
98869838	78859838	'14:05:02'	0	2	'failed'	'voice'
98869839	78859839	'16:05:02'	0	2	'failed'	'voice'
98869840	78859840	'15:05:02'	0	2	'failed'	'voice'
98869848	78859848	'14:05:02'	0	2	'failed'	'voice'
98869849	78859849	'16:05:02'	0	2	'failed'	'voice'
98869850	78859850	'15:05:02'	0	2	'failed'	'voice'

Analysis 5 execution

```
hs  call_log_map_analysis2.py  call_log_reduce_analysis2.py  <input_path>
<output_path>
```

Analysis 5 output

98869841	78859841	'12:05:02'	0	2	'failed'	'sms'
98869842	78859842	'12:05:02'	0	2	'failed'	'sms'
98869843	78859843	'12:05:02'	0	2	'failed'	'sms'
98869844	78859844	'12:05:02'	0	2	'failed'	'sms'
98869845	78859845	'12:05:02'	0	2	'failed'	'sms'

2.10 Network Log Analytics Using Hadoop

In this section, one more case study on network log analytics is discussed using MapReduce programming model. In a network log of telecom companies, the different features that are available are *IMSI*, *IMEI*, *call_type*, *cell_id*, *subscriber_no*, *latitude*, and *longitude*. In this section, the different types of analysis that can be carried out using MapReduce are discussed.

2.10.1 Network Log Analytics Random File Input

A random file input is considered for the network log as shown below. The dataset considered for the network log analytics consists of 51 instances, out of which few rows are as shown below. The different types of analysis that are carried out using this file are:

- IMEI numbers for the given latitude and longitude;
- IMEI numbers for the given call type;
- IMSI for the given latitude and longitude;
- Subscriber phone numbers that belong to a particular latitude and longitude.

timestamp	IMSI	IMEI	call_type	cell_id	subscriber	latitude	longitude
'07:09:42'	'410072821393801'	'123456789123401'	'sms'	'1'	'9886598801'	'66.5 N'	'56.4S'
'17:09:42'	'410072821393801'	'123456789123402'	'sms'	'1'	'9886598802'	'66.5 N'	'56.4S'
'17:09:42'	'410072821393801'	'123456789123403'	'sms'	'1'	'9886598803'	'66.5 N'	'56.4S'
'17:09:42'	'410072821393801'	'123456789123404'	'sms'	'1'	'9886598804'	'66.5 N'	'56.4S'

2.10.2 Map Function for Network Log Analytics Example in Hadoop

For the input file considered, the following code shows the map() function for analysis. Since there are eight data features in the input, the data length is first checked to see whether the length of the data is correct or not. For each analysis case considered, the appropriate data columns are used. For example, data [6] give the latitude, data [7] give the longitude, data [2] give the IMEI numbers, and so on. In this way, for each analysis appropriate data are selected from the dataset considered. The map functions for each of the analysis cases need to be stored in separate files for execution.

```
import sys
for line in sys.stdin:
    data = line.strip().split(',')
    if len(data) != 8:
        continue

    #Analysis 1: IMEI numbers for the latitude and longitude
    lat = '\'16.5 N\''
    lon = '\'56.4S\''
    if (data[6] == lat and data[7] == lon):
        print(data[2])

    #Analysis 2: IMEI numbers for the call type voice and cell id = 5
    if (data[3] == '\'voice\''):
        print("{0}\t{1}".format(data[2],data[4]))

    #Analysis 3: IMSI for the given latitude and longitude
    lat = '\'46.5 N\''
    lon = '\'55.4S\''
    if (data[6] == lat and data[7] == lon):
        print(data[1])

    #Analysis 4: Subscriber phone numbers that belong to a particular
latitude and longitude
    lat = '\'76.5 N\''
    lon = '\'56.4S\''
    if (data[6] == lat and data[7] == lon):
        print(data[5])
```

2.10.3 Reduce Function for Network Log Analytics Example in Hadoop

The following code demonstrates the reduce functions for the analysis cases considered in the previous map functions. In the analysis case 1, since only IMEI numbers are required, data [0] are only selected from the key–value pairs of the map function. Similarly, for the other analysis cases, respective data are extracted from the dataset for analysis.

```
import sys
cur_num = ""
for line in sys.stdin:
    data = line.strip().split('\t')

    #Analysis 1: IMEI numbers for the latitude and longitude
    print(data[0])

    #Analysis 2: IMEI numbers for the call type voice and cell id = 5
    if data[1] == '\5\':
        print(data[0])

    #Analysis 3: IMSI for the given latitude and longitude
    if (cur_num != data[0]):
        print(data[0])
        cur_num = data[0]

    #Analysis 4: Subscriber phone numbers that belong to a particular
    #latitude and longitude
    if (cur_num != data[0]):
        print(data[0])
        cur_num = data[0]
```

2.10.4 Execution of Network Log Analytics Example in Hadoop

The MapReduce functions for network log analytics in each analysis case need to be stored separately for execution. For example, the analysis 1 case can be stored as network_log_map_analysis1 and network_log_reduce_analysis1. The execution of the MapReduce can be done using the following command. The <input_path> refers to the csv file of the input path. Here, the execution refers to the case of the analysis 1 where the IMEI numbers of a particular region are selected. The output of the analysis 1 case is as shown, where the IMEI numbers are selected. Similarly, for other cases, the execution commands and the output are as shown below.

```
hs network_log_map_analysis1.py network_log_reduce_analysis1.py <input_path>
<output_path>
```

Analysis 1 output

```
'123456789123411'
'123456789123412'
'123456789123414'
'123456789123415'
'123456789123416'
'123456789123417'
'123456789123418'
'123456789123420'
```

Analysis 2 execution

```
hs network_log_map_analysis2.py network_log_reduce_analysis2.py <input_path>
<output_path>
```

Analysis 2 output

```
'123456789123416'
'123456789123417'
'123456789123426'
'123456789123427'
```

Analysis 3 execution

```
hs network_log_map_analysis3.py network_log_reduce_analysis3.py <input_path>
<output_path>
```

Analysis 3 output

‘410072821393822’
‘410072821393841’
‘410072821393852’

Analysis 4 execution

hs network_log_map_analysis4.py network_log_reduce_analysis4.py <input_path> <output_path>

Analysis 4 output

‘9886598806’
‘9886598807’
‘9886598808’
‘9886598810’

2.11 Exercises

1. Describe the workflow of MapReduce in Hadoop.
2. Is the NameNode a fallback for Hadoop? Explain the significance of it with Hadoop’s architecture.
3. For the case study of call log analytics in Hadoop, change the MapReduce function to retrieve the total calls made by a particular phone number.
4. Write a MapReduce program by considering an input in the following way and determine the following.
 - (i) The number of times a user has logged in total.
 - (ii) The number of times a user has logged in at a particular date.

Input:

21-02-17 user1
22-07-17 user 2
23-02-18 user 3

25-02-18 user1

5. Write a MapReduce program that counts the department-wise salary for the following dataset.

EmpId	EmpName	DeptName	Salary
1	Mahesh	CSE	25,000
2	Raja	CSE	45,000
3	Noori	ECE	20,000
4	Rashmi	ECE	20,000

References

1. Cafarella, M., & Cutting, D. (2004, April). *Building Nutch: Open source search*. ACM Queue, <http://queue.acm.org/detail.cfm?id=988408>.
2. Hadoop, A. (2009). *Hadoop*. 2009-03-06. <http://hadoop.apache.org>.
3. Borthakur, D. (2007). *The Hadoop distributed file system: Architecture and design*. Retrieved from January 5, 2013.
4. Dean, J., & Ghemawat, S. (2010). MapReduce: A flexible data processing tool. *Communications of the ACM*, 53(1), 72–77. <https://doi.org/10.1145/1629175.1629198>.
5. David, P. (2012). *The big data hub: Understanding big data for the enterprise*. Retrieved December 1, 2012, from <http://www.ibmbigdatahub.com/blog/lords-datastorm-vestas-and-ibm-win-bigdata-award>.
6. Dean, J., & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107–113. <https://doi.org/10.1145/1327452.1327492>.
7. Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., & Sears, R. (2009). *MapReduce online* (Tech. Rep. UCB/EECS-2009-136). Berkeley, CA: University of California.
8. Hedlund, B. (2010). *Understanding Hadoop clusters and the network*. Studies in Data Center Networking, Virtualization, Computing.
9. White, T. (2012). *Hadoop: The definitive guide*. “O’Reilly Media, Inc.”. Bhandarkar, M. (2010, April). MapReduce programming with apache Hadoop. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, (pp. 1-1).
10. Xu, C. (2018). *Big data analytic frameworks for GIS* (Amazon EC2, Hadoop, Spark).

Chapter 3

Apache Hive



3.1 Introduction

As the computer evolution made progress, databases were often used for managing the data. Relational database systems (RDBMS) are widely used in connecting the physical data for carrying out businesses. Structured Query Language (SQL) was effectively used on the RDBMS for querying processes. SQL was used by a wide range of developers and users for database application development.

As the volume of the data started to increase, there was great difficulty in handling large amounts of data. Data warehouses came into existence at this point of time. A typical data warehouse consists of a various formats of data from different sources and acts as a centralized component for getting the data. Various techniques of getting the data from the data warehouse such as extract, transform and load (ETL) were developed as a part of the warehouse solutions. SQL allowed databases to interconnect and perform queries through joins, aggregations. The format of the data until 2000s was normalized and structured. Thus, RDBMS with SQL solutions were helpful in analyzing the data and getting results.

After the 2K era, Internet technologies advanced and gradually the volume of the data increased with different variety. New paradigms like social media analytics, Web mining, data visualizations emerged that helped the companies to gain insights into customer markets, products, reviews of products, and competitions. A discrete set of new solutions were needed to manage and analyze the data easily. Hence, Big data analytics came into the trend with major ecosystems as Hadoop, Hive, HBase, and others. In this chapter, the data warehouse solution Hive, a component of the Hadoop ecosystem, is discussed.

Hive is a data warehouse solution that helps in analyzing the data stored in Hadoop file system as well as the local file system [1]. The main aim of developing Hive as warehouse solution was to facilitate SQL developers to adapt to the Hadoop

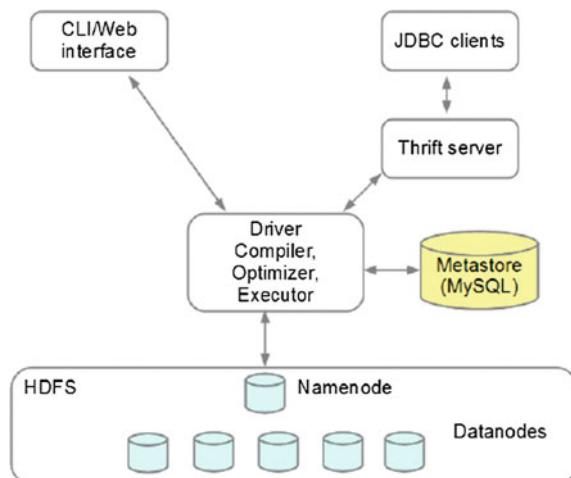
ecosystem. While using Hadoop with alone HDFS, MapReduce programming is needed as a prerequisite for gaining the results. Thus, Hive provides an alternative solution to MapReduce programming by offering the data stored in tabular formats and use SQL like queries for obtaining the results. The key points of Apache Hive are highlighted as follows:

- Hive is a warehouse solution for with HiveQL as the language for querying.
- Hive supports data querying and analysis on HDFS.
- It is insert only warehouse solution, where updates are not allowed for a specific row. However, alternative solutions with overwrite and updates are allowed.
- It supports ad hoc querying on HDFS.
- It supports JDBC/ODBC drivers for pulling and retrieving the data to/from Hive.
- It supports user-defined functions and customized I/O functions as function extensions.

3.2 Hive Architecture

Hive warehouse solution built on top of Hadoop consists of different architectural elements as shown in Fig. 3.1. The essential elements of Hive are Thrift server, JDBC clients, Web interface, and drivers [2]. Hive warehouse is built on top of HDFS that provides the base file systems for storing the data. The data nodes in the HDFS are used for storing the data in tables.

Fig. 3.1 Hive architecture



- **CLI/Web interface:** To access the Hive, a command line interface (CLI) or Web interface can be used. CLI can be accessed using the command *Hive* that opens the prompt as follows:

```
'Hive>>.....'
```

For accessing the Web interface, hue can be used to see the tables in Hive. Hue is a Web browser that provides access to the tables in Hive, and queries can also be written using it. The Web portal of Hive using Hue is as shown in Fig. 3.2.

- **Thrift server:** There can be different number of clients that are connected to the Hive. A thrift server in Hive manages the connections among different clients and provides access to the tables in Hive.
- **Metastore:** The metadata of the tables such as partitions, column types, and join information are stored in the metastore of the Hive. These are needed for supporting other types of clients outside the Hadoop ecosystem that are connected to the Hive.
- **Driver/Compiler/Executor:** This component in the Hive contains drivers for connecting the views and its contents in Hive. It supports JDBC/ODBC drivers for connection, inserting, and querying the data in Hive.
- The query submitted to the Hive in the form of HiveQL is converted into MapReduce job internally by HiveQL compiler. The results of the MapReduce job are presented in the output of the Hive tables. The executor in the Hive architecture executes the Hive queries by converting into the MapReduce jobs.
- **JDBC clients:** MySQL developers familiar with programming in SQL can use the JDBC drivers supported by Hive in the same way for programming. These JDBC drivers are needed for external connections and queries.

With these essential components, Hive can be used for storing data and executing queries on it. HiveQL called as Hive query language is used for querying in

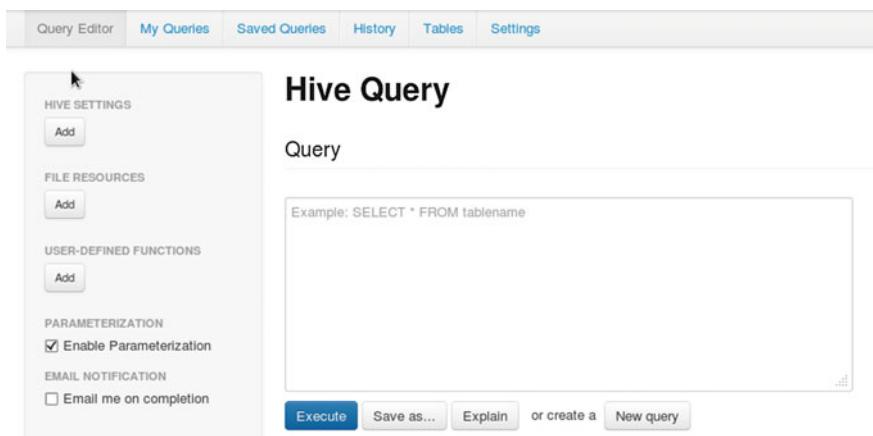


Fig. 3.2 Hue browser for Hive

Hive. Any query submitted through Hive both from CLI/programming, it is converted into a MapReduce job internally and then the results are displayed.

3.3 Installing Hive

In this section, the procedure for installation and configuration of Hive is discussed. The examples used in this chapter are shown in the Ubuntu platform. The prerequisites for installation of Hive are as follows:

- JDK (Any version 1.7 and above)
- Hadoop (0.20.x,1.x.y,2.x.y)
- Ubuntu 14.04

The steps to install Hive in a Linux platform are as follows:

- Download Hive from <https://hive.apache.org/downloads.html> for the Ubuntu platform.
- Unpack the tar file apache-hive-x.x.x-bin.tar.gz
- Add Hive to the system path using the following commands

```
export HIVE_HOME=.../hive/apache-hive-x.x.x-bin  
export PATH=$PATH:$HIVE_HOME/bin:$HIVE_HOME/conf
```

- The settings of the path need to be added to the profile in/etc./profile.
- The configuration files needed for starting the Hive are created as follows.

```
cp hive-default.xml.template to .../hive/conf/hive-site.xml  
cp hive-env.sh.template to .../hive/conf/hive-env.sh  
cp hive-log4j.properties.template to .../hive/conf/ log4j.properties
```

- The environment is configured using the following commands.

```
export HIVE_HOME=.../apache-hive.x.x.x/conf
```

Once the installation is completed in the command prompt, the command 'hive' should show the following output:

```
[training@localhost ~]$ hive  
Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties  
Hive history file=/tmp/training/hive_job_log_training_201802080607_325567410.txt  
hive> ■
```

In this way, the installation of Hive can be done. The troubleshooting options can be referred in the site <https://cwiki.apache.org/confluence/display/Hive/Home> that gives the information on resolving the issues in Hive.

In the next section, first the different data types supported in Hive is discussed with examples and then the DDL and DML operations are presented with examples and case studies.

3.4 Hive Data Types

In this section, the different data types supported in Hive are discussed with examples. Table 3.1 consists of data type, description, and the syntax to use in Hive [3]. All of these data types are demonstrated as a part of call log analytics and network log analytics case study that is discussed in this chapter. These data types are mainly used in most of the scenarios for querying purposes.

3.5 Hive Data Description (DDL)

In this section, some of the small examples on Hive DDL are discussed. The DDL statements are primitive statements that are used for creating databases, updating databases, and using databases, etc. These constructs are presented with examples in this section.

3.5.1 *Creating Database*

The general format for creating a database is as follows:

```
CREATE (DATABASE|SCHEMA) [IF NOT EXISTS] database_name
[COMMENT database_comment]
[LOCATION hdfs_path]
```

Table 3.1 Hive data types

Data type	Description	Example
int	It specifies integer data in the table	<i>create table data_type (d1int)</i>
float	It specifies float data in the table	<i>create table data_type (d2float)</i>
double	It specifies double data in the table	<i>create table data_type (d3double)</i>
decimal	It specifies decimal data in the table	<i>create table data_type (d4 decimal)</i>
binary	It specifies binary data in the table	<i>create table data_type (d5binary)</i>
string	It specifies string data in the table	<i>create table data_type (d6 string)</i>
boolean	It specifies boolean data in the table	<i>create table data_type (d7 boolean)</i>
date	It specifies date data in the table	<i>create table data_type (d8 date)</i>
timestamp	It specifies the timestamp in the table	<i>create table data_type (d9 timestamp)</i>

[WITH DBPROPERTIES (property_name=property_value, ...)];

Here,

- DATABASE|SCHEMA and database_name represent the database name that need to be used with the keyword database.
- COMMENT specifies the description of the database. It used for identifying a database from a list.
- LOCATION specifies the HDFS path where the database needs to be maintained. It is optional if not used, then by default it will be maintained by HDFS under the Hive directory.
- DBPROPERTIES is an optional parameter that specifies the additional properties of the database such as key-value pairs created in the Hive table and its associations.

Example

In this example, a database by name *learning* is created in order to see if the database is created or not using the command ‘*show databases*’. If the database with the name already exists, then an error message is shown saying the database already exists.

```
Hive> create database learning;
OK
Time taken: 2.637 seconds
```

```
Hive> show databases;
OK
default
learning
Time taken: 0.343 seconds
```

```
Hive> create database learning;
Database learning already exists
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask
```

3.5.2 Using Database

The general format for using a database is as follows:

USE database_name;

The database name needs to be used for using a particular database, otherwise *default* database is used for analysis.

Example

The database created earlier with the name *learning* can be used as the default database as follows.

```
Hive> use learning;
OK
Time taken: 1.896 seconds
```

3.5.3 *Create Table*

In Hive, there are two types of table that can be created. One is an internal table, and another is an external table. In the internal table, the data is loaded using the file stored within HDFS. In the case of external table, the data are loaded using the file within the local system. The general format for creating a table in Hive is as follows:

```
CREATE TABLE <TABLENAME><COLUMN_NAME DATATYPE>
(Optional)ROW FORMAT DELIMITED FIELDS TERMINATED BY <,\t>
(Optional)STORED AS TEXT FILE <location>
```

where

- <TABLENAME> is the name of the table to be created
- ROW FORMAT is the format of the rows to be loaded into the table. If the file is in csv format the delimiter ‘,’ is used.
- The file location specifies where the file is stored for loading the data into the table.

For creating the external table, the external keyword is used as shown.

```
CREATE EXTERNAL TABLE <TABLENAME><COLUMN_NAME
DATATYPE>....
```

The difference between the internal table and the external table is the change in the warehouse directory. If an internal table is created by default, the table is stored in/user/Hive in HDFS. If the table is dropped, then the metadata of the table stored in/user/Hive are also deleted. But, in the case of external table, the metadata of the table are stored in the user-specified directory. When an external table is dropped, the metadata are not deleted.

Example

The following example creates a student table with features *name*, *age*, *branch*, and *cgpa*. An external is also created with the same file using the keyword external.

```
Hive> create table students(name string, age int, branch string,
  cgpa float) row format delimited fields terminated by ',';
OK
Time taken: 0.081 seconds
```

```
Hive> create external table student_external (name string, age int, branch  
string, cgpa float) row format delimited fields terminated by ',';  
OK  
Time taken: 0.085 seconds
```

3.6 Hive Operations

The different types of operations that can be done on the Hive table are load, select, join, partition, etc. [4, 5]. These operations are for data manipulation which is abbreviated as DML. Some of the DML operations such as load, select, and Join are discussed in this section.

3.6.1 Load the Data

The load operation is used for inserting the data into the table. Once the load operation is executed, the entire file is copied/moved to the metadata location of the Hive directory. The general syntax for loading the data into Hive table is as follows:

```
LOAD DATA <LOCAL> INPATH ‘filepath’ <OVERWRITE> into TABLE  
<TABLENAME> [PARTITION (partcol1=val1, partcol2=val2.....)]
```

where

- <LOCAL> is an optional parameter that is used when the data reside in the local file system.
- ‘filepath’ is the path of the location of the file from which the data are to be loaded.
- <OVERWRITE> is an optional parameter to overwrite the table contents in the database.
- <TABLENAME> is the name of the table where the data are to be loaded.

Example

In the following example, the data are loaded into the students table created earlier in the database learning. The data explorations can be done using select statements.

```
Hive> load data inpath ‘/user/training/  
hive_input_to_table’ into table students;  
Loading data to table learning.students  
OK  
Time taken: 0.156 seconds
```

The load operation for an external table is same as the internal table, and there are no changes. The external table *students* created in the earlier examples can be loaded with the data as follows. A local input file stored in/home/training is given as the input to the file.

```
hive> load data local inpath  
'/home/training/hive_input_to_table' into table  
student_external;  
Copying data from file:/home/training/hive_input_to_table  
Copying file: file:/home/training/hive_input_to_table  
Loading data to table learning.student_external  
OK  
Time taken: 0.268 seconds
```

3.6.2 Data Exploration Using Select

The data loaded into the Hive tables can be explored and viewed using *select* statement. The syntax of using the select statement is same as SQL where the column names and table name are specified. The following queries give the results of the students table created earlier.

```
hive> select * from students;  
OK  
Rahul      18    CSE    9.18  
Rakesh     20    ISE    9.15  
Sameer     25    ECE    8.17  
Abhishek   20    CSE    8.15  
Abhinandan 19    ISE    7.13  
Asha        18    ECE    6.18  
Shashank   19    CSE    7.89  
Time taken: 0.153 seconds
```

```
hive> select * from student_external;  
OK  
Rahul      18    CSE    9.18  
Rakesh     20    ISE    9.15  
Sameer     25    ECE    8.17  
Abhishek   20    CSE    8.15  
Abhinandan 19    ISE    7.13  
Asha        18    ECE    6.18  
Shashank   19    CSE    7.89  
Time taken: 0.129 seconds
```

3.7 Hive Examples on Network Log

In telecommunication, mobile call records play an important role and it is the main data for analysis. Mobile call records are used for billing, customer analysis, recharge offers, network lags, and others [6]. In this section, two case studies on the network log, namely call data record and network log record, are discussed. The main aim of this section is to analyze the records to get necessary information on the calls using Hive.

3.7.1 Case Study 1: Call Data Record

In this section, a case study on the call data record is considered for analysis using Hive. A call data record (CDR) is often used by telecommunication companies for billing that contains the information such as call duration time, initiation, termination, and Internet services. This information is logged into a file for analysis. Some of the fields that are present in CDR can be listed as follows:

- Subscriber phone number;
- Recipient phone number;
- Start timestamp;
- Call duration;
- Record ID;
- Status of call was busy or failed;
- Call type (voice, SMS, etc.).

Create a table called as ‘CDR’

The table is created according to the fields present in the CDR using the create command as follows:

```
hive> create table cdr(
  subscriber  int
  receiver    int
  times string
  duration    int
  record_id   int
  status string
  call_type   string) row format delimited
  fields terminated by ',';
OK
Time taken: 0.13 seconds
```

Load the data into table cdr

The table created for the call data record earlier is loaded with the data using the following command. Some of the rows of the data inserted into the table are listed as follows.

```
Hive> load data inpath '/user/training/cdr.txt' into table cdr;
Loading data to table learning.cdr
OK
Time taken: 0.156 seconds
```

Sample data for the table

```
(98869801, 78859801, '12:05:02', 15, 1, 'busy', 'voice'),
(98869802, 78859802, '10:06:38', 12, 1, 'busy', 'voice'),
(98869803, 78859803, '12:05:02', 15, 1, 'busy', 'voice'),
(98869805, 78859805, '12:05:02', 15, 1, 'busy', 'voice'),
(98869804, 78859805, '12:05:02', 15, 1, 'busy', 'voice'),
(98869806, 78859806, '01:05:02', 15, 1, 'busy', 'voice'),
```

Analysis on call data record table

The data inserted into the table can be analyzed to see some of the information on the call data records. In this section, some of the HiveQL queries are executed on the call data record table to check for the information like call duration, call type.

Query1: Call duration of at least 15 min.

The following query lists the calls that have duration greater than 15 min.

```
Hive>select * from cdr where duration>=15;
```

subscriber	receiver	timestamp	duration	record_id	status	call_type
98869801	78859801	12:05:02	15	1	busy	voice
98869803	78859803	12:05:02	15	1	busy	voice
98869805	78859805	12:05:02	15	1	busy	voice
98869804	78859805	12:05:02	15	1	busy	voice
98869806	78859806	01:05:02	15	1	busy	voice
98869807	78859807	01:05:02	15	1	busy	voice
98869808	78859808	12:05:02	15	1	busy	voice
98869809	78859809	12:05:02	15	1	busy	voice
98869810	78859810	12:05:02	15	1	busy	voice
98869812	78859812	11:05:02	15	1	busy	sms
98869811	78859811	10:05:02	15	1	busy	sms
98869813	78859813	11:05:02	15	1	busy	sms

subscriber	receiver	timestamp	duration	record_id	status	call_type
98869814	78859814	11:05:02	15	1	busy	sms
98869815	78859815	11:05:02	15	1	busy	sms
98869816	78859816	11:05:02	15	1	busy	sms
98869817	78859817	11:05:02	15	1	busy	voice
98869818	78859818	11:05:02	15	1	busy	voice
98869819	78859819	11:05:02	15	1	busy	voice
98869820	78859820	11:05:02	15	1	busy	voice

Query2: Call type as ‘voice’ and call duration less than 10 min

The following query selects the call data records that have call duration less than 10 min and status as *voice*.

```
Hive>select * from cdr where call_type='voice' and duration<10;
```

subscriber	receiver	timestamp	duration	record_id	status	call_type
98869821	78859821	05:05:02	0	2	failed	voice
98869822	78859822	05:05:02	0	2	failed	voice
98869823	78859823	05:05:02	0	2	failed	voice
98869824	78859824	05:05:02	0	2	failed	voice
98869825	78859825	04:05:02	0	2	failed	voice
98869826	78859826	05:05:02	0	2	failed	voice
98869827	78859827	05:05:02	0	2	failed	voice
98869828	78859828	04:05:02	0	2	failed	voice
98869829	78859819	06:05:02	0	2	failed	voice
98869830	78859830	05:05:02	0	2	failed	voice
98869838	78859838	14:05:02	0	2	failed	voice
98869839	78859839	16:05:02	0	2	failed	voice
98869840	78859840	15:05:02	0	2	failed	voice
98869848	78859848	14:05:02	0	2	failed	voice
98869849	78859849	16:05:02	0	2	failed	voice
98869850	78859850	15:05:02	0	2	failed	voice

Query3: Call type as ‘sms’ and call duration more than 10 min

The following query can be used to select the call data records that have duration greater than 10 min and call type as *sms*.

```
Hive>select * from cdr where call_type='sms' and duration>10;
```

subscriber	receiver	timestamp	duration	record_id	status	call_type
98869812	78859812	11:05:02	15	1	busy	sms
98869811	78859811	10:05:02	15	1	busy	sms
98869813	78859813	11:05:02	15	1	busy	sms

```

98869814 78859814 11:05:02 15      1           busy     sms
98869815 78859815 11:05:02 15      1           busy     sms
98869816 78859816 11:05:02 15      1           busy     sms

```

Query 4: Calls that are before 12 p.m. and call type as ‘voice’.

The following query can be used to select the call data records before 12 p.m. and the type as *voice*. This type of analysis can be used for billing the *voice call records* for a customer on a particular date before 12 p.m.

```
Hive>select * from cdr where call_type='voice' and timestamp <'12:00:00';
```

subscriber	receiver	timestamp	duration	record_id	status	call_type
98869802	78859802	10:06:38	12	1	busy	voice
98869806	78859806	01:05:02	15	1	busy	voice
98869807	78859807	01:05:02	15	1	busy	voice
98869817	78859817	11:05:02	15	1	busy	voice
98869818	78859818	11:05:02	15	1	busy	voice
98869819	78859819	11:05:02	15	1	busy	voice
98869820	78859820	11:05:02	15	1	busy	voice
98869821	78859821	05:05:02	0	2	failed	voice
98869822	78859822	05:05:02	0	2	failed	voice
98869823	78859823	05:05:02	0	2	failed	voice
98869824	78859824	05:05:02	0	2	failed	voice
98869825	78859825	04:05:02	0	2	failed	voice
98869826	78859826	05:05:02	0	2	failed	voice
98869827	78859827	05:05:02	0	2	failed	voice
98869828	78859828	04:05:02	0	2	failed	voice
98869829	78859819	06:05:02	0	2	failed	voice
98869830	78859830	05:05:02	0	2	failed	voice

Query 5: Calls that are between 12 p.m. and 1 p.m. and call type as ‘sms’.

The following query selects the call data records that are between 12 p.m. and 1 p.m. and type as *sms*. This type of analysis is helpful when the charges need to be waived for a particular duration and call type as sms or voice.

```
Hive>select * from cdr where call_type='sms' and timestamp
between '12:00:00' and '13:00:00';
```

subscriber	receiver	timestamp	duration	record_id	status	call_type
98869841	78859841	12:05:02	0	2	failed	sms
98869842	78859842	12:05:02	0	2	failed	sms
98869843	78859843	12:05:02	0	2	failed	sms

```

98869844    78859844  12:05:02  0          2      failed  sms
98869845    78859845  12:05:02  0          2      failed  sms

```

3.7.2 Case Study 2: Network Log

In this case study, network log is considered for analysis. The different types of information that can be there in the network log are subscriber phone number, cell id of the network tower, latitude of the location of the tower, longitude of the location of the tower, and so on. This information can be used by the telecom providers for analyzing the log information related to the cellular networks. The different types of information that are available in the network log are listed as follows.

- Timestamp of the event,
- IMSI (a unique ID associated with a cellular network),
- IMEI (a unique ID identifying a mobile phone),
- Call type (code for voice call, SMS, etc.),
- Cell ID (ID of the cellular tower recording this information),
- Subscriber phone number,
- Latitude (geographical coordinate of the tower),
- Longitude (geographical coordinate of the tower).

Create network log table

```

Hive> create table network_log(
  timestamp string,
  IMSI string,
  IMEI string,
  call_type string,
  cell_id int,
  subscriber string,
  latitude string,
  longitude string
) row format delimited fields
terminated by ',';
OK

```

Time taken: 0.13 seconds

Load the data into network log table

The table created for the network log record earlier is loaded with the data using the following command. Some of the rows of the data inserted into the table are listed as follows.

```
Hive> load data inpath
'/user/training/network_log.txt' into table network_log;
Loading data to table learning.network_log
OK
Time taken: 0.156 seconds
```

Sample data for the table

```
(‘07:09:42’,‘410072821393801’,‘123456789123401’,‘sms’,‘1’,‘9886598801’,
‘66.5 N’,‘56.4S’),
(‘17:09:42’,‘410072821393801’,‘123456789123402’,‘sms’,‘1’,‘9886598802’,
‘66.5 N’,‘56.4S’),
(‘17:09:42’,‘410072821393801’,‘123456789123403’,‘sms’,‘1’,‘9886598803’,
‘66.5 N’,‘56.4S’),
(‘17:09:42’,‘410072821393801’,‘123456789123404’,‘sms’,‘1’,‘9886598804’,
‘66.5 N’,‘56.4S’),
(‘17:09:42’,‘410072821393801’,‘123456789123405’,‘sms’,‘1’,‘9886598805’,
‘66.5 N’,‘56.4S’),
```

Analysis on network log record table

The data inserted into the network log table can be analyzed to see some of the information on the network log data records. In this section, some of the HiveQL queries are executed on the call network log data record table to check for the information like cellular id, IMEI, and IMSI numbers in a particular region.

Query1: IMEI numbers that belong to a particular region

The following query can be used to list the IMEI numbers that belong to a particular region. This type of analysis is used in knowing the demand for the number of subscribers in a particular region. The IMEI numbers in the data can be used to determine the subscribers in the particular region.

```
Hive> select IMEI from network_log where latitude=‘46.5 N’ and
longitude=‘55.4S’;

      IMEI
123456789123426
123456789123427
123456789123428
123456789123430
123456789123441
123456789123442
123456789123444
123456789123470
```

Query 2: IMEI numbers for the call type voice and a particular cell id

The following query can be used to select the IMEI numbers of a particular network that belong to a cell id and the call type as voice. This type of analysis can be used for determining the number of subscribers for a particular network.

```
Hive>select IMEI from network_log where call_type='voice' and cell_id='5';

      IMEI
123456789123416
123456789123417
123456789123426
123456789123427
```

Query 3: IMSI for the particular region

The following query can be used to select a set of IMSI numbers for a particular region.

```
Hive>select IMSI from network_log where latitude='46.5 N' and
longitude='55.4S';
```

```
      IMSI
410072821393822
410072821393822
410072821393822
410072821393822
410072821393841
410072821393841
410072821393841
410072821393852
```

Query 4: Subscriber phone numbers that belong to a particular region

The following query determines the number of subscribers that belong to a particular region.

```
Hive>select subscriber from network_log where latitude='46.5 N' and
longitude='55.4S';

subscriber
9886598826
9886598827
9886598828
```

```
9886598830
9886598841
9886598842
9886598844
9886598870
```

Query 5: Cell ID and the subscriber phone numbers that fall into a particular latitude and longitude

The following query determines the number of subscribers in a particular region.

```
Hive>select cell_id,subscriber from network_log
where latitude='46.5 N' and
      longitude='55.4S';
```

cell_id	subscriber
5	9886598827
1	9886598828
1	9886598830
2	9886598841
2	9886598842
4	9886598844
4	9886598870

3.8 Exercises

1. Explain the workflow of Hive with Hadoop.
2. Differentiated between the Hive operations and SQL operations with an example.
3. Explain the difference of an external table and internal table in Hive with an example.
4. Explain the different data types in Hive.
5. For the network log analytics, write a query to select the subscriber id for the call type as sms.
6. Create a table by name employees with the following fields.

EMP_ID	EMP_NAME	DEPT	SAL
1023	Raj	Sales	25,000
1034	Ramesh	Marketing	30,000

Execute the following queries.

- i. Insert at least six rows into the table.
 - ii. Find all the employee names beginning with S
7. Create a Hive table by name Movies with different columns as shown in the table and write HiveQL for the following.
- (i) Display the movies that have rating above “9”.
 - (ii) Determine the number of movies an actor “A” has acted from the table.
 - (iii) Determine the number of movies directed by a director “A”.

Name	Rating	Actor	Actress
Lord of the Rings	9.8	A	GS
Batman	9.7	B	DS
Superman	8.6	A	FS

References

1. <https://hive.apache.org/>.
2. Du, D. (2015). *Apache Hive essentials*. Packt Publishing Ltd.
3. Capriolo, E., Wampler, D., & Rutherglen, J. (2012). *Programming Hive: Data warehouse and query language for Hadoop*. O'Reilly Media, Inc.
4. Chen, Y., Qin, X., Bian, H., Chen, J., Dong, Z., Du, X., et al. (2014, March). A study of SQL-on-hadoop systems. In *Workshop on big data benchmarks, performance optimization, and emerging hardware* (pp. 154–166). Cham: Springer.
5. Edward, C., Dean, W., & Jason, R. (2012). *Programming hive*.
6. Lin, X., Wang, P., & Wu, B. (2013, November). Log analysis in cloud computing environment with Hadoop and Spark. In *5th IEEE International Conference on Broadband Network & Multimedia Technology (IC-BNMT)* (pp. 273–276).

Chapter 4

Apache Spark



4.1 Introduction

With the Big data computing, Hadoop and Hive supported large processing of data. But, the data sources started to generate data that required more processing capability. For example, Twitter feeds, sensor data needed to be collected in near real time and process it as fast as possible. Even though Hadoop and Hive were capable of processing such data, it was batch-oriented. The MapReduce jobs submitted to the Hadoop was batch-oriented where the data was processed in batches in Hadoop. Apache Spark helps in solving the batch-processing problem of data analysis.

Apache Spark is built on top of Hadoop, and thus it uses HDFS as the file system for storing the files [1]. However, there are major differences between Hadoop and Spark. The major difference is Hadoop stores the data on disk to run analytics, whereas Spark uses in-memory caching mechanism for storing the data and processing. Here, the caching mechanism is referred to as resilient distributed datasets (RDDs) that reduces the network I/O and thus enable faster data processing.

Apache Spark is an in-memory data analytics system based on distributed processing and supports different programming languages like Python, R, scala, and Java. It was initially under BSD (Berkeley software distribution) license and later it was open sourced under Apache license. Spark provides four submodules MLlib, GraphX, SQL, and Streaming.

4.2 Working of Spark Application

A Spark application contains the components driver, cluster manager, executor, master, and the worker nodes. Each of these components assigned a specific task that is needed for computation functions during the runtime of the application [2]. The different components of the Spark application are as shown in Fig. 4.1.

Driver: A client submits the Spark application to the driver that manages the start and finish of the application. It is responsible for planning the execution of the application by coordinating with the worker nodes in the cluster. When the program is completed, the results are returned by the driver to the client.

- A Spark context is created by the driver when the application is submitted to the master. It is represented as sc in the programs that specifies the context of connection between the master and workers. In the beginning, the initiation of the context is created and used for the entire program.
- Another function of driver is to create a plan for execution of the application using a directed acyclic graph (DAG). The driver takes the input from the client side and if necessary the data transformations are made first before running the application. DAGs are composed of stages and tasks where task representing the smallest unit of work to be done. Stages comprise of different types of tasks that are dependent on each other.
- The driver also keeps track of the applications to be run on the cluster by maintaining the log of available resources and coordinate the movement of data between the stages in DAG.

The driver of the Spark cluster can be accessed through the port 4040 as shown in Fig. 4.2. It keeps track of the active jobs in the cluster and the stages present in it.

Workers and executors: These are the processes where the tasks of the DAG are run. A dedicated memory is reserved for workers and executors in the cluster. Workers run the executors on the slave nodes and the results are returned back once the program is completed. A Spark executor can run a number of tasks specified in the Spark program. A Web portal through the port 8081 for worker can be accessed as shown in Fig. 4.3.

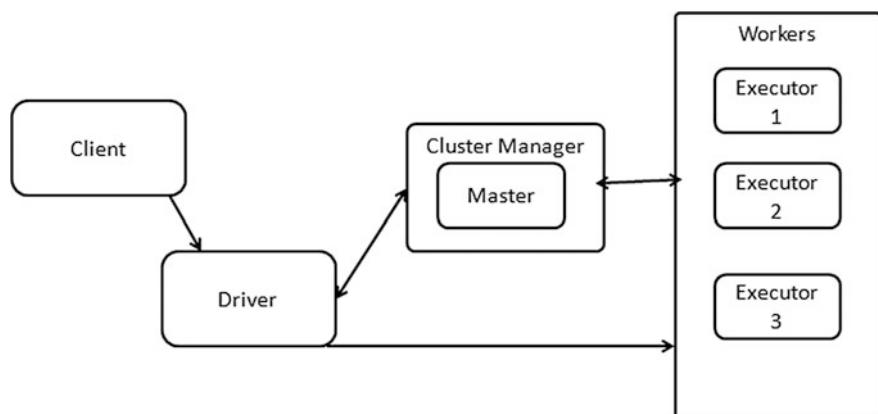


Fig. 4.1 Working of Spark

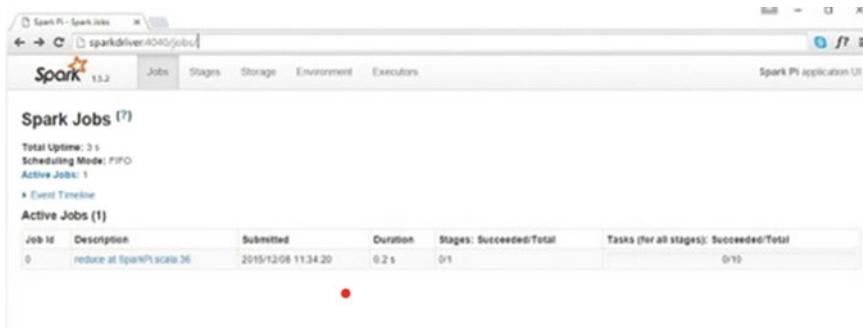


Fig. 4.2 Spark application UI through driver

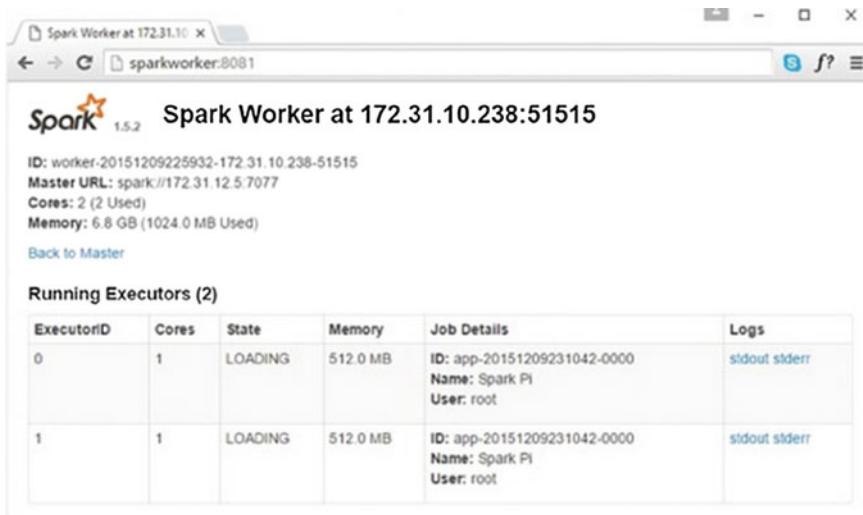


Fig. 4.3 Spark executor

Spark Master: It is the master process that first requests the cluster manager for the resources required to run the application in cluster. It is responsible for negotiating the resources with the workers and executors and monitors the status of them. It can also be accessed through the port 8080 as shown in Fig. 4.4.

Cluster manager: It manages the cluster by reserving the resources on the worker nodes needed for execution of application in the cluster. It monitors the request from the master and in turn makes sure that the resources are made available for the workers and executors.

The screenshot shows the Spark Master web interface at `spark://172.31.12.5:7077`. It displays the following information:

- Cluster Summary:**
 - URL: `spark://172.31.12.5:7077`
 - REST URL: `spark://172.31.12.5:6066 (cluster mode)`
 - Alive Workers: 1
 - Cores in use: 2 Total, 0 Used
 - Memory in use: 6.8 GB Total, 0.0 B Used
 - Applications: 0 Running, 1 Completed
 - Drivers: 0 Running, 0 Completed
 - Status: ALIVE
- Workers:**

Worker Id	Address	State	Cores	Memory
worker-20151209225932-172.31.10.238-51515	172.31.10.238.51515	ALIVE	2 (0 Used)	6.8 GB (0.0 B Used)
- Running Applications:**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20151209231042-0000	Spark PI	2	512.0 MB	2015/12/09 23:10:42	ubuntu	FINISHED	8 s
- Completed Applications:**

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
app-20151209231042-0000	Spark PI	2	512.0 MB	2015/12/09 23:10:42	ubuntu	FINISHED	8 s

Fig. 4.4 Spark Master

4.3 Installing Spark

Spark can be installed in three modes, namely *standalone*, *with Hadoop* and *Mesos*. In the *standalone* mode, Spark is installed without Hadoop. In *mesos* mode, the Spark can be installed on a cluster with two or more nodes. In this section, the installation part is described with *Hadoop*.

Installation of Spark with Hadoop

Spark can be installed and deployed in cross-platforms like Linux, Windows, and Mac OS. The prerequisites for Spark installation are:

- 4–8 GB RAM
- Java
- Python for using PySpark

The following steps are followed for the installation of Spark with Hadoop:

- Download the Spark distribution using `wget` or `curl` from <https://spark.apache.org/downloads.html>
- If Java is not installed use the following commands.
`sudo-apt get install openjdk-7-jre`
`sudo-apt get install openjdk-7-jdk`
- The java installation can be confirmed as follows.
`$java—version`
`java version '1.7.0_67'`
`Java(TM) SE Runtime Environment (build 1.7.0_67-b01)`
`Java HotSpot(TM) 64-Bit Server VM (build 24.65-b04, mixed mode)`

- Extract the Spark file downloaded to the preferred directory.
export SPARK_HOME = .../spark
export PATH = \$SPARK_HOME/bin:\$PATH
 - The installation of the Spark can be verified using the command *pyspark* in the terminal as follows.

```
Welcome to
      / \ / \
     /   \ - \ \ - / \ / \
    /_ \ / . \ \ / / \ \ \ \
   / \ / \ / \ \ \ \
Using Python version 2.7.5 (default, Feb 11 2014 07:46:25)
SparkContext available as sc, HiveContext available as
sqlContext.
>>>
```

- The pi program estimator can be executed as follows for successful installation of the Spark as follows.

```
spark-submit --class org.apache.spark.examples.SparkPi \
--master local \
$SPARK_HOME/lib/spark-examples*.jar 10
```

Pi is roughly 3.140576

4.4 Resilient Distributed Datasets

A resilient distributed dataset (RDD) is a collection of objects that are partitioned across a set of machines with read-only permission. The elements in RDD do not exist in a physical storage but a handle is used to access the information related to RDD [3]. This means that RDDs can always be reconstructed if nodes fail. In Spark, each RDD is represented by a Scala object. An RDD can be constructed using the following ways in Spark:

- A shared file system such as HDFS can be used.
 - A collection of data can be created in scala and sending it to the multiple nodes for slicing of data.
 - RDD can be created with the existing ones as well. A dataset that is created of element type A can be transformed into element type B using the operation flatmap. The filter functions can also be used to pick the elements from specific type to another one.

- RDD created are discarded from the memory after the use of it. These datasets are lazy and ephemeral. The dataset that is divided into partitions is put into parallel operation and discarded after it is used. It is accomplished through two actions, namely cache action and save action.
 - If the cache action is initiated on the RDD, then it is kept in the memory after the first time it is computed since it will be reused.
 - If the save action is initiated on RDD, then it evaluates the dataset first and then it writes to the HDFS.

During the cache operation, if there is not enough memory then the Spark will compute once again and then use the dataset. This is the advantage of having cache mechanism in Spark where the RDD is again used.

The other levels of persistence such as in-memory application can also be used with the Spark for RDD. The main aim of Spark is to manage the trade-offs between storage of RDD, probability of losing it, and the speed of accessing it.

4.5 RDD Representation and Job Scheduling

The main aim of developing Spark is to support a range of transformations on RDDs with no modifications to the schedulers in the Spark. Each RDD has a set of partitions and its dependencies. Partitions act as atomic pieces of the datasets and functions are written for partitioning scheme and data placement of the RDDs [4]. For example, if an RDD is stored in HDFS the partitions and the blocks of the file are managed by the HDFS alone.

The dependencies in RDD are designed using two methods, namely narrow dependencies and wide dependencies. In narrow dependencies, each partition of the child RDD is dependent on the parent RDD that is constant in number and size. In the case of wide dependencies, the child RDD depends on all the partitions of the parent RDD [5]. The example for the narrow dependency is the map function and the example for the wide dependency is the join function. Pipelined execution on one cluster node is possible in the case of narrow dependencies where the computation can be performed on all the partitions of the parent node. In the case of wide dependencies, since the data from the parent partitions are available MapReduce operations can be applied to the computation.

The fault tolerant architecture for the recovery of the node failure is efficient in the case of narrow dependency than the wide dependencies. The data is recomputed in narrow dependencies, whereas there might be a data loss for the wide dependency case.

4.6 Spark Examples

In this section, the different examples of programming with Spark are discussed.

4.6.1 Word Count with Caching

The following code demonstrates the example on word count in Spark. The random text file is considered for the program is as shown below.

'The Kawasaki Ninja H2 is a 'supercharged supersport' class motorcycle in the Ninja sportbike series, manufactured by Kawasaki Heavy Industries, featuring a variable-speed centrifugal-type supercharger. The track-only variant is called Ninja H2R and produces maximum of 310 horsepower (230 kW) and 326 horsepower (243 kW) with ram air, and it is the most powerful and fastest production motorcycle on the market. The H2R has 50% more power than the fastest street-legal motorcycles, while the street-legal Ninja H2 has a lower power output of 200 hp (150 kW) 210 hp (160 kW) with ram air.'

Its namesake is the 750 cc Kawasaki H2 Mach IV, an inline triple that was introduced by Kawasaki in 1972 to disrupt what it saw as a sleeping motorcycle market'.

In the program, first a Spark context is created initially. The cache() method is used for enhancing the lookup speed of the file for word count. A map function is used to get the words in the file where for each character; the value of 1 is initialized. The reduce lambda function takes the character each time and adds the value to get the total word count of the file considered.

```
from pyspark.context import SparkContext
from pyspark.sql import SQLContext

sc=SparkContext()
spark=SQLContext(sc)
#load the dataset
data=sc.textFile("text_file.txt")
data.cache() #caches the read text file for better lookup speeds
#counting the number of occurrences of each characters
count=data.flatMap(lambda x : list(x)).map(lambda character:
(character, 1)).reduceByKey(lambda a, b: a + b)
print(count.collect())
```

Output

```
[('i', 35),
('o', 9),
('p', 20),
('d', 13),
('g', 7),
('l', 20),
('t', 5),
```

```
(‘c’, 21),
(““, 4),
(‘y’, 9),
(‘W’, 4),
(‘4’, 1),
(‘R’, 2),
(‘b’, 4),
(‘h’, 24),
(‘j’, 4),
(‘N’, 4),
(‘r’, 42),
(‘9’, 1),
(‘s’, 40),
(‘K’, 4),
(‘.’, 4)]
```

4.6.2 Estimation of Pi

The following code demonstrates the program on estimation of Pi in Spark. A sample of 20,000 random numbers are initialized first, and then the filter() function is used with parallelization. In this way, Spark can be used for advanced computational problems that require more iteration.

```
from pyspark.context import SparkContext
import random
sc=SparkContext()

def func(p):
    x, y = random.random(), random.random()
    if x**2 + y**2 < 1:
        return 1
    else :
        return 0
samples=20000
df = sc.parallelize(range(0, samples)).filter(func)
count=df.count()
print("Pi is around ",4.0 * count / samples)
```

Output

Pi is around 3.1492

4.6.3 Filter Log File

When a log file is generated by the database or any other software, errors can be present in the log file. It takes a certain amount of time to filter out the log file and see the exact error in the log file. Suppose if the log file is large in volume, then it becomes hard to filter out the error. In this section, log file analysis is carried out using Spark. The log file considered for the analysis is as shown. The file has some error lines that need to be identified for analysis.

A Spark context initializes the program with the required configuration. The input to the program is converted to a data frame first with the help of DF function. The parameter passed to the DF() is *line*. For each line, the regular expression %*error%* is matched to see if the line contains the error message in it or not. The program outputs the total number of lines with error message in the log file. In this way, a large log file can be analyzed to filter out the error lines in the log.

```
from pyspark.context import SparkContext
from pyspark.sql import SQLContext, Row
sc=SparkContext()
spark=SQLContext(sc)
#load the data(the log file)
data=sc.textFile("log_file.txt")
#create a single column called line where each row refers to the lines in the log file
df = data.map(lambda r: Row(r)).toDF(["line"])
#filter out the lines which do not have the word error in them
df = df.filter("line like '%error%'") # line is the column name and %error% matches the substring error just like an sql query
print("Number of errors = ",df.count())
print("The log file related to the errors are \n",df.collect())
```

Output

```
Number of errors = 2
The log file related to the errors are
[Row(line='py4j.protocol.Py4JJavaError: An error occurred
while calling o43.avg.'), Row(line=':
org.apache.spark.sql.AnalysisException: error resolving
'product' given input columns: [Product, cost];')] 
```

4.6.4 Retail Analytics Revisited in Spark

In the previous chapters, retail analytics case study was discussed with MapReduce in Hadoop. The same case study is revisited in this section with Spark as the platform. The data frame is created initially using the map function where only *products* and *cost* values are extracted from the dataset. A groupby() method is used to get the average of each product in the dataset. In this way, the MapReduce functions can be combined into one in Spark for retail analytics.

```
from pyspark.context import SparkContext
from pyspark.sql import SQLContext

sc=SparkContext()
Spark=SQLContext(sc)
#load the dataset
data=sc.textFile("sampled_purchases.txt")

#choosing just the columns with the product names and its cost
reviews = Spark.createDataFrame(data.map(lambda x: tuple([x.split(",") [4],float(x.split(",") [5])])),[“Product”, “cost”])

#use of group by to calculate the average cost of a product
product_averages=reviews.groupBy(“Product”).avg().collect()
print(product_averages)
```

Output file

```
(Product='Books', avg(cost)=227.798),
(Product='Women's Clothing', avg(cost)=283.17555555555555),
(Product='Children's Clothing', avg(cost)=204.636875),
(Product='Sporting Goods', avg(cost)=270.13374999999996),
(Product='Consumer Electronics', avg(cost)=184.445),
(Product='Music', avg(cost)=334.57714285714286),
(Product='Toys', avg(cost)=215.1708333333335),
(Product='Baby', avg(cost)=217.052222222222),
(Product='Video Games', avg(cost)=227.00466666666665),
(Product='Cameras', avg(cost)=255.6143749999997),
(Product='Crafts', avg(cost)=305.5055555555556),
(Product='Garden', avg(cost)=261.77),
(Product='Men's Clothing', avg(cost)=264.2476923076923),
(Product='DVDs', avg(cost)=308.5591666666667),
(Product='Pet Supplies', avg(cost)=239.10375),
```

(Product='Computers', avg(cost)=259.85636363636365),
(Product='Health and Beauty', avg(cost)=248.8518181818182),
(Product='CDs', avg(cost)=211.8458333333333)

4.7 Exercises

1. What do you mean by RDD? Explain the significance of it in Spark.
2. Compare Hadoop and Spark with differences.
3. What is caching in Spark? Explain how filter operation works in Spark.
4. Describe the advantages of Spark over MapReduce.
5. For the retail analytics case study in the chapter, execute the following in Spark.
 - (i) Total profit made by each store location.
 - (ii) Total number of transactions that fall under type 'Amex'.
6. Consider a log file in the following format and execute the following in Spark.
 - (i) Display the number of lines that contains only 5 fields.
 - (ii) Display the number of lines that contain 5–6 fields.

Input

12-03-2017,User1,34,78,98
14-05-2018,user2,33,55,66,44,66
12-02-2018,user3,22,36,27,12,23
7. Consider the network log case study in Hadoop and repeat the same using Spark.

References

1. Spark, A. (2016). *Apache Spark: Lightning-fast cluster computing*. URL: <http://spark.apache.org>.
2. Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., et al. (2016). Apache Spark: A unified engine for big data processing. *Communications of the ACM*, 59(11), 56–65.
3. Shanahan, J. G., & Dai, L. (2015, August). Large scale distributed data science using Apache Spark. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 2323–2324). ACM.
4. Frampton, M. (2015). *Mastering Apache Spark*. Birmingham: Packt Publishing Ltd.
5. Karau, H. (2013). *Fast data processing with Spark*. Birmingham: Packt Publishing Ltd.

Chapter 5

Pig



5.1 Introduction

Since there is a growing need for analysis of large datasets, the procedural programming like SQL seems to be unnatural. MapReduce programming is rigid, and the customer user code is hard to reuse and maintain. Apache Pig is used between declarative style SQL and procedural programming. It uses Pig Latin as the language for analysis. The jobs written in Pig Latin are translated into MapReduce jobs first and executed on distributed HDFS for processing large data in real time.

Pig is a high-level scripting language developed by Yahoo, used with Apache Hadoop who's SQL like scripting language is called Pig Latin. The code written using Pig Latin is named Pig Latin script. It is an Apache open-source project which means users can download, use, and contribute for it. Pig runs on Hadoop and uses both Hadoop Distributed File System (HDFS) and Hadoop's Processing System.

MapReduce on Hadoop was too low-level and rigid, hard to maintain and reuse, and to overcome this drawback, a new language was introduced called Pig Latin. It is a dataflow which allows users to describe how data from one or more input can be processed to one or more output in parallel. There are no 'if' statements or 'for' loops in Pig Latin.

Directed Acyclic Graph (DAG) are described in Pig Latin script in which nodes are considered as operators for processing data and edges are dataflows. Dataflow may be a simple or a complex workflow. A set of transformations such as aggregate and sort are defined by Pig Latin on dataset, user-defined functions written in java or Pig Latin script are used and directly called from Pig Latin which is translated into MapReduce, and data are executable on Hadoop. To offer more control over Pig script, we can embed Pig Latin in Java, Python, and JavaScript scripting languages.

5.2 Installing Pig

Apache Pig supports cross-platform installation and working. In this section, the installation steps for Pig are shown for the Ubuntu.

- Download the Pig using the following command in the terminal.
- wget <http://www-us.apache.org/dist/pig/pig-0.16.0/pig-0.16.0.tar.gz>
- Extract the tar file into a folder.
- Edit the environment to add Pig to the bash using the following commands.
`export PIG_HOME = /.../pig-0.16.0`
`export PATH = $PATH:/.../pig-0.16.0/bin`
`export PIG_CLASSPATH = $HADOOP_CONF_DIR`

5.3 Architecture of Pig in Hadoop

Pig is made of two components:

- It includes the programming language for the Pig which is called Pig Latin.
- The Pig Latin code is converted to an executable code by the Pig Latin compiler.

The executable code is in a MapReduce job, or a process is spawned where virtual Hadoop instance is created to run the Pig as shown in the Fig. 5.1. Data processing and analysis of Pig program are done in parallel which is enabled by sequence of MapReduce programs [1]. Useful strategy for testing Pig script is running the Pig job in the virtual Hadoop instance. Pig programs can run on MapReduce without doing any changes in the code. Tez API can also be used to run the Pig scripts. YARN is used for more efficient execution than MapReduce.

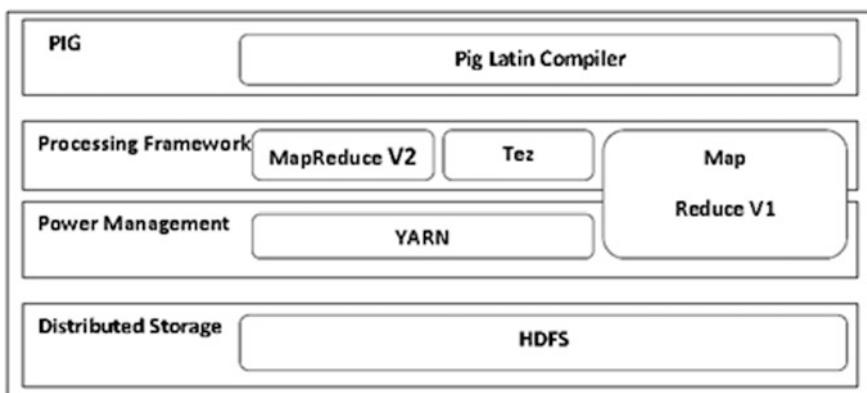


Fig. 5.1 Architecture of Pig

5.4 Modes and Working of Pig

The interaction with Pig differs from Hadoop and other systems related to it. There are three modes of user interaction in Pig [2].

- Interactive Mode
- Batch Mode
- Embedded Mode

Interactive Mode

An interactive shell called Grunt is provided to the user which accepts the Pig commands. STORE command is used when the user asks for output for compilation and execution.

Batch Mode

A script containing series of Pig commands is submitted by the user and then ending with the STORE command.

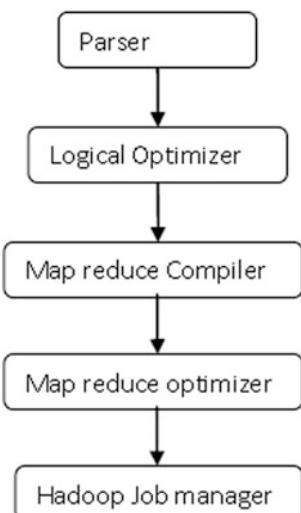
Embedded Mode

Pig Latin commands can be submitted via method invocation from a Java, Python program. This provides dynamic control flow of Pig Latin.

With these three modes of execution, a series of steps are executed for transformation of the data as shown in the Fig. 5.2.

- First step is parsing in which the parser verifies the correct syntax of program, and type checking of program is done. The output of parser is logical plan with one-to-one correspondence between Pig Latin statements and logical operators in the form of DAG.

Fig. 5.2 Execution stages in Pig



- The DAG is then passed to logical optimizer which is then compiled into a series of MapReduce jobs and then passed to another optimization phase.
- The optimized MapReduce jobs DAG is sorted and submitted to Hadoop for execution. Hadoop execution status is monitored by Pig and reports the overall progress of the program to the user. Hence, any warnings or errors during execution are reported to the user by the Pig.

Running Pig can be done in following ways:

- **Running Pig Locally On Machine:** Running Pig locally on machine is called local mode. Prototyping and debugging Pig Latin scripts are done in local mode. It is also used for small data applied when same processing of large data.
- **Running Pig On Hadoop Cluster:** The location of cluster's NameNode and JobTracker is the only thing the Pig needs to know to run on cluster. NameNode is manager of HDFS, and coordination of MapReduce jobs is done by JobTracker. Maintaining and setting up a Hadoop cluster are expensive in terms of hardware, costs, and administration.
- **Running Pig In The Cloud:** Cloud offering of Amazon's Elastic MapReduce (EMR) is different that is rather than allowing customers to rent machine for any type of process EMR allows users to rent virtual Hadoop cluster. Data are read and written to Amazon's simple storage service (s3) by these clusters. Rented Hadoop cluster can be accessed by EMR users via browsers or a Web services API.

5.5 Dataflow Language

In Pig Latin, a sequence of steps is specified by the user where each step is a single transformation. Pig Latin is preferred by the experienced programmers who use large datasets. In Pig Latin, it is not necessary that the operations must be executed in defined sequence order.

Consider an example,

```
spam_urls = FILTER urls BY isSpam(url);  
culprit_urls = FILTER spam_urls BY pagerank > 0.8;
```

In the above Pig Latin script, isSpam is used first to find the spam urls and then filtered by pagerank whose result is contained in culprit_urls. This may not be an efficient method. To filter the pagerank first and then execute isSpam later only on those having high pagerank would be an efficient method which is available using Pig Latin.

5.6 Pig Latin: Functions

Here, we are going to discuss some of the functions of Pig Latin [3–5] using the example of album dataset, and dataset contains the details of user, album id, album name, track id, and track name. The symbols <> used to denote space.

Load

It is used to load the input file or datasets.

Syntax: LOAD <file name or location> using PigStorage AS <field name: type>;

Example:

```
Songs = LOAD „/userid-track.tsv“ AS (userid:int, time:long, albumid:int, albumname:chararray, trackid:int, trackname:chararray); //load input file named userid-track.tsv
```

Loading can be done in following alternative formats

- By default, loading data as tab separated as text files.

Songs = LOAD „lastfm/songs.tsv“;

Songs = LOAD „lastfm/.csv“ USING PigStorage(„, “);*

- Schema can be specified.

Songs = LOAD „lastfm/songs.tsv“ AS

(user: chararray, time, album, track: chararray);

- Data from HBase table can be loaded.

Songs = LOAD „ hbase ://SongsTable“ USING HBaseStorage();

- A custom load user-defined function (UDF) can be used.

Songs = LOAD „lastfm“ USING MyCustomlaodUDF();

Duplication

DISTINCT removes duplicate records. It works only on the entire record, not on individual fields. It can be nested with foreach.

Syntax:

DISTINCT <relation name>;

Example:

Unique = DISTINCT songs; //remove all duplicate tuples in songs data model.

Grouping

GROUP collects together records with the same key. It produces records with two fields: the key and the bag of collected records.

Syntax:

```
GROUP <name> BY <field>;
```

Example:

Grouped = GROUP Unique BY albumname; //it group file by album name

It supports an expression or user-defined function (UDF) as group key and also grouping on multiple keys.

Projection

FOREACH...GENERATE: takes a set of expression and applies them to every record.

Syntax:

```
FOREACH <name> GENERATE <field name>;
```

Example:

Projected = FOREACH songs GENERATE albumname, trackname; // create album name and track name

In this, field is referenced by name or by position (start from 0). All fields(*) or a range of field can be referenced (...). A custom UDF can be invoked.

Illustrate

Displaying step-by-step execution of a sequence of statements.

Syntax:

```
ILLUSTRATE <name>;
```

Example:

ILLUSTRATE Projected;

Aggregation

It used for Build-in functions like AVG, COUNT, COUNT_STAR, MAX, MIN, SUM.

Syntax:

```
FOREACH <relation name> GENERATE <name>, <function name> AS <name>;
```

Example:

Counted = FOREACH Grouped GENERATE group, COUNT(Unique) AS CNT;
//count the number of given field

Storing Data

STORE saves the results. A directory with part files is created while storing to the file system rather than single file.

Syntax:

```
STORE <relation name> INTO <output file name>;
```

Example:

```
STORE Counted INTO „songs_per_albums“; //store the result in song_per_album file.
```

Describe

It uses to returns the schema of a particular relation.

Syntax:

```
DESCRIBE <name>;
```

Example:

```
projected = FOREACH songs GENERATE albumname;
```

```
Grouped = GROUP projected BY albumname;
```

```
DUMP Grouped; //display details of grouped relation
```

5.10Joining

JOIN: joins two or more sets.**Syntax:**

```
JOIN <relation-name> BY <name> BY <name> USING <joins name or type>;
```

Example:

In the given example, we are going to join two relation sets named Songsproj and Users. Details of SongsProj and Users tuple are as follows:

Songsproj.

(user_002, Travis, Sing)

(user_002, Travis, The Weight)

Users.

(user_2000, peru)

(user_3000,U.S)

Joined = JOIN Songsproj BY UserId, Users BY UserId USING „replicated“;

Output:

(user_002, Travis, Sing, user_2000, peru)

(user_002, Travis, The Weight, user_2000, peru)

Types of joining are:

- Merge join—Join key is used to pre-sort the sets.
- Merge-Sparse join—Sets are pre-sorted, and one set has few matching keys.
- Replicated join—One set is very large, while other sets are small enough to fit into memory.
- Skewed join—When a large number of records for some values of the join key is expected.
- Regular Join

Flattening

FLATTEN used to un-nest a tuple or a bag.

Syntax:

FLATTEN <tuple name>

Example: Counted = FOREACH Grouped GENERATE FLATTEN (grop), COUNT (Joined) AS Cont;

Ordering

ORDER sorts relation by one or more fields. Sorting by maps, tuples, or bags produces errors. NULL is taken to be smaller than any possible value for a given type.

Syntax:

ORDER <relation-name> B <condition: ascending(ASC) or descending order (DESC)>;

Example:

ORDER songs BY * ASC; //order songs in ascending order

Limiting Results

LIMIT returns or display limited number of results. It forces a reduce phase.

Syntax:

LIMIT <relation name> <tuples name>;

Example:

LIMIT songs 2; //only first two row will get displayed from songs.

SAMPLE

Use to generate a data based on specific condition and size.

Syntax:

SAMPLE <relation name> <size>;

Example:

Page | 14

Smple = SAMPLE users 0.2; //smple will contain 20% of users data.

SPLIT

Separate a relation into more relations.

Syntax:

SPLIT <relation-name1> INTO <relation-name2> IF <one or more expression>;

Example:

SPLIT song INTO song1 IF userid == 3, song2 IF albumid < 5; //relation song is split into two relation.

UNION

Reckon the union of multiple relations

Syntax:

UNION <more than two relation names>;

Example:

UNION songs, song1; //generate union of both fields

UNION ONSCHEME song, song1; //onscheme used to arrange tuples have fields in different orders.

CROSS

Compute the cross-products of more than two relations.

Syntax:

CROSS <two or more relations name>;

Example:

CROSS song, song1; //generate the products of two relation song and song1

COGROUP

Same as GROUP operation. It re-involves multiple relations.

Syntax:

COGROUP <relation-name1> <field-name1>, <relation-name2> <field-name2>;

Example:

COGROUP <song> BY userid, song1 BY albumid; //co-grouped using field ‘userid’ from relation song and field ‘albumid’ from relation song1

5.7 Exercises

1. Explain the workflow of Pig with Hadoop.
2. Compare Hive and Pig with an example.
3. Justify the statement ‘Pig automates the MapReduce programming.’
4. Write Pig Latin queries for the retail analytics case study considered in the Hadoop to determine

- a. Total profit made department-wise.
 - b. Total transactions of type amex.
5. Consider an employee dataset with the details such as empid, empname, job_title, dept, salary. Write a Pig query that gives,
- (i) Total number of employees in a department
 - (ii) The employees of a particular designation and department
 - (iii) List the employees who are working in department number 10–20.

References

1. Pig, A. (2016). The Apache Software Foundation.
2. Gates, A., & Dai, D. (2016). *Programming pig: Dataflow scripting with hadoop*. O'Reilly Media, Inc.
3. Olston, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008, June). PigLatin: a not-so-foreign language for data processing. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 1099–1110). ACM.
4. Ahuja, S. P., & Moore, B. (2013). State of big data analysis in the cloud. *Network and Communication Technologies*, 2(1), 62.
5. Gates, A. F., Natkovich, O., Chopra, S., Kamath, P., Narayananurthy, S. M., Olston, C., & Srivastava, U. (2009). Building a high-level dataflow system on top of Map-Reduce: The Pig experience. *Proceedings of the VLDB Endowment*, 2(2), 1414–1425.

Chapter 6

Apache Flume



6.1 Introduction

Consider a simple scenario of analyzing social networking sites like Facebook and Twitter. If Twitter is considered in particular there are millions of tweets tweeted per day, and analysis of these tweets is very useful in understanding the user behavior. If a person tweets on cricket, then his interest is toward sports, or if a person tweets about a movie, then the user likes entertainment. Thus, recording such type of behaviors may help in analyzing user patterns based on their interest, and thus, many companies try to make benefit out of it by highlight their products based on these user patterns.

Since each type of data is generated in real time, data have to be processed in few seconds as fast as possible [1, 2]. The data generated by these applications are large in volume and need to be handled with care. In order to support such systems, the data need to be stored in centralized data storage systems such as Hadoop for further analysis. Apache Flume is one of such tools which suits well for the scenarios where real-time analytics is needed. It is one of the most reliable and distributed software platforms for real-time analytics.

Apache Flume collects the data from the data generators (such as Web servers, cloud servers) and aggregates all such data and finally pushes the data at high speed in real time into the data store like HBase or HDFS. An abstract overview of Apache Flume is as shown in Fig. 6.1.

The features of Flume are highlighted as follows:

- It is based on Java platform and can be easily understood by the Java developers.
- Flume is one of the data ingestion mechanisms which collects the data from various sources and Web servers and stores such a log data into the available data store efficiently.

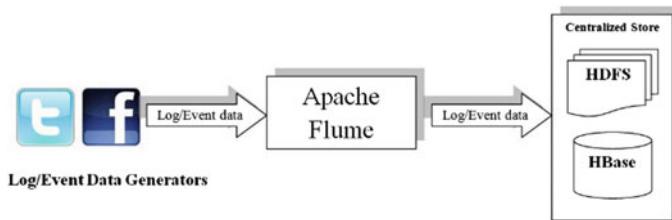


Fig. 6.1 Data processing with Flume

- The spontaneity of Apache Flume is well suited for real-time data analytics, and it is horizontally scalable.
- It supports different kinds of source and destination platforms. It is highly robust and fault-tolerant mechanism.
- At extreme conditions of failure, it supports failover and recovery mechanisms.

6.2 Background

The technological growth has lead to many different types of technologies for processing Big data. The data source for many of the analytics in Big data world is a log/event file that captures different types of events in the transaction. In this section, the drawbacks of the traditional data ingestion mechanisms are discussed that gave the motivation of building the Flume platform for analysis.

6.2.1 *Log File/Events*

Most of the e-commerce applications such as Flipkart, Amazon and social networking applications such as Facebook, Google, and Twitter accept multiple queries per day, millions of clicks per day within a short span of time. Such types of data are huge in volume and are stored as server log/click-streams. Here, click-streams refer to the events generated during a particular period of time [1, 3].

A log file is a file that may be a record with particular id and contains the data related to the particular id and stored as a csv file. For example, in email logs might contain the information like id, time-of-login, and it consists of the each login time recorded every time for the login. There may be many such ids in the csv file separated by comma. Analysis on such log files like, how many times a particular user signed with a id in a particular day, how long the user had signed in, the duration of breaks for a particular login id bring insights into the application use.

6.2.2 Log File Processing Using Hadoop and Its Drawbacks

A log file can be stored in HDFS using the *put* command. The *put* command can be used in the following way. It copies the file from the local file system specified by the *<local-input-path>* to the HDFS path *<hdfs-path>*. Even though HDFS supports the storage of large volumes of files, there are certain drawbacks that are highlighted as follows.

- A single file only can be transferred to the HDFS at a time.
- Hadoop is based on batch processing, and the support of real-time analysis is low and expensive.
- The time to transfer the file into HDFS increases with the size of the file.

Apache Flume is one of such platforms that can be used to overcome the drawbacks of Hadoop for real-time data analysis. However, Flume is also a part of Hadoop ecosystem and used HDFS internally for analysis. In the subsequent sections of this chapter, the architecture of Flume and its components, examples on it are discussed.

6.3 Flume Architecture

Apache Flume collects the data from different source and aggregates it together and performs analysis [1]. It involves major elements like data aggregation, source, sink, Flume agents, and other architectural elements as shown in Fig. 6.2. In this section, the architecture and the core elements of Flume are discussed.

Web server

The different types of log data like network logs, Web logs, and server logs are stored in the Web server. It acts as a source for data analysis in Apache Flume. These logs are aggregated together so that analysis can be done on individual logs if needed. Since the data need to be handled in real time, the data should be processed immediately for analysis.

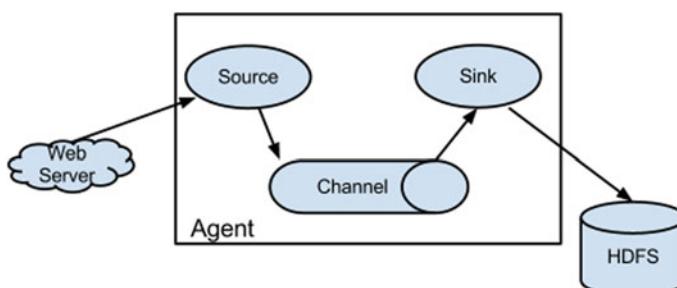


Fig. 6.2 Flume architecture

Flume Agent

An independent daemon process (JVM) in Flume is known as agent. It initiates other components of Flume such as sources, sinks, and channels and thus involved in receiving, storing, and passing the events to the final destination. Flume Agent consists of three major components:

Source

The data generated from the Web servers are received by source and transferred to one or more channels during the Flume event. The various types of sources supported by Apache Flume are utilized for different types of analysis. The implementation can be in such a way that each source can receive data from different specified data generators like Thrift, Avro, Twitter.

Sink

The data received from the Web servers are finally stored in the centralized stores such HBase and HDFS that act as the destinations for sink. The examples of sink are HBase sink, HDFS sink terminal sink, Kafka sink, Avro sink, etc.

Channel

It is a typical flow pipeline which can receive the events from the source and pass on to the neighbor sinks called as channels. They form a connecting link between the source and sinks. The count of source and sink may be any number like file system channel, memory channel.

Additional Components of Flume Agent

Other than the primitive components of the Flume agent, some additional components are needed for streaming analysis. These additional components are listed as follows.

Interceptors

In some cases, there is need for altering or inspecting the Flume events and such a task is done by the interceptors. If there is a problem in the flow of events, then the interceptors help in altering the event workflows.

Channel Selectors

There can be several channels for the transfer of the data, and a particular channel for the transfer has to be selected; this task is done by the channel selectors. Channel selectors are of two types, namely default selectors and multiplexing selectors.

- **Default selectors:** These selectors select such a channel which is best for the transfer based on the replication mechanism of Flume events.
- **Multiplexing selectors:** These selectors select a channel for the transfer of events based on the address present in header.

Processors

Sinks can be selected as a group for transferring the data to the required destination. The initiation of the request when made for the sink transfers the data. However, the status of the sink needs to be assessed for the transfer. In such cases, failover paths can be easily created using this Sink processors and support load balancing.

The architectural elements discussed in this section are embodied in a flow of events for log processing. The different steps in the log processing using Flume are discussed in the next section.

6.3.1 Log/Event Data Processing in Flume

The processing of events in the log is carried out using the different elements of Flume as shown in Fig. 6.3. The key component of the event processing is client. Client initiates the data processing and transfer of the data from source to sink. Depending on the application, there can be different sources and sinks.

The flow of events happens in Flume with the help of agents, and they pass on the data through several channels eventually. The channels receive the event and try to pass on them through neighboring sinks. In case of regular sink, the event gets forwarded to its neighboring destination which may be another agent.

If it is a terminal sink, then event reaches its final destination. The decoupling of sources from sinks is done using producer-consumer design pattern. Thus, each source and sinks differ in their performance characteristics based on the availability and functions needed by the application. There are two types of flow in Flume:

- **Fan-out flow:** If the flow is from one source through different channels to reach the final destination, then it is called fan-out flow.
- **Fan-in flow:** The data may flow from many sources through single channels to reach multiple destinations called as fan-in flow.

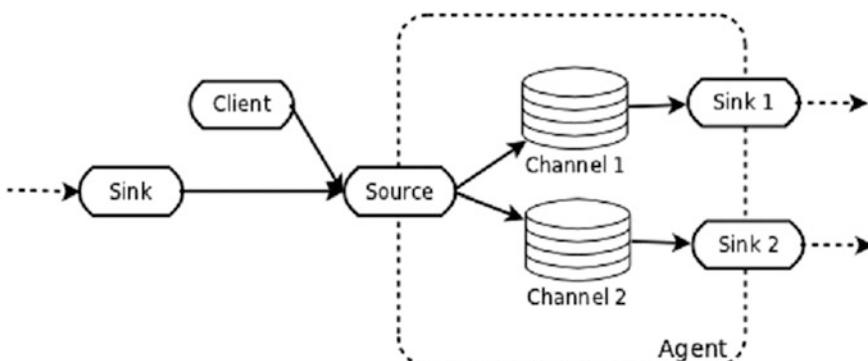


Fig. 6.3 Log/event processing in Flume

6.3.2 Failure and Event Handling in Flume

The processing of events in Flume should be reliable in message delivery and guaranteed service. In Flume, the transaction/event initiates when the message moves from one agent to another agent. The agent sends commits to the transaction/events when it receives the authentication from the received agent that it has successfully received the message. This mechanism ensures guaranteed message delivery. The sequence diagram in Fig. 6.4 shows the transaction commits between two agents in Flume.

There can be congestion between the neighboring agents and results in failure. This sort of failure results due to the miscommunication between the agents. Since real-time events need to be handled, the failures have to be overcome very quickly, or else they may end up affecting the other events running concurrently. This may lead to the unexceptional failure of event.

It can be resolved using the reporting back mechanism. When the client process the events, if the event fails at some stage, it reports back to the client and that event will be suspended for some time so that it should not affect other events. Simultaneously, there will be the processing of events normally and soon the failure is overcome that event is normally processed to reach the destination. Thus, the process runs on action and reaction scenario, and there is a constant communication between each event.

Figure 6.5 illustrates a case in which there is a data flow between the two neighboring agents and when the failure occurs, how it can be resolved. Figure 6.5a shows normal flow of event from client to centralized data store. Later in Fig. 6.5b, there is a miscommunication between Agent-2 and the data store which results in failure. In Fig. 6.5c, the reason behind the failure was discovered and the flow suspended was restarted and there was a normal flow of event from Agent-2 to data store.

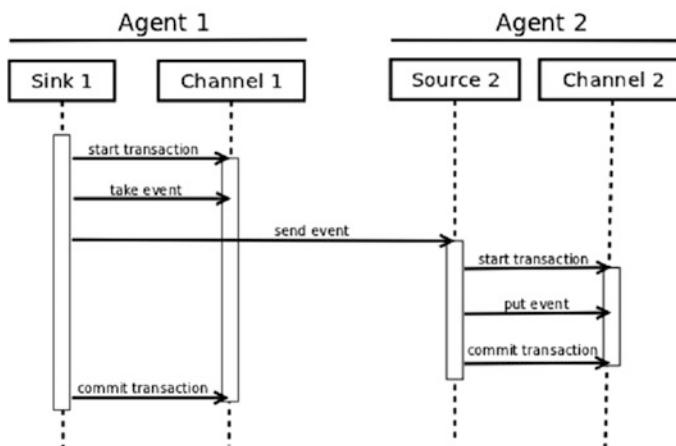


Fig. 6.4 Transaction commit between the two agents

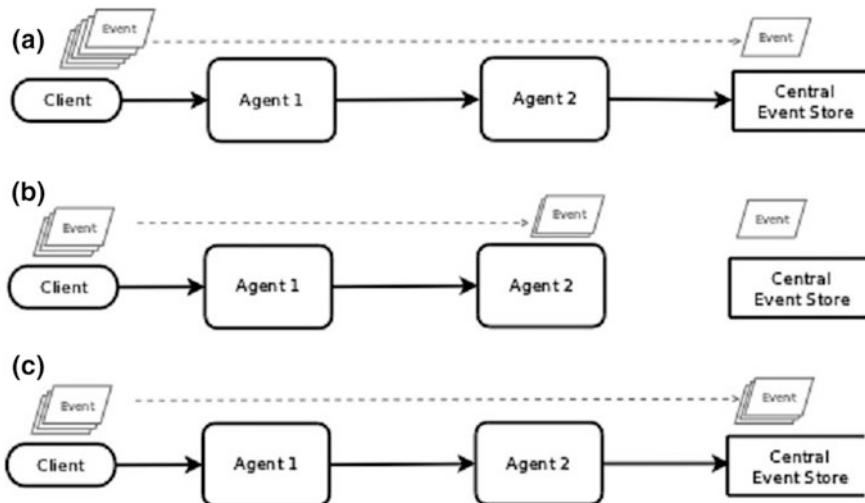


Fig. 6.5 Failure handling

In this way, the communication between the agents and sources/sinks is used for handling failures. In real-time processing of events, there can be different sources/sinks with agents communicating with each other. Depending on the network bandwidth, there can be a breakdown in the communication among the agents and source/sinks. In the event of breakdown, there should be no loss in the data. Hence, Apache Flume handles such failures using the reply back communication between the source/sink and the agents. It can be mapped to the heartbeat message in Hadoop between the data nodes and NameNodes in Hadoop.

6.4 Installing Apache Flume

Apache Flume is a cross-platform that can be configured in Ubuntu, Windows, and Linux platforms. In this section, the installation procedure of Apache Flume is discussed. The pre-requisites for the installation of Apache Flume are

- Java 1.7 or later
- 4–8 GB RAM

The procedure in the following steps is adopted for the Ubuntu platform.

- Download the Apache Flume from the site <https://flume.apache.org/download.html> and select the apache-flume-1.x.x-bin.tar.gz for the Ubuntu operating system.
- Extract the tar file into a folder.

- The system PATH variable is set to the Flume's bin folder using the following command.

export PATH = \$PATH: .../flume-1.x.x/bin

- The installation of Apache Flume can be verified using the command

flume-ng -help

6.5 Flume in Action

As seen in the earlier sections, the different architectural elements of the Apache Flume are used for implementing the Flume in action. In this section, Apache Flume is implemented with the required source, sink, and channels. While configuring the Apache Flume for collecting the real-time data, first the source, sinks are named, channel creation and then the Flume agent is created. The following sections describe in detail the implementations for running Flume in action [4].

6.5.1 Naming of Source, Sink, and Channels

In the first step, all the components source, sink, and channels have to be named based on the type of source, sink, and channels to be used. The different types of example for the sources, sinks, and channels are listed as follows.

- Sources:** Avro, Thrift, Exec, JMS, Twitter, Kafka, Syslog, NetCat Sequence Generator, Syslog TCP.
- Sinks:** HDFS, Hive, Logger, Avro, MorphlineSolr, Thrift, IRC, Null, HBase, AsyncHBase, ElasticSearch, Kite Dataset, Kafka, etc.
- Channels:** Memory Channel, JDBC, Kafka Channel, File, Channel, etc. Based on the requirements, you can use any.

It can be done with the following code.

```
AgentName.sources = SourceName
AgentName.sinks = SinkName
AgentName.channels = ChannelName
```

```
AgentName.sources =Avro
AgentName.sinks =HBase
AgentName.channels =JDBC
```

6.5.2 Defining Source

A source is a data store from which the data are generated and are transferred using the Flume workflow. The general syntax for defining the source is as shown below.

```
AgentName.sources.SourceName.type = "value"
AgentName.sources.SourceName.property one = "value"
AgentName.sources.SourceName.property two = "value"
.....
AgentName.sources.SourceName.property n = "value"
```

where,

- **AgentName** = Name of the Agent such as Twitter, Sequence generator, etc.
- **SourceName** = Type of source we are using such as Avro, Thrift, Exec, JMS, etc. This should always be defined first and it differs based on the type of source.
- **property** = There may be various kinds of properties based on the source we are using.
- **value** = any data type such as integer, character.

The following code demonstrates the source as Twitter in Apache Flume [5], where the customer key, access token, token secret, and keywords are given.

```
TwitterAgnt.sources.Twitter.type=
org.apache.flume.source.twitter
(Apache Flume 1.5.0 API)
TwitterAgnt.sources.TwitterAPI.customer Key = 111
TwitterAgnt.sources.TwitterAPI.customer Secret = 222
TwitterAgnt.sources.TwitterAPI.access Token = 333
TwitterAgnt.sources.TwitterAPI.access Token Secret = 444
TwitterAgnt.sources.TwitterAPI.keywords =
cassandra, Storm, mahout
```

6.5.3 Defining Sink

Sink is the data source that specifies the destination of the data flow in the Flume. The general syntax for defining the sink is as shown below.

```

AgentName.sinks.SinkName.type = "value"
-
-
-
AgentName.sinks.SinkName.propertyone = "value"
AgentName.sinks.SinkName.propertytwo = "value"
.....
AgentName.sinks.SinkName.propertyn = "value"

```

where,

- **AgentName** = Name of the Agent such as Twitter, Sequence generator.
- **SinkName** = Type of source we are using such as MorphlineSolr, Thrift, IRC. This should always be defined first, and it differs based on the type of sink.
- **property** = There may be various kinds of properties based on the Sink we are using.
- **value** = any data type such as integer, character.

The following code demonstrates the sinks for the twitter as the data source. Here, the sink considered is Hadoop file system with the type as streaming data and text file.

```

TwitterAgnt.sinks.HDFS.type = hdfs
TwitterAgnt.sinks.HDFS.hdfs.type of file =
streaming data
TwitterAgnt.sinks.HDFS.hdfs.file format = txt
TwitterAgnt.sinks.HDFS.hdfs.size of batch = 3063
TwitterAgnt.sinks.HDFS.hdfs.size of roll = 0
TwitterAgnt.sinks.HDFS.hdfs.count of roll = 12000

```

6.5.4 Defining Channel

The source and sinks for the flow of data a channel is needed for the communication. The generic syntax for defining the channel is as shown below.

```

AgentName.channels.ChannelName.type = "value"
AgentName.channels.ChannelName.property one = "value"
AgentName.channels.ChannelName.property two = "value"
.....
AgentName.channels.ChannelName.property n = "value"

```

where,

- **AgentName** = Name of the Agent such as Twitter, Sequence generator.
- **ChannelName** = Type of channel we are using such as JDBC, Kafka, File. This should always be defined first, and it differs based on the type of channel.
- **property** = There may be various kinds of properties based on the channel we are using.
- **value** = any data type such as integer, character.

The channel for the source and sinks defined in the earlier sections is defined as follows. The channel should consist of volume of the character and the transactions volume as shown in the code below.

```
TwitterAgnt.channels.MemoryCha.type =
Memory
TwitterAgnt.channels.MemoryCha.volume =
11456
TwitterAgnt.channels.MemoryCha.transaction volume = 120
```

6.5.5 Binding Source and Sink to Channel

The data sources and data sinks are bound to the channel using the following code. The general syntax for binding the source and sink to the channel is as shown below. The channel is bounded to both source name and sink name.

```
AgentName.sources.SourceName.channels = ChannelName
AgentName.sinks.SinkName.channels = ChannelName
```

For the Twitter example, considered in the previous sections, the channel is bounded with source and sinks as with the code as shown below.

```
TwitterAgnt.sources.Twitter.Channels = MemoryCha
TwitterAgnt.sinks.HDFS.Channels = MemoryCha
```

6.5.6 Flume Agent Start-up

Once the source and sink are configured with the channel, the Flume agent can be started up using the following command.

```
bin/flume-ng agent -c conf -f conf/flume-conf.properties.template -n  
<flume agent name>  
where
```

- **agent:** starts Flume agent.
- **-c:** refers to the configuration file.
- **-f:** If there is any missing path it configures that path.
- **-n:** It is the name of the agent.

6.5.7 Configuring Flume with Twitter

The following steps are used to configure the Flume with the Twitter data stream.

- Login to the Twitter Web site <https://twitter.com>
- After log in, go to the browser, type <https://apps.twitter.com> that will lead to twitter apps page and click on the option “Create New App”.
- Fill the ‘Application Details’ form by reading the agreement and click on ‘Create your Twitter application’. This creates your own Twitter application, and a new window appears.
- Go to the top level of the window to find ‘Keys and Access Tokens’ option, and click on it. It shows Access Key and Tokens.
- Under ‘Token Action,’ click on ‘Create my access token’ button. This creates you own access token.
- The data can be fetched from twitter by starting Flume Agent. Once the Flume start to stream the data for few seconds on hdfs, later break the command (ctrl +c) and stop streaming.
- Open the browser and go to the path where the Flume data was downloaded and click on the file. It can now be configured with Hive for exploring and analyzing the data.

6.6 Exercises

1. What is Apache Flume? What are its benefits and applications?
2. Discuss in detail about the architecture of Flume with a neat diagram.
3. Discuss how the Event Handling and Failure Handling are done in Flume.
4. Explain Source, Sink, and Channels with an examples for each.
5. Discuss in details how to fetch the Twitter data with the help of Flume.
6. Discuss in details how to fetch the data from Sequence Generator Source using Flume.
7. Discuss in details how to fetch the data from NetCat Source using Flume.

References

1. Liu, X., Iftikhar, N., & Xie, X. (2014, July). Survey of real-time processing systems for big data. In *Proceedings of the 18th International Database Engineering and Applications Symposium* (pp. 356–361). ACM.
2. Wang, C., Rayan, I. A. & Schwan, K. (2012). Faster, larger, easier: Reining realtime big data processing in cloud. In *Proceedings of the Posters and Demo Track* (p. 4). ACM.
3. Ranjan, R. (2014). Streaming big data processing in datacenter clouds. *IEEE Cloud Computing*, 1(1), 78–83.
4. Lin, J., & Kolcz, A. (2012). Large-scale machine learning at twitter. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 793–804). ACM.
5. Khuc, V. N., Shivade, C., Ramnath, R., & Ramanathan, J. (2012). Towards building large-scale distributed systems for twitter sentiment analysis. In *Proceedings of the 27th annual ACM symposium on applied computing* (pp. 459–464). ACM.

Chapter 7

Storm



7.1 Introduction

Social media data are used by the companies for understanding the consumer behavior and product analytics. For example, when a new product is launched by the company, the tweets in Twitter can be gathered to know the immediate feedback of the product. Most likely a java/Python program is used at the backend for collecting the tweets one-by-one, parse it and count the number of positives and negatives of it. A Web application can be framed around this that gives the overall analysis of the tweets as shown in Fig. 7.1.

The main drawback of such Web-based application is it might crash at any time because of large volume of the tweets coming in real time at the back-end for processing. There can be inconsistency in the view of tweets and processing of them in the backend as shown in Fig. 7.2. In regard of this problem, the computation and the real-time data might be lost during the analysis phase [1, 2].

Apache Storm is a real-time infrastructure that enables real-time analytics [1]. It is used by many companies with Twitter for its real-time analytics. It is developed by Apache software foundation and an open-source software platform recommended for streaming data analytics. Storm contains small set of primitive that can be used to express real-time computations on unbounded stream of data. With the help of Storm, huge set of data can be processed. Compared to Hadoop it is stateless, and with collaboration of Zookeeper, it provides a cluster state and distributed environment. It is used to handle most of the real-time tasks efficiently in a parallel manner.

The key highlights of Apache Storm are listed as follows.

- It is an open-source platform and can be installed on commodity hardware.
- It is highly reliable and guarantees message delivery for each message.

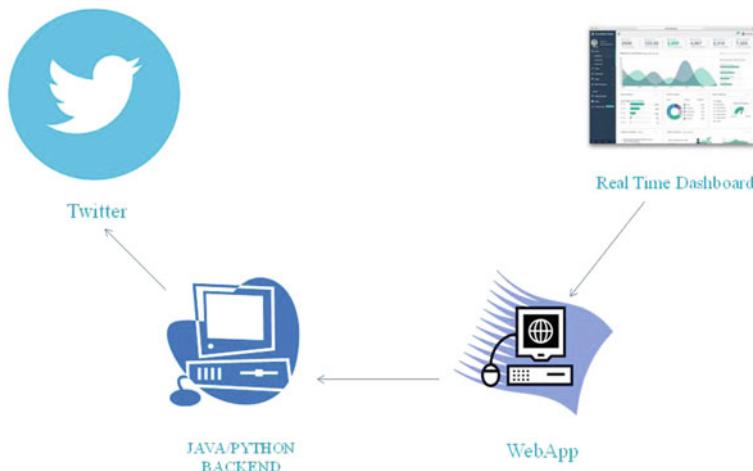


Fig. 7.1 Real-time analysis

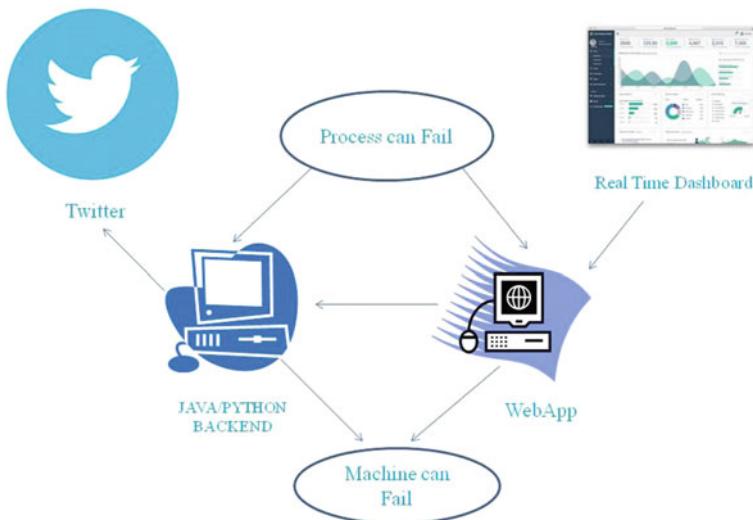


Fig. 7.2 Real-time analysis problem with Web application

- It allows real-time streaming data analytics in a fault-tolerant way and supports all the programming languages.
- It supports load balancing and multi-cluster environment, provides end-to-end delivery, and guarantees horizontal scalability.

7.2 Components of Storm

Storm is a real-time data analytics framework which reads stream of data and process them through small computational units to obtain useful results [3]. The different components of the Storm are discussed in this section which is summarized as shown in Fig. 7.3.

Spouts and Bolts

Spouts are the sources of data for topology created in Storm. They are sources or data generators such as MySQL/Kafka/Kestrel. Bolts are the units of computation on data that are processed on the tuple for operations like filtering/aggregation/joins/transformations. Tuples are immutable-ordered list of elements. Topology is a DAG with the vertices as units of computation and edges as data streams.

Figure 7.4 shows the creation of topology in Storm where three spouts and five bolts are used for data analysis. Considering the scenario of collecting the tweets and analysis of it, the following steps are followed for topology creation [4].

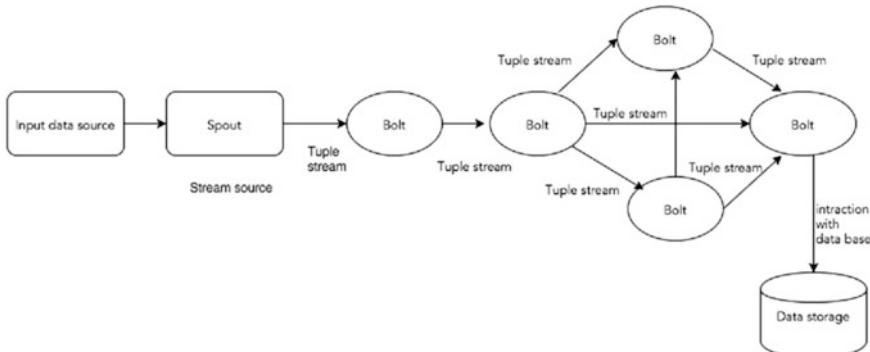
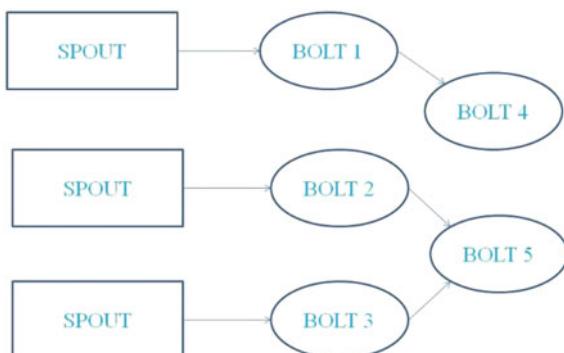


Fig. 7.3 Components of Storm

Fig. 7.4 Storm topology



- The input data are fetched from Twitter using the streaming API.
- Read the tweets with the help of spouts from Twitter API and give the tuple as output. Tweets are mainly contained in .csv file, and a single tuple may have different attributes.
- Output from spout is absorbed by bolts, and they break each tweet into individual words and calculate word count. Bolts produce tuple which are processed simultaneously and the obtained results are again fed to another bolt for further processing.

Topology

Storm topology is a combination of Spouts and Bolts. All the logical specification of the system is specified inside the topology. Usually, the topology consists of spouts first which process data feeds to bolts. The bolt is mainly a node or logical processing unit that can feed the data to another bolt as input. Topology is always running inside the Storm until it is being killed or terminated. Thus, the main function of Storm is to run one or more topology.

Tasks

The execution of spouts and bolts is known as ‘task.’ There may be multiple instances of spouts and bolts running in multiple threaded environments.

Worker Nodes

Strom supports distributed multi-node cluster setup for analysis. Each task is submitted to the respective worker nodes. As soon as the tasks arrive at each worker node the process is initiated by the worker nodes.

Stream Grouping

The data arrive as input streams, and they can be grouped based on our needs [4]. The different types of grouping are listed as follows.

- **Shuffle and field grouping:** Shuffle grouping is nothing but randomly grouping the data feed obtained from spouts and sending them to the respective bolts as shown in Fig. 7.5. In the other way Fields grouping is based on the particular column value or different column values in the CSV file.

Fig. 7.5 Shuffle and field grouping

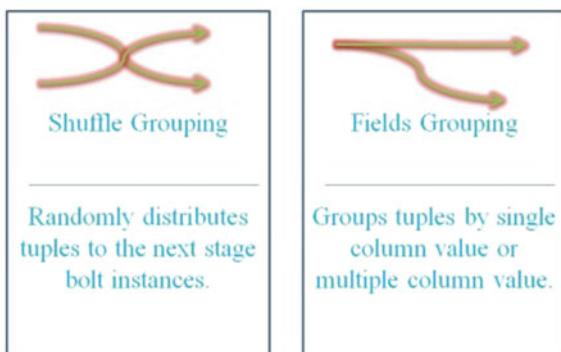
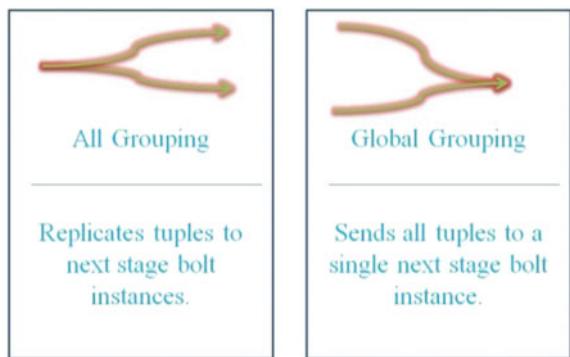


Fig. 7.6 All grouping and global grouping



- **All grouping and global grouping:** In all grouping, the data feeds are grouped as multiple instances and sent to the next stage bolt instance as shown in Fig. 7.6. But in global grouping, the multiple instances are grouped and sent to the single-targeted next stage bolt instance.

7.3 Storm Architecture

In the previous sections, the main components of the Storm such as spouts, bolts, and topology and its types were discussed. These are the components that form the backbone of the Storm cluster. However, the architectural framework of Apache Storm is based on master-slave architecture with elements as Nimbus, supervisor, executors, and workers. The overall architecture of Storm is as shown in Fig. 7.7.

- **Nimbus:** Nimbus acts as the master of the Storm architecture and helps to run the Storm topology efficiently. It is responsible to assign tasks among the worker nodes and get the job done. Thus, Nimbus has to be initiated or started before running the topology.
- **Supervisor:** Supervisors are basically the collection of nodes which are involved in doing the tasks assigned by Nimbus. They are immediately started after starting Nimbus. They manage the different jobs that need to be executed in the cluster.
- **Worker process:** The tasks that are related to the particular topology are run by the worker process. A worker process initiates multiple executors to perform the set of tasks.
- **Executor:** It runs on the slave nodes where in the multi-threaded environment, the single thread is defined as executer for a particular worker process. For a particular spout or bolt, the specific tasks are run by the executor.

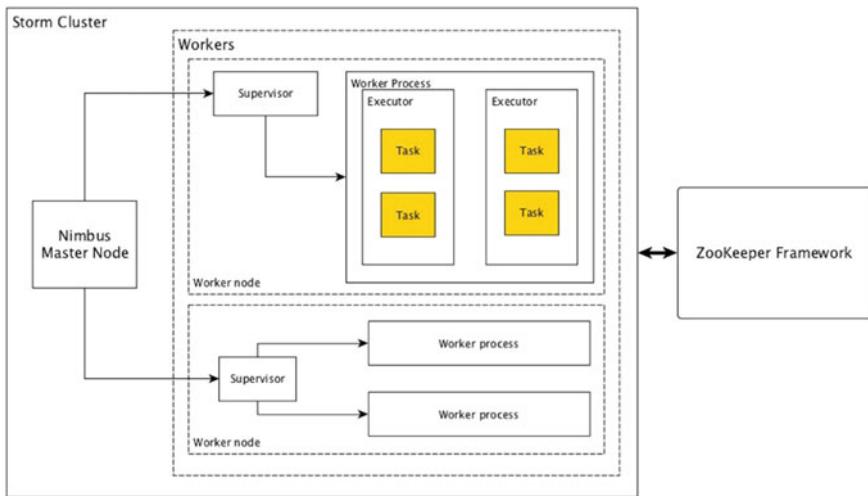


Fig. 7.7 Apache Storm architecture

- **Zookeeper framework:** Generally Storm is stateless and with the help of the Apache Zookeeper framework, a distributed cluster environment can be created. It helps in monitoring the status of the worker nodes assigned by Nimbus. It also helps in interaction between the Nimbus and supervisor.

Work-flow of Storm

The process is initiated soon after the submission of Storm topology to the Nimbus. Nimbus accepts the Storm topology and group the tasks based on the grouping mechanism based on the order of execution. Once the tasks are grouped, they are distributed among the supervisors available. The Nimbus senses the worker nodes through the acknowledgment sent by each node that they are still alive. Now Nimbus assigns the tasks to the alive supervisors and waits for the task to be completed. If the node fails suddenly, it is sensed by the Nimbus and that node is replaced with new node with the same task assigned to it. Once the task is completed, it is submitted to the data store and waits for the other task to come in.

7.4 Installation of Apache Storm

Apache Storm supports cross-platform and can be installed and executed across Ubuntu, windows, and MacOS. In this section, the installation instructions for Apache Storm are discussed. The prerequisites for the installation of Apache Storm are Java and sufficient RAM.

Installation Zookeeper

For the installation of Apache Storm, first Zookeeper needs to be installed using the following steps.

- Download the Zookeeper in tar file format from <https://zookeeper.apache.org/releases.html>
- Extract the file into a suitable folder.
- Configure ‘zoo.cfg’ file to fill following parameters as tickTime=2000 dataDir='.../zookeeper clientPort=2181, initLimit=5, syncLimit=2

Storm Installation

Once the Zookeeper is installed, Apache Storm can be installed using the following steps.

- Download Apache Storm from <http://storm.apache.org/>
- Create a configuration file as storm.yaml in the extracted tar file .../storm/conf and add the following details to it.

```
storm.zookeeper.servers: "localhost"
storm.local.dir: Storm path in your system
nimbus.host: "localhost"
supervisor.slots.ports: 6700 6701 6702 6703
```

Once the installation is completed, first the Zookeeper needs to start using the following commands.

```
bin/zkServer.sh start
bin/zkCli.sh
```

The master Nimbus in Storm and supervisors can be initiated using the following command.

```
bin/stormnimbus
bin/stormsupervisor
```

The Apache Storm UI is initiated using the command *bin/stormui*. Once, the Storm is started, it can be seen using the Web portal *localhost:8080* as shown in Fig. 7.8.

7.5 Storm in Action

In this section, Apache Storm is configured with necessary spouts and bolts for real-time analysis of the data. For getting Storm into implementation/action, spouts and bolts have to be created initially for analysis. In this section, the implementation of Storm is discussed.

The screenshot shows the Apache Storm UI interface. It includes:

- Cluster Summary:** Shows version 0.8.5, Nimbus uptime 2m 54s, 1 supervisor, 0 used slots, 4 free slots, 4 total slots, 0 executors, and 0 tasks.
- Topology summary:** Shows 1 topology with 1 ID, 1 status, 1 uptime, 1 num workers, 1 num executors, and 1 num tasks.
- Supervisor summary:** Shows 1 supervisor with 1 ID, 1 host (192.168.0.101), 1 uptime (1m 38s), 1 slot (4), and 1 used slots (0).
- Nimbus Configuration:** A table showing configuration settings like dev.zookeeper.path, drpc.child.opts, and drpc.invocations.port.

Fig. 7.8 Apache Storm topology

7.5.1 Spout Creation

It is mainly used as the source for generation of data. With the help of IRichSpout interface, the spout is created with the following methods.

Open: This method is used for the initiation of the spout and to provide an environment for the executors. The syntax for using the open() method is as follows.

open(Configuration config, TopologyInfo cont, CollectorSpout col)

where

- **config** provides Storm configuration information for the particular spout.
- **Cont** provides information about the spout position in the topology and the associated task input and output information.
- **Col** emits the tuple which is later processed by bolts.

nextTuple: *nextTuple()* the data which is generated by the collector. The syntax for using this method is as follows. It is mainly used to generate the next tuple that has to be processed by the bolt. It gets initialized as soon as the task is submitted, and it gets killed or terminated when the task is completed so that other methods can be executed. It has to sleep for few milliseconds to efficiently balance the load.

close: *close()* is used to close the task submitted.

After processing the specific tuple, the servers pass acknowledgment to the Nimbus that message is successfully delivered. The syntax for using the acknowledgment is *ack(Object msg id)*. If at all any node fails in the topology, then this message is sent to the Nimbus with the id associated to it that the particular node is failed using the command *fail(Object msg id)*.

7.5.2 Bolt Creation

Bolt takes the input as tuples, process them and produce the new tuples as output. Bolts are implemented using IRichBolt interface. The different methods that are used for bolt creation are listed as follows.

Prepare: The bolt is prepared using the prepare method() using the following command.

prepare(Configuration config, TopologyInfo cont, CollectorBolt col)
where,

- **config** provides Storm configuration information for the particular Bolt.
- **context** provides information about the bolt position in the topology, the id associated with task and the input and output information.
- **col** emits the tuple which are input to bolts.

Execute: For *execute(Tuple tup)*, the input is taken as tuple and single tuple is executed at a time. *getValue()* method is used to access the tuple. Multiple tuples can also be processed at a time and output the single tuple information at a time to the **col** which is a collector for bolt.

cleanup: The *cleanup()* method is used to clear the tuple after the task is completed.

declareOutputFields: Attributes such as id, fields are declared, and output schema can be declared using *declareOutputFields(OutFilDel dec)* method.

7.6 Twitter Analysis Using Storm

Twitter is a well-known social networking site where users can send the tweets and receive tweets. The Twitter user can post his tweet, but the non-Twitter user can only read the tweets. Tweets can be categorized for analysis using the most popular hashtags [5]. In this section, the configuration and analysis of tweets using Storm is discussed.

7.6.1 Spout Creation

The first step of the configuration is spouts creation. For the spouts to process the tweets, it has to be obtained before only. This is obtained by using Twitter Streaming API. The steps to obtain the Twitter data are same as discussed in our previous chapter of Apache Flume. The following Java code can be used once the Twitter data are collected.

```
public class SpoutCreation extends BaseRichSpout {  
  
    SpoutOutputCollector OPcollector;  
    LinkedBlockingQueue<Status> Q = null;  
    TwitterStream StreamTwit;  
    String con_key;  
    String con_sec; String a_tok;  
    String a_tok_se;  
    String[] key_words;  
  
    public SpoutCreation(String con_key, String con_sec, String  
    a_tok, String a_tok_se, String[] key_words) { this.con_key =  
    con_key; this.con_sec = con_sec; this.a_tok = a_tok;  
    this.a_tok_se = a_tok_se; this.key_words = key_words;  
    } public TwitterSampleSpout() {  
    // No need to implement for this example.  
    } @Override public void open(Map configuration, Topology-  
    Context TopoContext, SpoutOutputCollector collector) {  
    Q = new LinkedBlockingQueue<Status>(1000);  
    OPcollector = collector;  
    StatusListener listener = new StatusListener() {  
        @Override public void onStatus(Status status) {  
            Q.offer(status); }  
        @Override public void onDeleteNotice(StatusDeletionNotice  
        sdn) {  
            }  
        @Override public void onTrackLimitationNotice(int i) {  
            }  
        @Override public void onScrubGeo(long l, long ll) {  
            } @Override public void onException(Exception ex) {  
            } @Override public void onStallWarning(StallWarning arg0) {  
            // No need to implement for this example.  
            }  
    };  
  
    ConfigurationBuilder ConfBldr = new ConfigurationBuilder();  
    ConfBldr.setDebugEnabled(true).setOAuthConsumerKey(con_key)  
    .setOAuthConsumerSecret(con_sec)  
    .setOAuthAccessToken(a_tok)
```

```

.setOAuthAccessToken(a_tok_se);
StreamTwit = new TwitterStreamFacto-
ry(ConfBldr.build()).getInstance();
StreamTwit.addListener(listener); if (key_words.length ==
0) {
    StreamTwit.sample();
} else { FilterQuery query = new Filter-
Query().track(key_words);
    StreamTwit.filter(query);
}
@Override public void nextTuple() { Status ret_status =
Q.poll(); if (ret_status == null) {
    Utils.sleep(50);
} else {
    OPcollector.emit(new Values(ret_status));
}
}
@Override public void close() {
    StreamTwit.shutdown();
}
@Override
public Map<String, Object> getComponentConfiguration() {
Config ret_conf = new Config();
ret_conf.setMaxTaskParallelism(1); return ret_conf; }
@Override public void ack(Object id) {
}
@Override public void fail(Object id) {
}
@Override public void declareOutput-
Fields(OutputFieldsDeclarer dclr) { dclr.declare(new
Fields("TWEET"));
}
}
}

```

7.6.2 Bolt Creation: Identify Hashtags

In this section, the bolt is created for identifying the hashtags. The tweets which are produced by the spouts are spread across the bolts. This kind of bolt implements the recognition of unique Hashtags, and they are filtered. The pseudocode for bolt creation of unique hashtag recognition is as follows:

```

public class HashBolt implements IRichBolt
{
private OutputCollector OPcollector;

```

```

@Override
public void prepare(Map configuration, TopologyContext Topo-
Context, OutputCollector collector)
{ this.OPcollector = collector;
}
@Override public
void execute(Tuple t) {
Status tweet = (Status) t.getValueByField("tweet");
for(HashtagEntity hashtag : tweet.getHashtagEntities()) {
System.out.println("Hashtag: " + hashtag.getText());
this.OPcollector.emit(new Values(hashtag.getText()));
}
}
@Override public void cleanup() {}
@Override public void declareOutput-
Fields(OutputFieldsDeclarer dclr) { dclr.declare(new
Fields("hashtag"));
}
@Override public Map<String, Object> getComponentConfigura-
tion() {
return null;
}
}
}

```

7.6.3 Bolt Creation: To Count HashTags

After creation of the bolts, they are processed to another bolt which saves each hashtag in the memory and counts their occurrence. Thus, we can identify the details of the product which was mostly tweeted. Such an information is collected by the company for advertising. The pseudo code for the same is given below:

```

public class HashCountBolt implements IRichBolt {
Map<String, Integer> count_map;
private OutputCollector OPcollector;
@Override
public void prepare(Map configuration, TopologyContext
TPcontext, OutputCollector collector) {
this.count_map = new HashMap<String, Integer>();
this.OPcollector = collector;
} @Override
public void execute(Tuple t) {
String KEY = t.getString(0);
if(!count_map.containsKey(KEY)) {

```

```

        count_map.put(KEY, 1);
    }
    else{
        Integer c = count_map.get(KEY) + 1; count_map.put(KEY,
c);
    }
    OPcollector.ack(t);
}
@Override public void cleanup() {
    for(Map.Entry<String, Integer> en-
try:count_map.entrySet()){
        System.out.println("Result: " + entry.getKey() +“
: ” + entry.getValue());
    }
}
@Override public void
declareOutputFields(OutputFieldsDeclarer dclr) {
    dclr.declare(new Fields("hashtag"));
}
@Override public Map<String, Object>
getComponentConfiguration() {
    return null;
}
}
}

```

7.6.4 Execution of Topology

All the spouts and bolts created are submitted to topology, and they are executed to get the results. The pseudocode for the same is given below:

```

public class THStorm {
public static void main(String[] args) throws Exception{
String c_Key = args[0];
String c_Secret = args[1];
String a_Token = args[2];
String a_Token_Secret = args[3];
String[] cloned_args = args.clone();
String[] key_Words = Arrays.copyOfRange(cloned_args, 4,
cloned_args.length);
Config configuration = new Config();
configuration.setDebug(true);
TopologyBuilder TopoBuilder = new TopologyBuilder();

```

```
TopoBuilder.setSpout("twitter-spout", new SpoutCreation(c_Key, c_Secret, a_Token, a_Token_Secret, key_Words));
TopoBuilder.setBolt("twitter-hashtag-reader-bolt", new
HashBolt())
.shuffleGrouping("twitter-spout");
TopoBuilder.setBolt("twitter-hashtag-counter-bolt", new Hash-
CountBolt())
.fieldsGrouping("twitter-hashtag-reader-bolt", new
Fields("hashtag"));
LocalCluster cluster = new LocalClus-
ter();
cluster.submitTopology("THStorm", configuration,
TopoBuilder.createTopology());
Thread.sleep(10000);
cluster.shutdown();
}
```

Execute the four java files to get the below output.

Output

The output is obtained in this format where the hashtag is mentioned and the number of occurrences or count is displayed.

```
cake: 2 pastry: 3 Samsung: 2 Redmi: 4 Purse: 3 iPhone: 2 watch: 1
T-Shirt: 41
Music: 5
```

7.7 Exercises

1. What is Apache Storm? What are its benefits and applications.
2. Briefly discuss about the Usecases of Apache Storm.
3. Explain the functions of Nimbus and Supervisor.
4. What is stream grouping. Briefly discuss about the different types of stream grouping.
5. Discuss in detail about the dataflow in Storm.
6. How do Zookeeper framework help in running Storm topology?
7. Discuss in details how to identify the unique hashtags from streaming Twitter data.
8. Discuss in details how Storm helps Yahoo in its financial updates in real time.

References

1. Jain, A., & Nalya, A. (2014). *Learning storm*. Birmingham: Packt Publishing.
2. Zikopoulos, P., Eaton, C., et al. (2011). Understanding big data: Analytics for enterprise class hadoop and streaming data. NewYork: McGraw-Hill Osborne Media.
3. O'callaghan, L., Mishra, N., Meyerson, A., Guha, S., & Motwani, R. (2002). Stream-ing-data algorithms for high-quality clustering. In *ICDE* (vol. 2, p. 685).
4. Ranjan, R. (2014). Streaming big data processing in datacenter clouds. *IEEE Cloud Computing*, 1(1), 78–83.
5. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, et al. (2014). Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (pp. 147–156). ACM.

Part II

Machine Learning

Chapter 8

Basics of Machine Learning



8.1 Introduction

An algorithm is the basic entity of a computer program for problem solving. It specifies a set of steps to solve a problem that transform the given input into output. For example, to sort a given set of numbers any sorting algorithm such as merge sort, quick sort can be used to get the sorted numbers. However, the essential task is to find out the efficient algorithm for sorting and not on the knowledge of the input.

There are certain problems related to computer science where the algorithm requires knowledge of the input to transform and give the desired output. For example, consider the problem of ‘email spam classification’. In this problem, the input is an email message which can be basically structured as a file with sequence of characters and the output as message saying the mail is spam/non-spam. The algorithm for this problem should be efficient in identifying whether the email is spam or not. In order to identify the spam email, knowledge on the input data (emails) is needed because the classification of the spam differs from individual to other as well as the emails. ‘Learning’ is essential in these types of problems as there is a lack in the knowledge of data.

In the problem of ‘email spam classification’, some of the messages can be compiled and classified as spam/non-spam [1]. The algorithm needs to take care of identifying such messages and automate the task of classification with the help of computer (machine). In other words, ‘computer’ (algorithm) needs to ‘learn’ for classification and identify the spam/non-spam emails. But, in the case of sorting problem, the algorithm need not have a knowledge on the numbers to sort rather the technique for sorting of numbers should be efficient. In the current era of Internet of Things, cognitive computing predictive answers are needed for particular questions such what is the average rainfall in a particular area?, what is the stock price of a share?, how many group of people earn more than 25,000? Machine Learning helps to answer these questions with predictive models and analysis.

Machine learning deals with programs for computers that use the data from the past history or example data. The programs optimize the performance criterion based on some parameters. A model is first defined with some parameters and ‘learning’ involves the execution of the program. The execution of the program involves optimizing the parameters using the past or training data. The model can be ‘predictive’ or ‘descriptive.’ A predictive model makes the predictions of the future, whereas descriptive model describes the general observations or statistics of the data gathered.

The core task of machine learning consists of past or historical data. A mathematical model is built on this data with the use of statistics and concluding with the inference of the model. In order to build the model, the data is split into ‘training’ and ‘test.’ Training data is used to build the model and test data is used to validate the inference. Computer science is essential for machine learning because of two reasons. Firstly, efficient algorithms are needed for training and storing as well. Secondly, the algorithm results need to be represented clearly in the form of graphs, charts as well as time and space complexity.

Machine learning algorithms can be categorized into two main types namely supervised learning and unsupervised learning. In **supervised learning**, the training data is labeled and used for prediction. The parameter to be predicted is already known in the supervised learning. **Unsupervised learning** is used to find the hidden patterns of data but here there is no specific label of the data is used. One example of unsupervised learning technique is clustering which detects groups in the data that belongs to a particular feature.

8.2 Tasks in Machine Learning

In machine learning, different types of tasks are involved in carrying out the analysis [2]. These tasks are as shown in Fig. 8.1. These tasks may resemble to the data mining tasks; however, the model evaluation and prediction are the steps that differ data mining and machine learning. These tasks can be correlated to the different steps involved in the data analytics lifecycle as discussed in part 1. In this section, a brief overview of each task of machine learning is discussed.

- **Data Acquisition:** The first step in developing a machine learning application is collection of data. The data sources exist in various sources and forms. For example, the data about a topic can be obtained by web scraping of RSS feeds or through an API. The data can also be obtained from the sensors such as glucose level, temperature, wind speed.
- **Data cleaning and preparing:** In the second step, the data needs to be prepared for analysis through formatting. The data obtained has to be converted to a suitable format for analysis. The format differs from one programming language to another. In this book, we use Python as well as R for analysis and suitable data formats are used for analysis. Cleaning of data to a specific format also

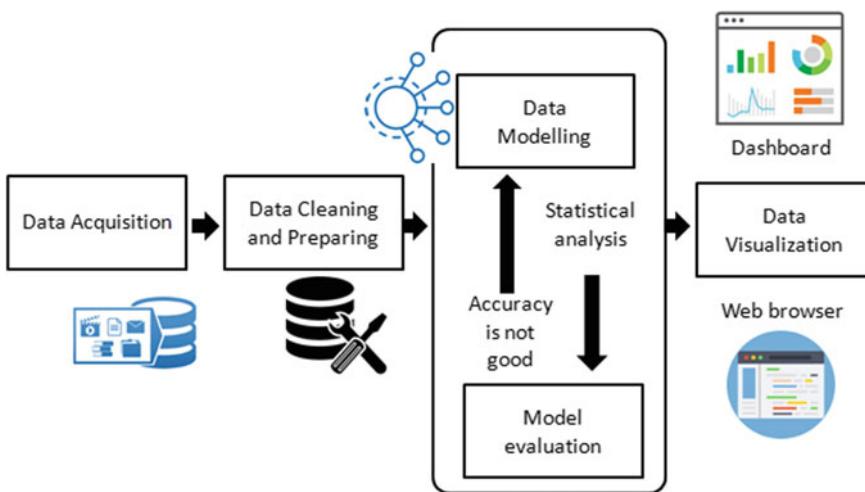


Fig. 8.1 Tasks in machine learning

depends on the algorithms used for analysis. Some of the algorithms need specific format for processing. For example, in the classification algorithms some of the data deals with strings and others with integers. In such cases, care should be taken to transform the algorithm to suitable data format.

- **Data Modeling:** This step deals with analysis of data that is collected and prepared in the previous two steps. The key task of machine learning lies in this step where the actual ‘model’ is implemented for analysis. The data is first overlooked to see if any patterns exist. It depends on the features of the data as patterns cannot be identified easily with multiple features of data. A scatter plot of different features of data helps in identifying the algorithm for analysis. The key task of this step is identifying whether supervised or unsupervised learning method to be used. In the case of supervised learning method, the data have to be split into ‘training’ and ‘test data’ for analysis. But, in the case of unsupervised learning since the target variable is unknown, the training set is not needed as the patterns need to be discovered first.
- **Model evaluation:** The algorithm used for analysis needs to be evaluated whether the model used for learning is accurate enough for prediction. In the case of supervised learning approach, the ‘test’ data are used to evaluate the model accuracy for the known values and the model predicted values. In unsupervised learning approach, some other metrics such as entropy are used for evaluation. In either case, if the model is not accurate enough, then step 3 needs to be revisited again and some parameters need to be refined. In some cases, step 2 also needs to be revisited again as the format of the data might not be suitable for the algorithm used for analysis.
- **Data visualization:** In this step, the machine learning model used for analysis is visualized with the help of charts, graphs, etc. R and Python support various

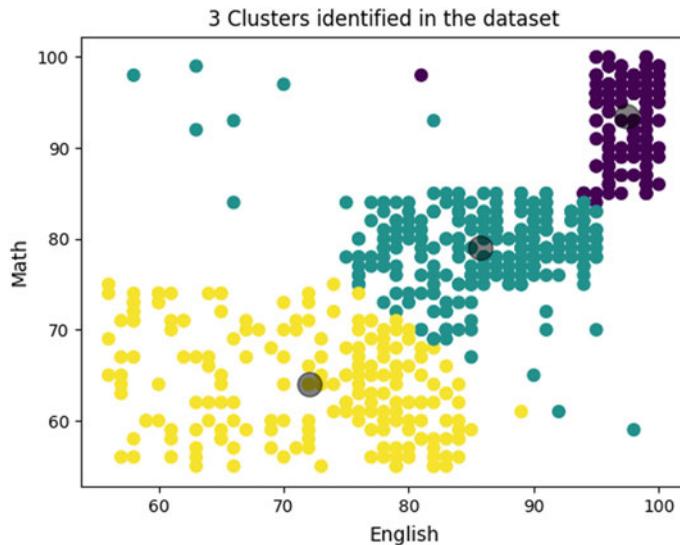


Fig. 8.2 Data visualization of clusters

forms of visualization which are discussed in this book in Part 4. Data visualization helps in getting the inferences from the machine learning models used for analysis in step 3. For example, clustering one of the unsupervised learning methods can be visualized to identify the data points that belong to different clusters.

A small example is as shown in Fig. 8.2 where there are three clusters for a class of students and their marks. In this Fig. 8.2 with the help of visualization, we can easily identify the students that are good, average, and excellent where the purple-colored students are excellent (94–100), blue-colored students are good (78–93), and yellow-colored students are average (60–75).

8.3 Data Mining and Machine Learning

Most of the times, there is an overlap between data mining and machine learning and used interchangeably. But, there are subtle differences between data mining and machine learning [2, 3]. In this section, we highlight the differences using an example.

Firstly, both data mining and machine learning tasks need dataset which may be in any form for analysis. Once the data are preprocessed and prepared for analysis, certain statistical methodologies are used to find the hidden patterns in data. These statistical methods can be normalization, regression, clustering, etc. So, essentially data mining is the use of statistical methods to find the patterns in data and explain the phenomenon behind it.

In the case of machine learning, data mining algorithms are used to build a model that predicts the future outcome using dependent and independent attributes [4]. The identification of attributes is not necessarily involved in data mining but is involved in machine learning. It is mainly because the learning method depends on the nature of the attributes. For example, in the case of regression choosing the right dependent attributes is most important for predicting the independent attribute. Regression is a machine learning technique that predicts the value of an attribute that is dependent on other attributes in the dataset. On the same lines, in the case of association rule mining minimum support and confidence is required to determine the association rules. Association rule mining is a data mining technique that determines the most frequent item sets for given transactions of purchase data.

The second difference between data mining and machine learning is accuracy. In the case of machine learning, accuracy of the model is most important is concluding whether the model built is correct or not. At the end of analysis in machine learning, certain measures such as precision, recall, confusion matrix are used for determining the accuracy of the model. In the case of data mining techniques such as clustering, different clusters of data points are found and grouped together.

Another difference between data mining and machine learning is dealing with obstacles during analysis. In the case of data mining, any obstacle has to be dealt with the intervention of a person for initiation of analysis, whereas in the case of machine learning, learning will take care of the obstacle and moves ahead. This can be better understood with clustering and regression techniques. Clustering is a technique where a group of data points are clustered together. Regression is a technique of predicting the value of an independent variable that is dependent on the variables in the dataset.

A mining technique that does the grouping does not know what to do in the case of arrival of new points as it needs to be regrouped based on the clusters. But with regression arrival of a new point and its coordinate can be easily identified as the dependency is known. The new point gets its value using the previous values of dependent variables.

8.4 Machine Learning Terminology

In machine learning certain terminologies such as attributes, features, model, and accuracy are used. Before the machine learning algorithm is used, a dataset is first needed for analysis and learning. Dataset plays a key important role in machine learning methods. A machine learning model cannot be built without understanding the data. The main characteristics of data are format, attributes, and volume. The attributes of the data play an important role in building the machine learning model [5]. In this section, an overview of different terminologies is discussed that are helpful in the upcoming chapters of this part.

8.4.1 Supervised Learning

In supervised learning methods, a set of labeled training data is used for learning and prediction. For each input object, a labeled output object is used in the training dataset. Consider the example of a twitter dataset as shown in Table 8.1. In this Table 8.1, tweets are assigned with certain labels.

Now, if another tweet ‘Hadoop 2.0 upgraded finally!!!’ can be assigned a label '#Hadoop' automatically by the machine learning method. Likewise, the supervised learning approaches use the labeled training dataset to learn and categorize the data. Some of the supervised learning techniques are regression, Naïve Bayes, etc, which are discussed in the subsequent chapters of this book.

8.4.2 Unsupervised Learning

In unsupervised learning, the methods initially do not know the variables to be used for learning. It finds the hidden patterns of data to learn and try to build a model on it. It is more likely associated with data mining methods such as clustering where a group of clusters is formed based on certain similarities. Consider an example of a student dataset as shown in Table 8.2. Table 8.2 consists of four students and three subjects S1, S2, S3.

Table 8.1 Twitter for supervised learning

Tweet	Label
Yeah!! I got Hadoop working	#Hadoop
Regression for prediction of stocks is working!!	#stock
Machine learning is the next frontier!!	#machine learning
Network data analytics is selling out!	#selling

Table 8.2 Student data for unsupervised learning

Student ID	S1	S2	S3
1	55	43	56
2	32	47	25
3	67	75	87
4	57	45	89
5	56	86	55
6	88	65	78
7	78	43	77
8	55	88	44

Looking at this dataset, it is a challenging task to determine which of the students excel in subjects S1 and S2, S2 and S3 and likewise. In this case, we can apply clustering and build a group of clusters that helps to identify the clusters of students for a pair of subjects. Unsupervised learning technique clustering is discussed in the subsequent chapters of this book.

8.4.3 Attributes

A dataset is always associated with attributes that play a major role in building the machine learning model. The best way to understand the attributes is through an example. Consider a salary dataset as shown in Table 8.3. The attributes of this dataset are ID, Income, Age, Education, and Gender. Each person in the dataset has identification number (ID). Income is expressed in terms of thousands, for example, (65→65,000). Age and education are expressed in the number of years. Gender M represents male and F represents female. Likewise, for every other dataset and machine learning technique, we need to identify the type and units it is referring and then decide what type of machine learning can be applied on it.

In this case of income dataset, regression can be applied to estimate or predict the income of a person. For this technique, we need to know dependent variables and independent variables. Here, the independent variable is income and dependent variables are age, education, and gender. On the same, dataset clustering can be applied between age and income where different groups of income can be grouped together. Similarly, for every machine learning technique, different attributes of dataset need to be identified first and then the machine learning techniques need to be used.

8.4.4 Model

Model is the basic entity of machine learning that describes the various attributes the dataset and their relationships. In machine learning, analysis is carried out through a model and the same is used for prediction. For example, regression modeling can be done on the income dataset considered in Table 8.3. In Python

Table 8.3 Income dataset example

ID	Income	Age	Education	Gender
1	91	50	12	M
2	117	55	10	M
3	65	43	7	F
4	78	45	18	F
5	45	24	17	M
6	55	65	20	F
7	12	28	21	F

programming, the regression modeling can be applied as shown below where LinearRegression() is the method to build linear regression model. For this model, income_education_train and income_age_train are used as training datasets for predicting income of the future data points.

```
Model = LinearRegression()
model.fit(income_X_train, income_y_train)
```

8.4.5 Accuracy

In Machine learning, once the model is built its score of accuracy decides the prediction rate of future data points. The accuracy of a machine learning model is determined with measures such as confusion matrix, precision, recall, entropy. For example, in the case of a regression technique the accuracy score is determined by the coefficients of the model. The coefficients are called as slope and intercept. Using the same example of income dataset, the coefficients of the regression model can be determined as below.

```
Print('Coefficients: \n', model.coef_)
```

8.4.6 Prediction

The main aim of machine learning algorithms is to predict values of new data points. Prediction of data depends on the type of learning, i.e., supervised learning and unsupervised learning. Regression is an example of supervised learning where its coefficients can be used for prediction. For example, to predict the value with regression, an X vector, model coefficient, and model intercept are needed as shown below. Here, the vector a is the prediction vector. The regression model coefficient is multiplied with vector X and added with model intercept for regression. However, the fitting line can also be used to see if the model is fitting the points or not as shown in Fig. 8.3. There are two outliers that are not fitted into the curve as shown in Fig. 8.3. A detailed explanation of fitting the curve with regression is discussed in the subsequent chapters.

```
A = model.coef_ * X + model.intercept_
print(a)
[[ 6.2]
 [ 7.2]
 [ 9.2]
 [11.2]
 [12.2]]
```

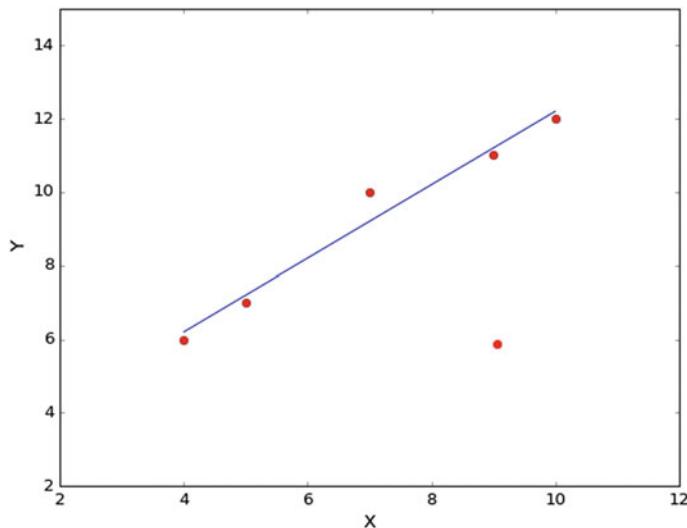


Fig. 8.3 Linear regression line curve fitting

Clustering is an example of unsupervised learning wherein a group of points form one or more clusters. Clusters are formed based on the centroid calculation for a number of iterations until the number of clusters remains the same. The prediction of belongingness of a new point to an existing cluster is difficult as the centroid may change and the algorithm once again needs to be executed to determine the centroid of the cluster. So, in the case of unsupervised learning methods, it is hard to make predictions but rather they are used for finding the hidden patterns of data.

8.5 Getting Started with Machine Learning in Python

In this book, Python is used as the primary language for machine learning methods. It is widely used popular language used for building machine learning applications. Python has a clear syntax and easy to manipulate with text. It supports higher level data types such as lists, dictionaries, sets. The instructions on installing Python and the basic examples of Python are included in Appendix. Python interpreter ‘Anaconda’ is used widely for machine learning and statistical programs. The instructions on installing and configuring Anaconda are included in Appendix. In the Anaconda interpreter, ‘Scikit-learn’ module is used for machine learning programming using Python.

8.5.1 Getting Started with Scikit-Learn

Scikit-learn is used for machine learning in Python [6]. It includes modules for machine learning such numpy, regression, Naïve Bayes, k-means. The instructions for installing scikit-learn are included in the Appendix. Before we explain how different machine learning methods can be used with scikit-learn, a simple example on loading the dataset from scikit-learn is as shown below.

```
>>> from sklearn import datasets
>>> iris = datasets.load_iris()
```

In the sklearn module, datasets include all the datasets such as iris, digits, boston_housing. The load() method needs to be used for loading the required dataset. For example, here iris data are loaded. The data points can be viewed with data() method. So, the output of data method for iris data is as shown below. Since the points are more in number, only some of the data points are shown.

```
>>> print(iris.data)
[[ 5.1 3.5 1.4 0.2]
 [ 4.9 3. 1.4 0.2]
 [ 4.7 3.2 1.3 0.2]
 [ 4.6 3.1 1.5 0.2]
 [ 5. 3.6 1.4 0.2].....]
```

Before we work on any dataset, the features or attributes of the data need to be known to us. It can be known with ‘feature_names()’ method in the sklearn module as shown. Here, the features of the iris data are shown namely sepal length, sepal width, petal length, and petal width.

```
>>> print(iris.feature_names)
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

The entire description of the dataset can be viewed with ‘DESCR()’ method as shown. It provides the description of the iris dataset. As we can see, it provides dataset characteristics specifying the number of instances and the attribute information. It provides a summary of the dataset on minimum, maximum, and mean values for each of the attribute in the dataset.

```
>>> print(iris.DESCR)
Iris Plants Database
=====
Notes
```

Data Set Characteristics:

:Number of Instances: 150 (50 in each of three classes)

:Number of Attributes: 4 numeric, predictive attributes and the class

:Attribute Information:

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- class:
 - Iris-Setosa
 - Iris-Versicolour
 - Iris-Virginica

:Summary Statistics:

	Min	Max	Mean	SD	Class Correlation
sepal length:	4.3	7.9	5.84	0.83	0.7826
sepal width:	2.0	4.4	3.05	0.43	-0.4194
petal length:	1.0	6.9	3.76	1.76	0.9490 (high!)
petal width:	0.1	2.5	1.20	0.76	0.9565 (high!)

:Missing Attribute Values: None

:Class Distribution: 33.3% for each of 3 classes.

:Creator: R.A. Fisher

:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)

:Date: July, 1988

This is a copy of UCI ML iris datasets.

<http://archive.ics.uci.edu/ml/datasets/Iris>

8.6 Exercises

1. What is machine learning? Explain the differences between machine learning and data mining.
2. Briefly explain the different types of machine learning.
3. Let us say you want to build an application to reduce the time for an ambulance to reach the nearby hospital. Would you choose machine learning ?
4. Discuss the scenarios where machine learning is useful and not useful.

References

1. Thrun, S., & Pratt, L. (Eds.). (2012). Learning to learn. Springer Science & Business Media
2. Han J, Kamber M (2006). Data mining: Concepts and techniques, 2nd edn. Morgan Kaufmann, San Francisco
3. Michalski, R. S., Carbonell, J. G., & Mitchell, T. M. (Eds.). (2013). Machine learning: An artificial intelligence approach. Springer Science & Business Media
4. Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). Data mining: Practical machine learning tools and techniques. Burlington: Morgan Kaufmann
5. Kotsiantis, S. B., Zaharakis, I., Pintelas, P. (2007). Supervised machine learning: A re-view of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering*, 160, 3–24
6. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011, Oct). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830

Chapter 9

Regression



9.1 Introduction

In the previous chapter, during the discussion of machine learning terminologies some concepts on regression were introduced. It was related to accuracy and prediction. In this chapter, machine learning technique regression is discussed in detail. Regression analysis is one of the methods that explain the influence of a set of variables on the outcome of variable of interest. In this method, the outcome variable is called as dependent variable and the additional variables that influence the outcome variable are called independent variables. Suppose, if we need to find the answers for the following questions, then regression analysis is used [1].

- What is the expected yield of crops for a given agricultural field?
- What is number of complaints in a call center depending on the waiting time to answer?
- What is the number of people who claim their amount in an insurance company?

9.2 Simple Linear Regression

It is one of the widely used regression methods that depicts the linear relationship between the input variables and the outcome variables [1, 2]. It can be represented by Eq. 9.1,

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_{p-1} x_{p-1} + \varepsilon \quad (9.1)$$

where

- y is the outcome or independent variable;
- x_1, x_2, \dots, x_{p-1} are the independent variables;
- β_0 is the value of y when $x = 0$;

- β_i is the change in the value of y corresponding to x_i ;
- ϵ is the random error that represents the difference between the linear model and the actual values.

The main goal of linear regression is to find the regression coefficients and predict the target numerical value. For example, consider the following equation where b is the independent variable and a, c are dependent variables. In the equation, 0.015 and 0.99 are coefficients/weights for regression analysis. The process of finding out the regression weights is known as regression analysis. In order to predict the new unknown value of b , the regression coefficients are multiplied with values of a and c and added together.

$$b = 0.015 * a - 0.99 * c$$

The generic approach to regression analysis consists of the following steps [3]:

- Data collection: The data that needs to be analyzed using regression can be static or dynamic. It is collected and stored first before analysis.
- Data preparation: Regression analysis can be performed only on numeric values. Hence, if there are any nominal values in the dataset, it has to be mapped to binary or numeric values.
- Data Analysis: If the dataset has less number of attributes (3–5) regression can be performed directly by using the appropriate dependent and independent variables. A visual 2-D plot can be used in the case of large dataset for identifying the dependent and independent variables.
 - Once, the dependent and independent variables are known regression analysis is carried out. The accuracy score and the coefficients are used to evaluate the successful rate of prediction of further numeric values and model obtained.

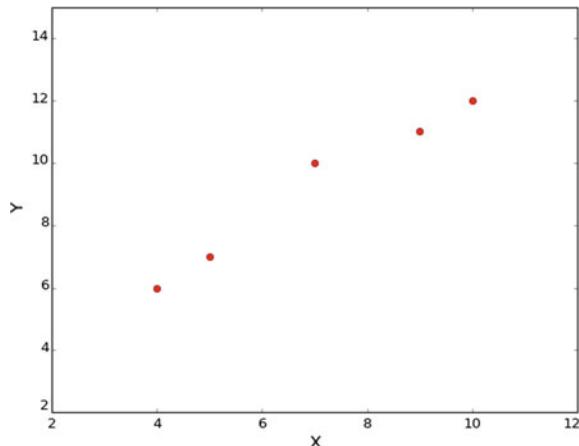
9.3 Linear Regression in Python Example

A small example on linear regression is discussed in this section. The input considered for this example is $X = [4, 5, 7, 9, 10]$ and $Y = [6, 7, 10, 11, 12]$. It represents the coordinates of the points in a X - Y plane. The modules required for simple linear regression in Python are:

- **LinearRegression:** It is the module required for running simple linear regression in Python.
- **Numpy:** It is the module required for initializing the values of X and Y .
- **Matplotlib:** It is the module used to draw the fitted model of the regression.

The X and Y data are initialised with the values and plotted with the help of `plot()` function. It takes the arguments X and Y . The output of the plot is as shown in Fig. 9.1. In this Fig. 9.1, it can be observed that there are five data points.

Fig. 9.1 Data for simple linear regression



LinearRegression() is used to build the regression model and fit() method is used to fit the model.

The prediction vector ‘ a ’ is initialised with the coefficients of the regression model. This vector ‘ a ’ is combined with ‘ X ’ to observe the fitted model. The fitted model of the regression is as shown in Fig. 9.2. In Fig. 9.2, it can be observed that there is only one outlier and the model is fitted for other data points. Hence, the regression model can be used to forecast the future data points.

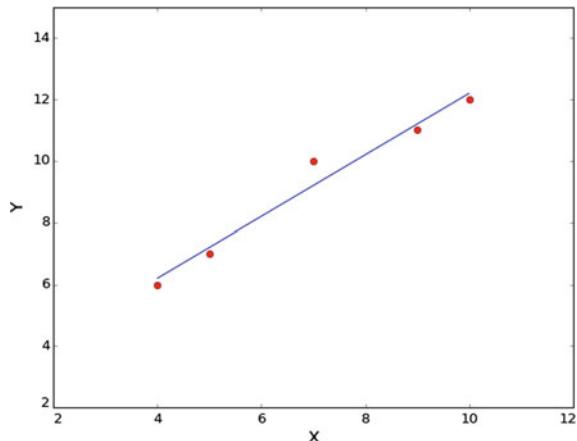
```
#import statements
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt

#Input data of X and Y
X = np.array([4,5,7,9,10]).reshape(5, 1)
y = np.array([6, 7, 10, 11, 12]).reshape(5, 1)

#Plot the input data
plt.plot(X, y, 'ro')
axes = plt.gca()
axes.set_xlim([2, 12])
axes.set_ylim([2, 15])
plt.show()

#Build Linear regression model
model = LinearRegression()
model.fit(X, y)
```

Fig. 9.2 Simple linear regression model in Python



```
#prediction vector a
a = model.coef_ * X + model.intercept_

#Plot the input points with predicted model
plt.plot(X, y, 'ro', X, a)
axes = plt.gca()
axes.set_xlim([2, 12])
axes.set_ylim([2, 15])
plt.show()
```

9.4 Polynomial Regression

In simple linear regression, if the fitted model has many outliers, then the accuracy of the model decreases. However, the accuracy of the model can be improved using polynomial features [4]. The basic idea of the polynomial regression is to increase the power of the input features of the regression curve as shown in Eq. 9.2.

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \cdots + \beta_n x_n + \varepsilon \quad (9.2)$$

where

- y is the outcome or independent variable;
- $x_1, x_2 \dots x_n$ are the independent variables;
- β_0 is the value of y when $x = 0$;
- β_i is the change in the value of y corresponding to x_i ;
- ε is the random error that represents the difference between the linear model and the actual values.

Polynomial regression can be better understood with an example. A small example is as shown below with ‘X’ and ‘Y’ as the inputs. A data plot of the input data is as shown in Fig. 9.3. In the Fig. 9.3, it can be seen that a linear regression model is built for the data using LinearRegression() function. The fitted model can be seen in Fig. 9.4. It can be observed in the Fig. 9.4 that there are many outliers in the fitted model that is based on simple linear regression. The score of the linear regression is 84% for this fit. It can be improved with polynomial regression.

```
#import statements
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
```

Fig. 9.3 Data plot for polynomial regression

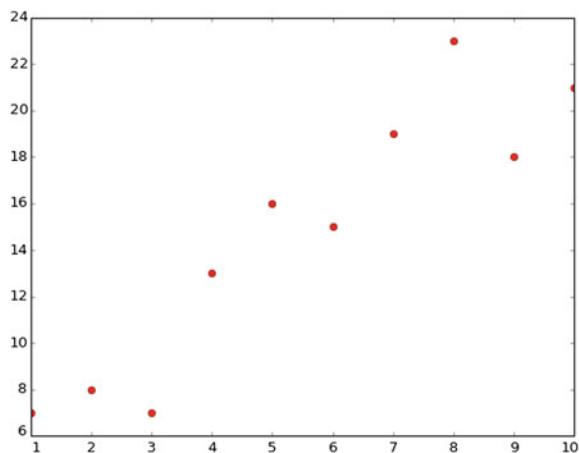
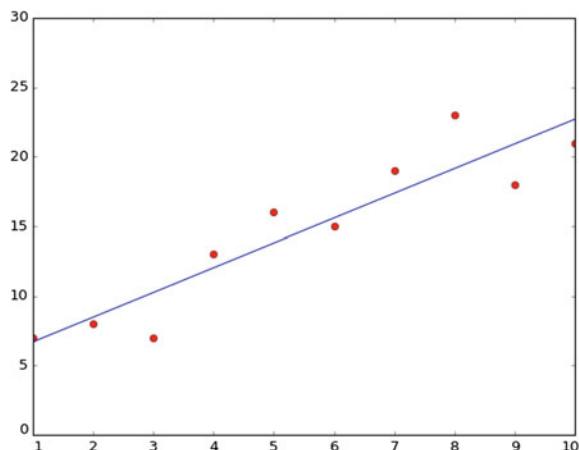


Fig. 9.4 Simple regression plot for polynomial regression data



```

#Data inputs
X = np.arange(1, 11).reshape(10, 1)
y = np.array([7, 8, 7, 13, 16, 15, 19, 23, 18,
21]).reshape(10, 1)

#Plot the data points
plt.plot(X, y, 'ro')
plt.show()

#Build the linear regression model

model = LinearRegression()
model.fit(X, y)

#prediction vector a
a = model.coef_* X + model.intercept_

#Plot the fitted model
plt.plot(X, y, 'ro', X, a)
axes = plt.gca()
axes.set_xlim([0, 30])
axes.set_ylim([0, 30])
plt.show()

print('Linear regression score', model.score(X, y))

```

Output:

Linear regression score 0.849880708424

Simple regression model obtained in Fig. 9.4 can be improved with polynomial regression as shown below. Initially, the polynomial features are added to vector ‘ X ’ by multiplying with X^{**2} for the data as shown in Fig. 9.5. The model is then fit to the polynomial features added. The plot of the fitted model is as shown in Fig. 9.6. It can be observed that in Fig. 9.6 the fitted model covers more data points than the simple linear regression model in Fig. 9.4. The accuracy of the polynomial fitted model is 87% which is better than the earlier simple linear regression model.

```

#Polynomial features to improve the accuracy
X = np.arange(1, 11)
X = np.c_[X, X**2]
x = np.arange(1, 11, 0.1)
x = np.c_[x, x**2]

```

Fig. 9.5 Polynomial regression data

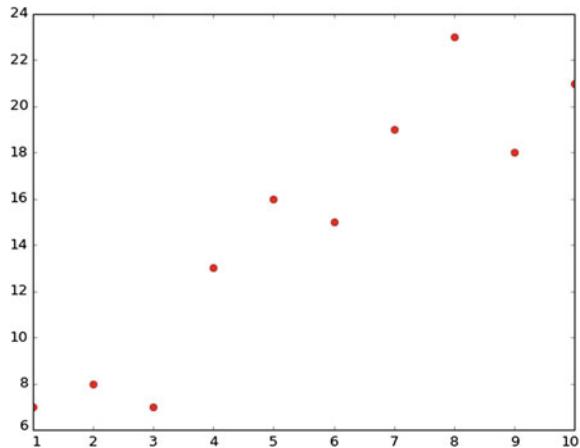
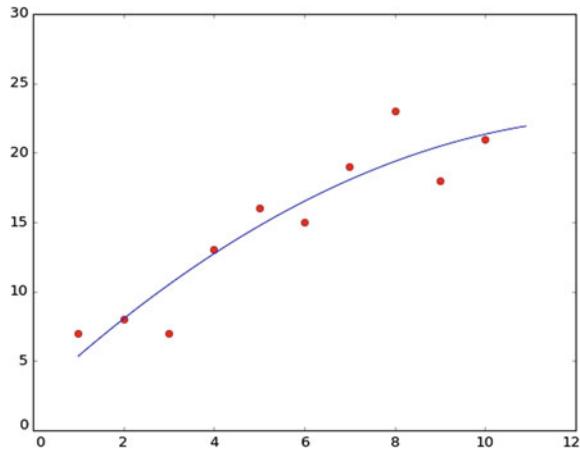


Fig. 9.6 Polynomial regression plot for polynomial regression data



```
#Build the linear regression model
model.fit(X, y)
a = np.dot(x, model.coef_.transpose()) + model.intercept_

#Plot the fitted model
plt.plot(X[:, 0], y, 'ro', x[:, 0], a)
axes = plt.gca()
axes.set_xlim([0, 30])
plt.show()
print('X^2 model score', model.score(X, y))
```

Output:

```
X^2 model score 0.87215506915
```

The main disadvantage of polynomial regression is it might lead to overfitting. With polynomial regression, care should be taken such that many polynomial features are not added. An example of overfitting with polynomial regression is as shown below. Initially, the polynomial features are added to vector ‘ X ’ by multiplying with $X^{**2}-X^{**9}$ as shown in Fig. 9.7. The model is then fit to the polynomial features added. The plot of the fitted model is as shown in Fig. 9.8. It can be observed that in Fig. 9.8 the fitted model covers all data points than the polynomial regression model in Fig. 9.6. The accuracy of the polynomial overfitted model is 99% which is overfitted.

Fig. 9.7 Polynomial regression data

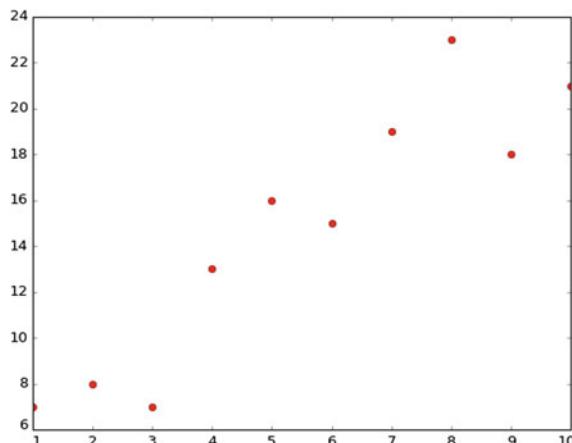
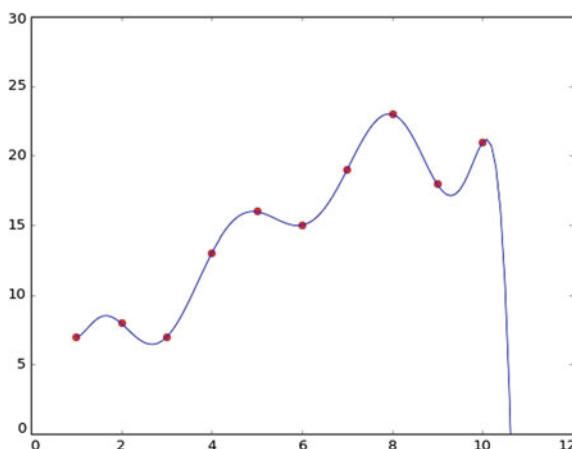


Fig. 9.8 Overfitted polynomial regression plot for polynomial regression data



```
#Polynomial features to improve the accuracy
X = np.arange(1, 11)
X = np.c_[X, X**2, X**3, X**4, X**5, X**6, X**7, X**8, X**9]
x = np.arange(1, 11, 0.1)
x = np.c_[x, x**2, x**3, x**4, x**5, x**6, x**7, x**8, x**9]

#Build the linear regression model
model.fit(X, y)
a = np.dot(x, model.coef_.transpose()) + model.intercept_

#Plot the fitted model
plt.plot(X[:, 0], y, 'ro', x[:, 0], a)
axes = plt.gca()
axes.set_xlim([0, 30])
plt.show()
print('X^9 model score', model.score(X, y))
```

Output:

X^9 model score 0.999999999976

9.5 Finding the Best Fit for Regression

The main drawback of polynomial regression is overfitting. To overcome this drawback, polynomial features need to be added to the regression curve carefully. In this section, an example on finding the best fit for regression is shown. The data considered for the previous examples of simple linear regression and polynomial regression are taken here as well. For the previous data, new data points from 11 to 16 are added to see the effect of models. The code is as shown below and the plot is as shown in Fig. 9.9 for the data and the fitted model in Fig. 9.10.

```
#new data from (11,16)
X = np.arange(1, 16)
y = np.append(y, [24, 23, 22, 26, 22])

#Plot the data
plt.plot(X, y, 'ro')
plt.show()
```

A simple linear regression model is fitted into the newly added data. The output of the simple linear regression model with new data is as shown in Fig. 9.10. It can be seen from Fig. 9.10 that the model does not fit to the newly added data points,

Fig. 9.9 Regression data to find the best fit

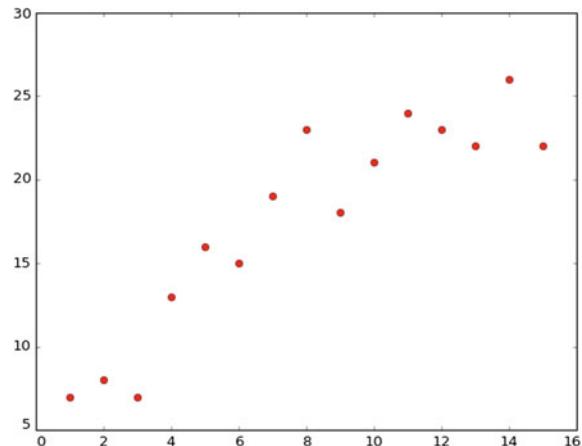
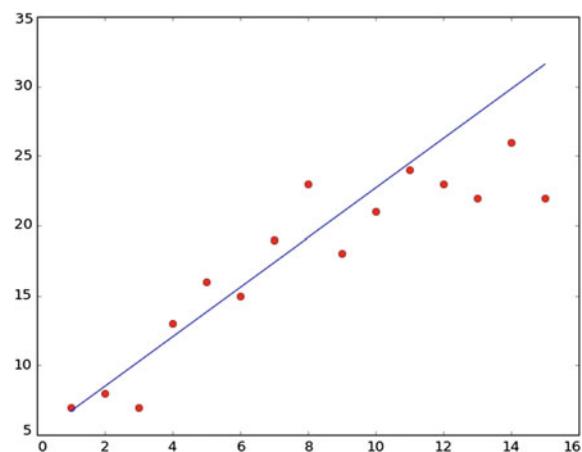


Fig. 9.10 Simple linear regression plot to find the best fit



i.e., from 11 to 16. All the data points lie under the regression curve. Hence, simple linear regression is not the best fit. Polynomial regression for the same data points is carried out in the next code section.

```
#Simple linear regression model to find the best fit
X = np.arange(1, 16).reshape(15, 1)
model.fit(X[:10], y[:10])
model.score(X[10:], y[10:])

#Plot the linear regression model with new data
a = np.dot(X, model.coef_.transpose()) + model.intercept_
plt.plot(X, y, 'ro', X, a)
plt.show()
```

Since the simple linear regression does not fit the data points previously, polynomial regression can be done using the code as below. The vector X is now increased with powers of X^{**2} . The plot of the data points can be seen in Fig. 9.11 where the new data points from 11 to 16 are added. The output of the polynomial regression plot to find the best fit is as shown in Fig. 9.12. Now, it can be seen that the regression curve fits to newly added data than the simple linear regression curve in Fig. 9.10.

```
#Polynomial linear regression model to find the best fit
X = np.arange(1, 16).reshape(15, 1)
X = np.c_[X, X**2]
x = np.arange(1, 16, 0.1)
x = np.c_[x, x**2]
model.fit(X[:10], y[:10])
model.score(X[10:], y[10:])

#Plot the polynomial regression model with newdata
a = np.dot(x, model.coef_.transpose()) + model.intercept_
plt.plot(X[:, 0], y, 'ro', x[:, 0], a)
axes = plt.gca()
axes.set_xlim([0, 30])
plt.show()
```

Now, the following code will try to improve the polynomial regression curve obtained in Fig. 9.12 by adding more polynomial features. The polynomial features are increased from X^{**2} to X^{**9} now. The plot of the data points is as shown in Fig. 9.13. The output of the polynomial regression plot can be seen in Fig. 9.14.

Fig. 9.11 Polynomial regression data to find the best fit

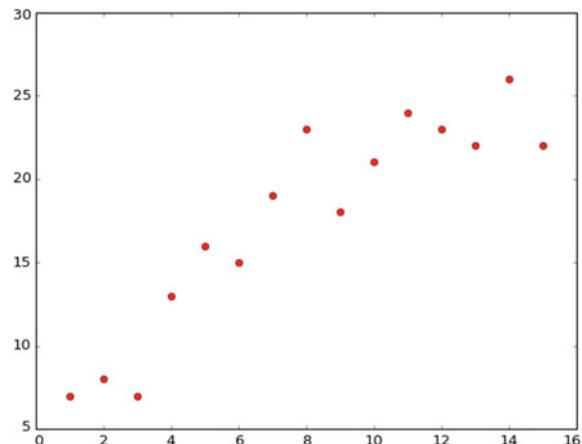


Fig. 9.12 Polynomial regression X^{**2} plot

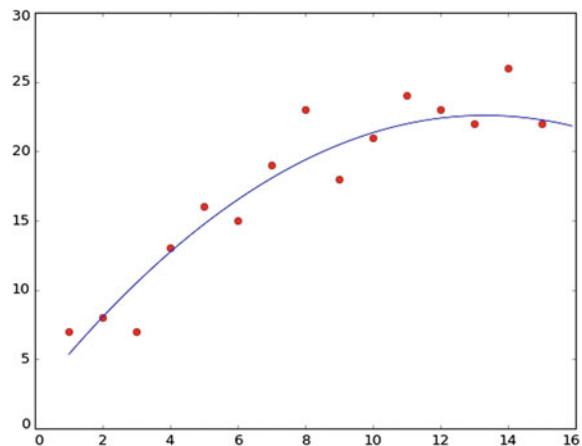


Fig. 9.13 Polynomial regression data to find the best fit

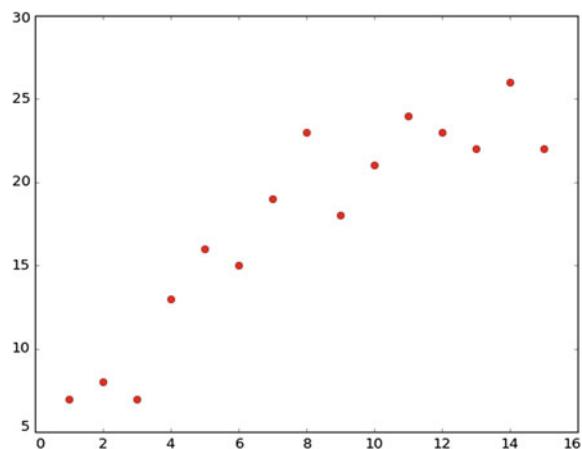
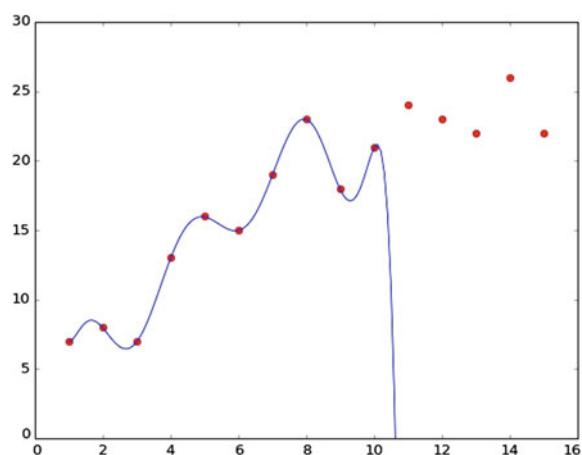


Fig. 9.14 Polynomial regression X^{**9} plot



It can be seen that newly added points in the regression curve do not fit. Hence, the regression model is overfitted and is not able to fit to the newly added points to dataset.

```
#Polynomial linear regression model X**9 to find the best fit
X = np.arange(1, 16).reshape(15, 1)
X = np.c_[X, X**2, X**3, X**4, X**5, X**6, X**7, X**8, X**9]
x = np.arange(1, 16, 0.1)
x = np.c_[x, x**2, x**3, x**4, x**5, x**6, x**7, x**8, x**9]
model.fit(X[:10], y[:10])
model.score(X[10:], y[10:])

#Plot the polynomial regression model X**9 with new data
a = np.dot(x, model.coef_.transpose()) + model.intercept_
plt.plot(X[:, 0], y, 'ro', x[:, 0], a)
axes = plt.gca()
axes.set_xlim([0, 30])
plt.show()
```

The best fit of regression curve is the polynomial regression with X^{**2} features as shown in Fig. 9.12 as it fits most of the points than the regression curves in Figs. 9.10 and 9.14. In this way, the best fit for the regression technique can be evaluated. However, the polynomial features vary from one dataset to other as the feature types differ. Thus, in the case of polynomial regression, the power that is increased needs to be evaluated correctly.

9.6 Regression Case Study on Mtcars Dataset

In this section, a case study on the regression is discussed with mtcars dataset [5]. The dataset consists of 33 instances with features model, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb. The main aim of the case study is to predict mileage or mpg for a given model with features number of cylinders (cyl), horsepower (hp), weight (wt), and disp. The rest of the features in the dataset are ignored as they are not necessary for implementing the regression.

Firstly, the modules required for running the regression are same as discussed in the previous examples. The data is first read from the csv file using pandas module. The features included for regression analysis are mpg, disp, hp, and wt. Only these features are extracted and included for analysis and rest of them are ignored.

The regression analysis is performed using the data attributes extracted and LinearRegression() module with 25% as the test dataset and remaining for training. The fit() function is used to fit the linear regression model. The fitted model coefficients and intercept are used to predict the values of 25% of the data. It can be

observed in the output that the predicted values are close to the actual values in the dataset. Thus in this way, the regression analysis is done on the mtcars dataset to predict miles per gallon (mpg) for the cars.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression as LR
from sklearn.model_selection import train_test_split as split

data = pd.read_csv('mtcars.csv')
attributes = ['mpg', 'disp', 'hp', 'wt']
data = data[attributes]
train, test = split(data, test_size=0.25)
print(train, test)

X = np.array(train[['disp', 'hp', 'wt']])
Y = np.array(train['mpg'])

model = LR()
model.fit(X, Y)
coeffcients = list(model.coef_)
intercept = model.intercept_
print("actual value\tpredicted value")
test = np.array(test)
for item in test:
    #print(item)

    print(item[0],"\t\t",item[1]*coeffcients[0]+item[2]*coeffcients[1]+item[3]*coeffcients[2] + intercept)

Output:
actual value      predicted value
17.3            17.4356998861
32.4            26.6439623622
15.2            17.2361160918
13.3            15.1116662782
22.8            25.3773264039
21.0            23.7865745238
18.1            20.7949293828
15.2            19.6743069392
```

9.7 Ridge Regression

Ridge regression is one of the machine learning techniques that is carried out when the data suffers from multicollinearity. Multicollinearity occurs in the regression modeling where the value of the independent variable is predicted based on the other data features. There will be only certain change in the accuracy of the model due to the multicollinearity in the data for building the ridge regression model. The main difference between the simple regression and ridge regression is in the model accuracy.

In this section, a small example on the ridge regression is discussed with mtcars as the dataset. The same dataset was considered as a part of the simple linear regression. The same features are extracted from the dataset and *ridge()* function is used to build the regression model. The following code demonstrates the ridge regression for the mtcars dataset. It can be seen that the results of the ridge regression indicate that the mean squared error is 25% and the actual values with the predicted values are correlated with each other.

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np

# Load the cars dataset.
data = pd.read_csv('mtcars.csv')
attributes = ['mpg', 'disp', 'hp', 'wt']
data = data[attributes]
train, test = train_test_split(data, test_size=0.10)

# Transform the train and test data into features and targets.
train_X = np.array(train[['disp', 'hp', 'wt']])
train_Y = np.array(train[['mpg']])
test_X = np.array(test[['disp', 'hp', 'wt']])
test_Y = np.array(test['mpg'])

# Train and test the Ridge regression model.
model = Ridge()
model.fit(train_X, train_Y)
results = model.predict(test_X)

# Metrics.
print("MeanSquarederror: {}".
      format(mean_squared_error(test_Y, results)))
```

```

print("Actual Value\tPredicted value")
for i in range(len(results)):
    print("{0}\t\t{1}".format(test_Y[i], results[i]))


Mean Squared error : 25.123987884593227
Actual Value  Predicted value
33.9          [ 27.84741429]
16.4          [ 14.96697201]
15.0          [ 10.28594409]
14.7          [ 8.4082383]

```

9.8 Exercises

1. Explain linear regression: Is Linear regression sensitive to outliers?
2. What are the cons of performing a regression?
3. Explain the need for regression analysis.
4. What is overfitting? Is overfitting for regression more likely if we have less data?
5. What is underfitting? Suppose a linear regression model is underfitting the data what will you do?
6. In mtcars dataset, Suppose a correlation between No of cylinders and Gross horsepower is found to be -1.05 On the basis of this what can we infer?

References

1. Seber, G. A., & Lee, A. J. (2012). *Linear regression analysis* (Vol. 329). Wiley.
2. Yan, X., & Su, X. (2009). *Linear regression analysis: Theory and computing*. World Scientific.
3. Neter, J., Kutner, M. H., Nachtsheim, C. J., & Wasserman, W. (1996). *Applied linear statistical models* (Vol. 4, p. 318). Chicago: Irwin.
4. Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to linear regression analysis* (Vol. 821). Wiley.
5. Lichman, M. (2013). UCI Machine Learning Repository (<http://archive.ics.uci.edu/ml>). Irvine, CA: University of California, School of Information and Computer Science.

Chapter 10

Classification



Classification is one of the machine learning techniques that help in categorizing the data into different classes based on the features present in the dataset. Here, first the features are studied in the dataset to see how the classification can be done. If the dataset contains only numerical data then the classification process can be carried out without any difficulty. However, the features need to be seen if they are suitable for classification. If the data are in other formats like ordinal or categorical, then the data need to be converted to another form for classification. The popular methods for classification are Naïve Bayes, support vector machines, and decision trees [1, 2]. These methods of classification are discussed in this chapter with examples.

10.1 Naïve Bayes

Classification machine learning technique consists of different methods. Naïve Bayes is one of the popular classification methods that are based on the conditional probability [1]. In this section, the basics of Naïve Bayes are discussed with its mathematical background and examples.

10.1.1 Introduction

It is one of the classification techniques based on conditional probability. It uses Bayes theorem for determining the relationship between the probabilities of two events [3]. Let us consider an example where a flower can be classified based on color and shape. A Naïve Bayes classifier considers such features in a dataset for classification based on Bayes theorem. It assumes that all the features are independent and contribute to the probability of classification.

The input variables to the Naïve Bayes classifier are usually categorical. For example, for a flower dataset, one of the features color is categorical. The values it can take are red, yellow, blue, orange, green, etc., which are categorical in nature. However, if the features are continuous, it can be converted to categorical using the method of discretization and then used for classification. For example, in the dataset of salary where the values are numeric such as 10,000, 50,000, it can be converted to categorical values as shown below:

Low salary: salary < 10,000,
 Middle class: 10,000 < salary < 20,000,
 Upper class: 25,000 < salary < 50,000.

The output of a classifier is `<class, probability_score>` where `class` specifies the `class_label` to which the data point belongs to with probability score. The score is not the actual true probability but includes the log probability based on the highest values.

10.1.2 Bayes Theorem

Naïve Bayes classifier is based on Bayes theorem which is as shown in Eq. 10.1.2. The components of the Bayes theorem are listed as follows:

- A and B are random two events that occur.
- $P(A)$: The probability that event A occurs.
- $P(B)$: The probability that event B occurs.
- $P(A|B)$: The conditional probability of occurrence of A given that the condition B occurs. It is also referred to as posterior probability.
- $P(B|A)$: The conditional probability of occurrence of B given that the condition A occurs. It is also referred to as likelihood.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (10.1.2)$$

Example on Bayes Theorem

In this section, an example on how Naïve Bayes classification works is shown with Bayes theorem. Consider a scenario where there are three factories (A, B, C) where batteries are manufactured. One of such battery is found to be defective. A Control Manager (CM) is responsible for investigating the source of found defects. The total percentage of batteries manufactured by A, B, C are 35, 35, and 30% as shown in Table 10.1. The percentage of defective batteries from A, B, C are 15, 10, and 20%. The CM would like to answer the following question: If a randomly selected battery is defective, what is the probability that the battery was manufactured in factory C ?

Table 10.1 Bayes theorem problem probability distribution

Factory	% Total production	Probability of defective lamps (likelihood probabilities)
A	$P(A) = 0.35$	$P(D A) = 0.15$
B	$P(B) = 0.35$	$P(D B) = 0.10$
C	$P(C) = 0.30$	$P(D C) = 0.20$

The answer to the problem is to find posterior probability $P(C|D) = ?$

$$P(C|D) = \frac{P(D|C)P(C)}{P(D)}$$

$$\begin{aligned} P(D) &= P(D|A) \cdot P(A) + P(D|B) \cdot P(B) + P(D|C) \cdot P(C) \\ &= (0.15)(0.35) + (0.10)(0.35) + (0.20)(0.30) \\ &= 0.0525 + 0.035 + 0.06 \\ &= 0.1475 \end{aligned}$$

$$P(C|D) = \frac{P(D|C)P(C)}{P(D)}$$

$$\begin{aligned} P(C|D) &= \frac{(0.20)(0.30)}{0.1475} \\ &= 0.407 \end{aligned}$$

Similarly,

$$\begin{aligned} P(A|D) &= \frac{P(D|A)P(A)}{P(D)} = 0.356 \\ P(B|D) &= \frac{P(D|B)P(B)}{P(D)} = 0.237 \end{aligned}$$

Since the percentage of likelihood of total production of batteries that are defective are greater for factory ‘C’, the posterior probability is more for factory ‘C’ than the others.

10.1.3 Naïve Bayes Classification in Python

In this section, an example on Naïve Bayes classification is shown with the example on iris dataset. Iris dataset consists of different species of flowers with different shapes, color, height, and width. Firstly, the modules required for classification are imported. The modules imported are `load_iris()`, `GaussianNB()`, `accuracy_score()`, and `train_test_split()`.

- load_iris(): It is used to the load the iris dataset.
- GaussianNB(): It is used for Naïve Bayes classification. It internally uses the Bayes theorem for classification.
- accuracy_score(): It is used for determining the accuracy score of the model.
- train_test_split(): It is used for splitting the training and testing dataset.

The dataset is loaded using load_iris() method and converted to a list of data and target values. Then, the dataset is split into training and test dataset using the train_test_slpit() method. The number of training instances and testing instances is printed out the console for differentiating purpose.

The split dataset is used for selecting the features and labels for building the Naïve Bayes classifier. The classifier is built using GaussianNB() function. The fit() function will fit the classifier to the trained dataset with trained features and labels. Finally, the accuracy of the model is calculated using the accuracy_score() function.

```
from sklearn.datasets import load_iris
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
#Load the dataset
data = load_iris()
dataset = list(zip(data.data, data.target))
#Split the dataset into train and test
train , test = train_test_split(dataset, test_size=0.25)
print('The training set consists of '+ str(len(train)) +'points')
print('The test set consists of '+ str(len(test)) +'points')
#To build the model select the features and labels
train_features = []
train_labels = []
test_features = []
test_labels = []
for item in train:
    train_features.append(item[0])
    train_labels.append(item[1])
for item in test:
    test_features.append(item[0])
    test_labels.append(item[1])
# Naive Bayes model
classifier = GaussianNB()
classifier.fit(train_features, train_labels)
results = classifier.predict(test_features)
```

```
#Accuracy of the model
print("model accuracy",accuracy_score(test_labels, results))
Output:
The training set consists of 112 points
The training set consists of 38 points
model accuracy 0.947368421053
```

10.1.4 Diagnostics for Improvement of Classifier

In the previous section, an example code on implementing Naïve Bayes classification was shown. Once the model of classifier is implemented, necessary metrics should be used for validation of the model. In this section, various diagnostics for finding the accuracy of the classifier are discussed. These diagnostics can be used for any method of classification in general.

Confusion Matrix

A confusion matrix is a visualization table for evaluating the performance of a classifier. The confusion matrix for a two-class classifier is as shown in Table 10.2. The matrix consists of the following terms:

- True positives (TPs): Number of instances identified as positive from the classifier.
- False positives (FP): Number of instances identified as positive from the classifier but actually they are negative.
- True negatives (TNs): Number of instances identified as negative from the classifier and actually they are negative as well.
- False negatives (FNs): Number of instances identified as negative from the classifier and actually they are positive.

In general, the confusion matrix of a classifier should have more number of TP and TN and less number of FP and FN.

Using the confusion matrix, other metrics can be used in the evaluation of model. These metrics are discussed as below.

- **Accuracy:** It is the measure that defines the rate at which classification has been carried by the model. It is defined as shown in Eq. 10.1.4.1.
- **True-positive rate (TPR):** It is the measure that defines the rate of correct classification of the model. It is defined as shown in Eq. 10.1.4.2.
- **False-positive rate (FPR):** It is the measure that defines the rate of false positives made by the classifier as shown in Eq. 10.1.4.3.
- **False-negative rate (FNR):** It is the measure that defines the rate of false negatives made by the classifier as shown in Eq. 10.1.4.4.

Table 10.2 Confusion matrix for a classifier

	Predicted class	
Actual class	Positive	Negative
Positive	TP	FN
Negative	FP	TN

- **Precision:** It is the measure as shown in equation that gives the percentage of data points that were predicted as positive as shown in Eq. 10.1.4.5.
- **Recall:** It is equivalent to TPR that gives the percentage of data points that were predicted correctly.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (10.1.4.1)$$

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{TN}} \quad (10.1.4.2)$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad (10.1.4.3)$$

$$\text{FNR} = \frac{\text{FN}}{\text{FN} + \text{TP}} \quad (10.1.4.4)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (10.1.4.5)$$

Example with Metrics on Classification

A small example of the confusion matrix on two variables ‘y_true’ and ‘y_pred’ is built using the method confusion matrix from the sklearn library as shown in the code. Here the labels are used in building the confusion matrix. The output displays the confusion matrix for the predicted values from ‘y_pred’. Table 10.3 displays how the confusion matrix is built based on the comparison of the actual values and predicted values. The rows have labels in the order of elephant, peacock, and tiger, and rows represent the actual values. The columns have labels in the same order as well and represent the predicted values.

Firstly, the actual value ‘tiger’ and ‘elephant’ is compared with each other. Since, they are mismatched confusion matrix $[3][1] = 1$. Similarly, in the next iteration value, ‘elephant’ and ‘elephant’ are compared with each other. Since, they are matched confusion matrix $[1][1] = 1$. On the same note, values ‘tiger’ and ‘tiger’ are compared with each other in the next iteration. Since, they are matched confusion matrix $[3][3] = 1$. For other actual and predicted values, comparison is done on the same basis and confusion matrix is updated.

Finally, the output represents the confusion matrix where there are four correct predictions and two wrong predictions. The two wrong predictions are in iterations 1 and 6. Hence, the entries of the confusion matrix help in determining the rate of classification.

Table 10.3 Confusion matrix building

Actual value	Predicted value	Confusion matrix
Tiger	Elephant	0 0 0 0 0 0 1 0 0
Elephant	Elephant	1 0 0 0 0 0 1 0 0
Tiger	Tiger	1 0 0 0 0 0 1 0 1
Elephant	Elephant	2 0 0 0 0 0 1 0 1
Tiger	Tiger	2 0 0 0 0 0 1 0 2
Peacock	Tiger	2 0 0 0 0 1 1 0 2

Code:

```
from sklearn.metrics import confusion_matrix
y_true = ["tiger", "elephant", "tiger", "tiger", "elephant", "peacock"]
y_pred = ["elephant", "elephant", "tiger", "tiger", "elephant", "tiger"]

print(confusion_matrix(y_true, y_pred, labels=["elephant",
"peacock", "tiger"]))
```

Output:

```
[[2 0 0]
 [0 0 1]
 [1 0 2]]
```

10.1.5 Case Study on Naïve Bayes Classification

In this section Naïve Bayes classification is carried out on a course_enrollment dataset as shown in the Table 10.4. In this example dataset, the features considered are age, income, job satisfaction, desire and enrolls. A decision has to be made whether a person enrolls for a course or not using the data features available. In the dataset, last row of the data are left blank to predict the enrollment.

Table 10.4 Course_enrollment dataset

Age	Income	Job satisfaction	Desire	Enrolls
<=30	High	No	Fair	No
<=30	High	No	Excellent	No
31–40	High	No	Fair	Yes
>40	Medium	No	Fair	Yes
>40	Low	Yes	Fair	Yes
>40	Low	Yes	Excellent	No
31–40	Low	Yes	Excellent	Yes
<=30	Medium	No	Fair	No
<=30	Low	Yes	Fair	Yes
>40	Medium	Yes	Fair	Yes
<=30	Medium	Yes	Excellent	Yes
31–40	Medium	No	Excellent	Yes
31–40	High	Yes	Fair	Yes
>40	Medium	No	Excellent	No
<=30	Medium	Yes	Fair	

Firstly the modules needed for the classification are imported as follows. The modules imported are GaussianNB(), accuracy_score(), train_test_split(), confusion_matrix(), LabelEncoder(), numpy, and pandas().

- GaussianNB(): It is used for Naïve Bayes classification. It internally uses the Bayes theorem for classification.
- accuracy_score(): It is used for determining the accuracy score of the model.
- train_test_split(): It is used for splitting the training and testing dataset.
- confusion_matrix(): It is used to print the confusion matrix to determine the rate of classification.
- LabelEncoder(): In the dataset used, the course enrollment is a categorical variable and needs to be converted to numeric variable for classification. LabelEncoder is used for the same purpose.

Firstly, the required modules are imported using sklearn. The data are read through read_csv() function. Since the last row needs to be removed from the training set, numpy is used to convert into a list of values and ignoring the last row. The categorical variable ‘Enrolls’ is converted to integer value 0 for ‘No’ and 1 for ‘Yes.’

Then, the dataset is split into training and test dataset using the train_test_split() method. The number of training instances and testing instances is printed out the console for differentiating purpose.

The split dataset is used for selecting the features and labels for building the Naïve Bayes classifier. The classifier is built using GaussianNB() function. The fit() function will fit the classifier to the trained dataset with trained features and labels. Finally, the accuracy of the model is calculated using the accuracy_score() function.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

#Read the data
data=pd.read_csv("NBOrDT.csv")
print(data)
#Neglect last row as
output label is not defined Select columns from income to enroll
dataset=np.array(data.loc[:len(data)-2,['Income':'Enrolls']])
print(dataset)

#Since my dataset contains strings (categorical data), encode
the strings to integers to proceed.
# For this use LabelEncoder

le=LabelEncoder()
for i in range(4):
    le.fit(dataset[:,i])
    dataset[:,i]=le.transform(dataset[:,i])

#Extract the features and labels now
X=dataset[:,3]
Y=dataset[:,3]
dataset=list(zip(X,Y))

#Split the dataset into train and test
train , test = train_test_split(dataset, test_size=0.25)
print('The training set consists of '+ str(len(train))+'points')
print('The test set consists of '+ str(len(test))+'points')

#To build the model select the features and labels
train_features = []
train_labels = []
test_features = []
test_labels = []
for item in train:
    train_features.append(item[0])
    train_labels.append(item[1])
for item in test:
    test_features.append(item[0])
    test_labels.append(item[1])
```

```
# Naive Bayes model
classifier = GaussianNB()
classifier.fit(train_features, train_labels)
results = classifier.predict(test_features)

#Accuracy of the model
print("model accuracy",accuracy_score(test_labels, results))
print(confusion_matrix(test_labels, results))
```

Output:

```
The training set consists of 10 points
The test set consists of 4 points
model accuracy 0.75
[[0 1]
 [0 3]]
```

10.2 Decision Tree

A decision tree employs tree structure for classification. It consists of a set of decisions and its outcomes. The main goal of a decision tree is to predict the response for a given input variable $X = \{x_1, x_2, x_3\dots x_n\}$ where $x_1, x_2, x_3\dots x_n$ are the features of the input data [4]. A decision tree mainly consists of set of test points and branches. A set of conditions or test points is used to predict the branches in the decision tree. Essentially, the test point refers to the decision condition and the branch refers to the decision being made. They can be easily visualized in the form of if-else condition as there is no underlying relationship between the input variables.

There are two basic varieties of decision trees, namely classification trees and regression trees. Classification trees are used for categorical variables and regression trees are used for continuos (numeric) variables. An example of classification tree can be ‘yes or no’ purchase for customer based on the wallet amount available. In the situations where ‘likelihood of subscription’ is required, regression trees are used. In this section, a brief overview of decision tree is discussed.

10.2.1 Basic Elements of Decision Tree

The basic elements of a decision tree can be better understood with the help of an example as shown in Fig. 10.1. The example considered is the prediction of rain based on the features ‘cloudy’ and ‘pressure’.

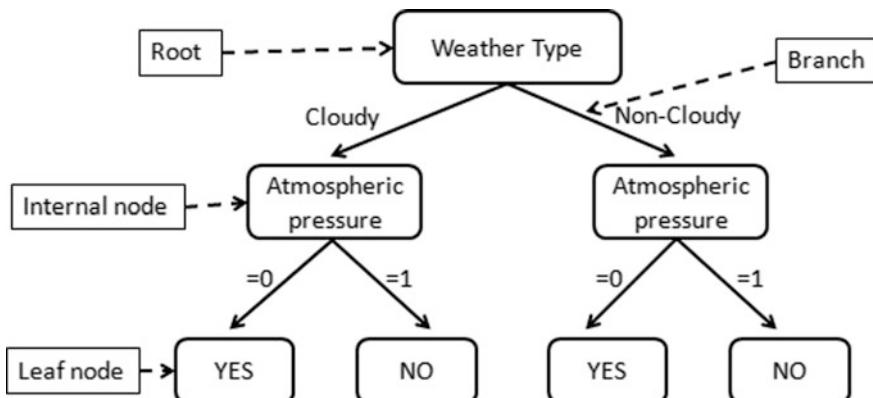


Fig. 10.1 Example of a decision tree

- **Branch:** It is a line between two nodes that refer to the decision being made. In Fig. 10.1, the branch is ‘cloudy’ or ‘non-cloudy.’
- **Internal node:** It is the decision point in the tree that refers to the feature of the input. The topmost internal node is called as root. For example in Fig. 10.1, atmospheric pressure is an internal node and input feature.
- **Leaf nodes:** They are the nodes present at the end of the tree that refers to the class or the outcome of the decision made in the internal nodes.
- **Depth:** It is the minimum number of steps needed to traverse from root to leaf node. For example, ‘Atmospheric pressure’ is at a depth of 1.

10.2.2 Decision Tree Classification Example in Python

An example on the classification using decision tree is discussed in this section. The example is based on the weather analysis where the decision to be decided is ‘rain’ or ‘not rain’ depending on the conditions of ‘weather type’ and ‘atmospheric pressure.’

Firstly the modules needed for the classification are imported. The modules imported are numpy, collections, tree, pydotplus, and graphviz.

- **Tree:** It is main module that builds the decision tree in ‘dot’ file format.
- **Numpy:** It is used to convert the decision variable ‘rain’ into an array of values.
- **Collections:** It is used to draw the edges of the decision tree.
- **Pydotplus:** It is used to convert the decision tree in ‘dot’ file format into graph form.
- **Graphviz:** It is the module used for the visualization of the decision tree in graphical way using the graph file obtained from pydotplus.

Here ‘X’ represents a list of values representing the weather_type and atmospheric_pressure with values from (0,2). The DecisionTreeClassifier() method is used to build the decision tree based on the data feature names. The fit() method is used to fit the decision tree using X and rain features. These steps will build a decision tree in non-graphical way.

In the next steps, graphviz is used to convert the decision tree into ‘dot’ format file. This ‘dot’ file is again converted to graph form using graphviz using graph_from_dot_data()method. The last part of the code with two for loops helps in drawing the exact graph from the dot file based on the edges created and its associated root nodes.

```
import numpy as np
import collections
import pydotplus
from sklearn import tree
import graphviz

# Data Collection
X = [ [0,0],
      [1,0],
      [1,1],
      [2,1],
      [2,1],
      [2,0] ]

rain = np.array(['not rain', 'not rain', 'not rain', 'rain', 'rain', 'not
rain'])

data_feature_names = [ 'weather_type', 'atmospheric_pressure']

# Training
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X,rain)

# Visualize data
dot_data = tree.export_graphviz(clf,
                                feature_names=data_feature_names,
                                out_file=None,
                                filled=True,
                                rounded=True)
graph = pydotplus.graph_from_dot_data(dot_data)

colors = ('turquoise', 'orange')
edges = collections.defaultdict(list)
```

```

for edge in graph.get_edge_list():
    edges[edge.get_source()].append(int(edge.get_destination()))

for edge in edges:
    edges[edge].sort()
    for i in range(2):
        dest = graph.get_node(str(edges[edge][i]))[0]
        dest.set_fillcolor(colors[i])

graph.write_png('tree.png')

```

In Fig. 10.2, we can see the root node is atmospheric_pressure and number of samples is six. In these six samples, three are classified as true and other three as false. The three samples that are classified as false are further split based on the value of weather_type ≤ 1.5 . Using this as the decision, two samples are classified as false and one sample is classified as true. Using this approach, a decision tree can be built for analysis. The gini value in the decision tree represents the information gain for the decision node in the tree.

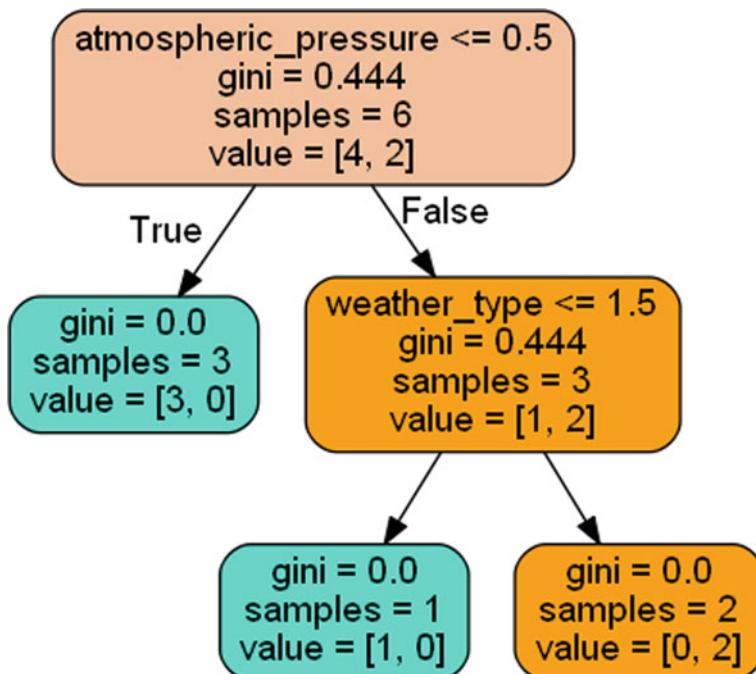


Fig. 10.2 Decision tree example for weather in Python

10.2.3 Decision Tree Classification on Iris Dataset

In this section, decision tree classification on iris dataset is discussed. Iris dataset consists of 150 instances with three classes, namely Setosa, Versicolor, and Virginica. It consists of four features, namely sepal width, sepal length, petal width, and petal length. The main idea is to use these features and build a decision tree model for classification of species. Firstly, the modules required for building the decision tree are as follows:

- Load_iris: It is used to load the iris dataset from the datasets in sklearn.
- DecisionTreeClassifier: It is used to build the decision tree using the tree module.
- Graphviz: It is used to visualize the decision tree of the classifier.

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import graphviz
```

The iris data are loaded using load_iris() method and converted to a list of data and target features. The data is then split into training and test features. The training features used here are sepal width, sepal length, petal length, and petal width. Next, the decision tree classifier is built and model is fitted into the test dataset.

```
data = load_iris()
dataset = list(zip(data.data, data.target))
train , test = train_test_split(dataset, test_size=0.25)
print(len(train))
print(len(test))
train_features = []
train_labels = []
test_features = []
test_labels = []
for item in train:
    train_features.append(item[0])
    train_labels.append(item[1])
for item in test:
    test_features.append(item[0])
    test_labels.append(item[1])
print(train_features, train_labels, test_features, test_labels)
classifier = DecisionTreeClassifier()
classifier.fit(train_features, train_labels)
results = classifier.predict(test_features)
print(accuracy_score(test_labels, results))
```

The decision tree classifier build using `DecisionTreeClassifier()` module produces the decision tree in dot format. The dot format file is a raw format and needs to be converted to a graphical form for visualization. `Graphviz` module is used to accomplish this where it takes the list of feature names and target names as the class names.

```
with open('iris_dt.dot', 'w') as f:  
    export_graphviz(classifier, out_file=f,  
                    feature_names=data.feature_names,  
                    class_names=list(data.target_names))  
    with open('iris_dt.dot', 'r') as f:  
        dot_graph = f.read()  
        graphviz.Source(dot_graph).view()
```

The output of the decision tree with `graphviz` module is as shown in Fig. 10.3. In this code, out of 150 samples of iris dataset, 112 are used for training purpose and 38 are used for testing to see which class it belongs. In Fig. 10.3, we can observe that 112 samples are at the root node level and decision is `petal_width <= 0.8`. Based on this decision, 38 of them are classified as true and 74 of them are classified as false. Now for the false classified samples, the next decision node is `petal_width <= 1.75`. For the 74 samples, 40 of samples are classified as true and 34 of them are classified as false and then the next decision is taken further on petal length and so on. In this way, the decision tree is classified and in the last level, the leaf nodes specify the classes virginica and versicolor. The other classes are not included in the test dataset and thus only these two classes are present.

10.3 Support Vector Machine (SVM)

SVM s are used for classification problems, and it is a supervised learning technique. The idea of supervised learning where the data are split into training and testing is also carried out in SVM as well. SVMs are generally used for Image classification, healthcare to identify multiple objects in a dataset [5]. There are two types of classifiers that can be build with SVM, namely linear classifier and non-linear classifier. In this section, a brief overview of linear classifiers with SVM and an example in Python is discussed.

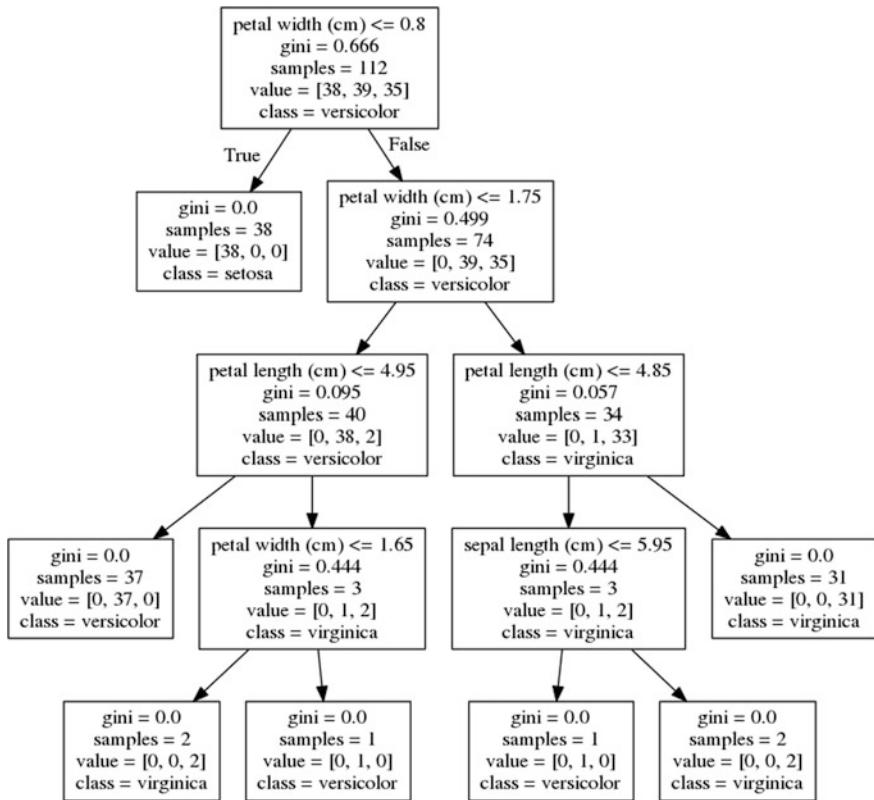


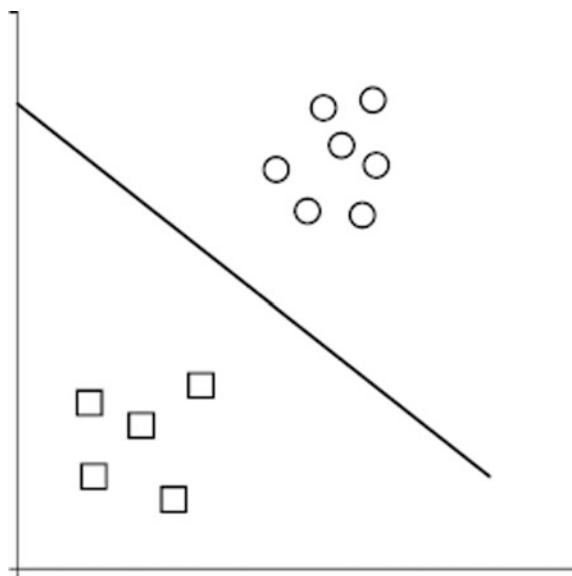
Fig. 10.3 Decision tree model on IRIS dataset

10.3.1 Linear Classifiers

A linear classifier divides the objects along a neat line called as hyperplane identifying the locations of the objects in the hyperplane. The hyperplane essentially group one set of objects on one side of plane and other set of objects on the opposite side of the plane. Suppose consider an example of a category of birds that can be represented visually as shown in Fig. 10.4. The circles represent one category, and triangles represent another category.

If we need to determine mathematically the category of the birds, then it can take the form $\text{sign}(ax + by + c)$, where a, b, c are values representing the line in the Fig. 10.4. Each data point (bird category) has specific x -axis value and y -axis value. The main goal here is to have clear separation of categories of birds. So, if the sign is $+1$, then it belongs to one side of the hyperplane and sign of -1 indicates that it belongs to the opposite side of the hyperplane. A good fit of the line can be obtained by tweaking the values of a, b, c .

Fig. 10.4 SVM with hyperplane



10.3.2 Non-linear Classifiers

The datasets in ideal world can be classified based on the linear classification where a certain set of objects belong to one side of the plane and other set of objects belong to the opposite side of the plane. However, there are cases where there are data points where a hyperplane cannot be used for classification. In such cases, nonlinear classifiers have to be used (Fig. 10.5).

Nonlinear classifiers are used with kernel tricks. Here, the kernel trick refers to hyperbolic functions that can be used for classification. The most commonly used kernel in SVM is radial basis function (RBF) for classification. Since the focus of the chapter is on classification, it goes beyond the scope of this book to go in depth about the kernel tricks for classification. However, in the next section, a case study on SVM with iris dataset shows how RBF kernel can be used for classification.

10.3.3 SVM Case Study on Iris Dataset

In this section, SVM is discussed with iris dataset as the example [6]. The main aim here is to differentiate the classes of iris dataset with SVM and how SVM visualization is easier than the decision tree. Firstly, the modules required for building SVM are as follows:

- Load_iris: It is used to load the iris dataset from the sklearn datasets.
- SVC: It is the support vector classifier that is used for building SVM.

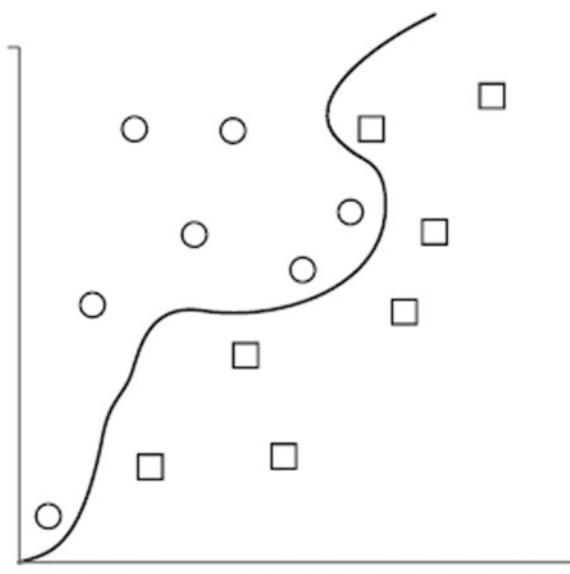


Fig. 10.5 Hyperplane does not divide

Other modules are same as in the case of decision tree and Naïve Bayes. The iris dataset is then split into training and test dataset as in the case of decision tree classifier.

```
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np

# load the dataset.
data = load_iris()
dataset = list(zip(data.data[:, :2], data.target))
print(len(dataset))
#print(dataset)
train, test = train_test_split(dataset, test_size=0.30)
train_features = []
train_labels = []
test_features = []
test_labels = []
for item in train:
    train_features.append(item[0])
    train_labels.append(item[1])
for item in test:
    test_features.append(item[0])
    test_labels.append(item[1])
```

```
train_labels.append(item[1])
for item in test:
    test_features.append(item[0])
    test_labels.append(item[1])
#print(train_features, train_labels, test_features, test_labels )
```

SVM is built for the iris dataset using the code as shown below. The kernel used here is rbf because there are three classes of iris dataset. The fit() function is used to fit the SVM model using the features sepal width and sepal length. Based on these features, the iris data are classified into three classes as shown in Fig. 10.6.

```
#train the model.
model = SVC(kernel='rbf')
model.fit(train_features,train_labels)
results = model.predict(test_features)

#visualization.

train_features = data.data[:, :2]
train_labels = data.target
x_min, x_max = train_features[:, 0].min() - 1 ,
train_features[:, 0].max() + 1
```

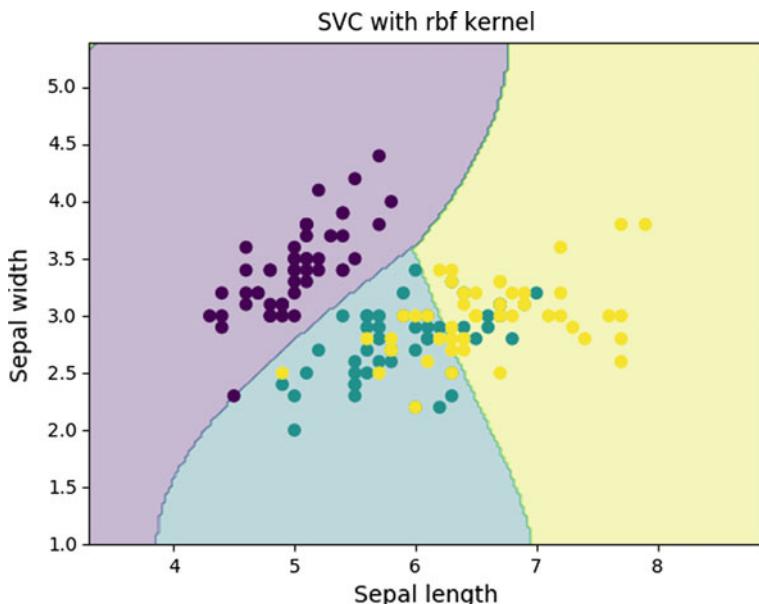


Fig. 10.6 SVM on iris dataset

```

y_min, y_max = train_features[:, 1].min() - 1,
train_features[:, 1].max() + 1
step = (x_max / x_min) / 100
mesh_x, mesh_y = np.meshgrid(np.arange(x_min, x_max, step),
np.arange(y_min, y_max, step))
plot_features = np.c_[mesh_x.ravel(), mesh_y.ravel()]

Z = model.predict(plot_features)
Z = Z.reshape(mesh_x.shape)

plt.contourf(mesh_x, mesh_y, Z, cmap=plt.cm.tab10, alpha=0.3)
plt.scatter(train_features[:, 0], train_features[:, 1],
c = train_labels)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(mesh_x.min(), mesh_x.max())
plt.title('SVC with rbf kernel')

```

The output of plot() function for visualization of the SVM for iris dataset is as shown in Fig. 10.6. It can be observed in Fig. 10.6 that the plane is divided into three classes based on the features sepal width and sepal length. It can also be seen that it is easy to identify the classes with SVM than decision tree when there are multiple classes in a dataset.

10.4 Exercises

1. Which classifier is computationally efficient to address the curse of dimensionality?
2. If the dataset consists of more categorical data, which classifier would be chosen?
3. Build a Naïve Bayes classifier using the dataset for breast cancer provided by sklearn in Python.
4. Identify the difference in approach for various split criterion like information gain (entropy) and gini impurity.
5. Can a decision tree be used for regression problems?
6. Build a decision tree classifier using the dataset for wine classification of sklearn in Python.
7. What are support vectors? How does the SVM classify data points?
8. Compare Naïve Bayes and decision tree classification.

References

1. Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging Artificial Intelligence Applications in Computer Engineering*, 160, 3–24.
2. Kotsiantis, S. B., Zaharakis, I. D., & Pintelas, P. E. (2006). Machine learning: A review of classification and combining techniques. *Artificial Intelligence Review*, 26(3), 159–190.
3. Manning, C. D., Raghavan, P., & Schütze, H. (2008). Text classification and Naive Bayes. *Introduction to Information Retrieval*, 1, 6.
4. Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3), 660–674.
5. Steinwart, I., & Christmann, A. (2008). *Support vector machines*. Springer Science & Business Media.
6. Moore, A. W. (2001). Support vector machines. *Tutorial. School of Computer Science of the Carnegie Mellon University*. Available at <http://www.cs.cmu.edu/~awm/tutorials>. Accessed August 16, 2009.

Chapter 11

Other Analytical Techniques



Machine learning methods discussed in the previous chapters such as regression and classification are the basic techniques used for prediction. However, there are other techniques such as clustering, association rule mining, principal component analysis (PCA), and others for machine learning. These techniques are used for the classification and regression as well. For example, random forest is one of the machine learning techniques where both classification and regression can be carried out. In this chapter, such machine learning techniques are discussed with examples.

11.1 Clustering

Clustering is unsupervised technique used for grouping similar objects. In the initial phase of clustering that is in the first phase of the analysis, the number of clusters is not known for the developer. The number of clusters depends on the structure of the data as it describes the best number for grouping. Here, the structure of the data is the attributes of the dataset, and they are grouped together based on the similarities between them [1]. In general, clustering is used for exploratory analysis of data, and no predictions are carried out. In this section, a brief overview of clustering is presented.

11.1.1 K-Means

One of the best-known algorithms used for clustering is *K*-means [2]. In *k*-means clustering method, *k*-clusters are identified for a given collection of *n* objects and a value of *k*. The objects are clustered based on the mean distance between the objects and its group's centers. In this section, *k*-means algorithm is discussed with steps and example.

The k -means algorithm to find k -clusters can be described in the following four steps.

1. Choose the value of k and the k initial guesses for the centroids.
2. Assign each data point to a cluster by computing the mean distance from each data point (x_i, y_i) to the centroid. The distance between any two data points (x_1, y_1) and (x_2, y_2) is calculated using Euclidean distance metric as shown in Eq. 11.1.1.1.

$$d = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \quad (11.1.1.1)$$

3. For each k -cluster defined in step 2, compute the center of mass, i.e., centroid calculated as shown in Eq. 11.1.1.2. The pair (x_c, y_c) is arithmetic means of m points in the cluster.

$$(x_c, y_c) = \left(\frac{\sum_{i=1}^m x_i}{m}, \frac{\sum_{i=1}^m y_i}{m} \right) \quad (11.1.1.2)$$

4. Repeat steps 2 and 3 until the algorithm reaches convergence. The convergence is reached where the centroids do not change or the number of clusters remains the same, i.e., the centroids computed should not oscillate back and forth during the iterations.

11.1.2 Elbow Curve Using Within Sum of Squares

In the k -means algorithm, the value of k has to be chosen in the first step. The computation time depends on the value of k chosen as initially whether $k + 1$ or $k - 1$ clusters' possibility is not known. Within sum of squares (WSS) is one of the metric that can be used for determining the optimal number of k . WSS is defined as shown in Eq. 11.1.2.

WSS is the sum of squares of distances between the data points and the centroid. In Eq. 11.1.2, p represents the data points in the cluster and q represents the centroid in M number of clusters. It indicates the centroid that is closer to the data point p_i . WSS is small if the points are close to the centroids. So, for $k + 1$ clusters if WSS do not reduce drastically, optimal number of clusters would be k .

$$\text{WSS} = \sum_{i=1}^M d(p_i, q^j)^2 \quad (11.1.2)$$

Elbow Curve Example

In this section, a small example on determining the optimal number of clusters using elbow curve is shown. The modules required for running clustering in Python are as follows:

- *K*-means: It is the module that uses *k*-means algorithm for clustering.
- Numpy: It is the module required for initializing the array values for clustering.
- Cdist: It is the module that computes the Euclidean distance between the data points and the centroid.
- Matplotlib: It is the module required for plotting the data points and clusters.

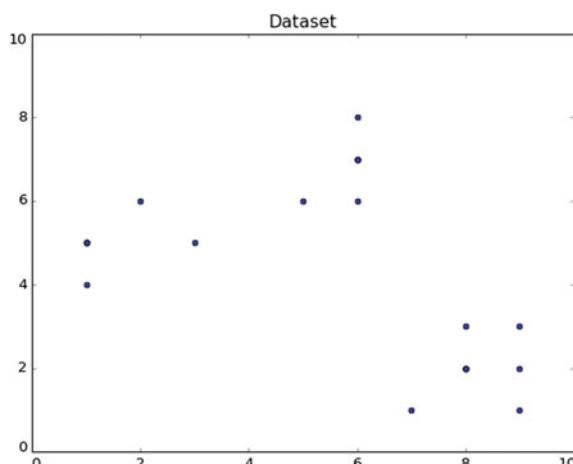
Firstly, the modules are imported and x_1 and x_2 data are initialized with the data values. The plot() function is used to plot the data points required for the cluster. It uses xlim() and ylim() to restrict the limits of the x -axis and y -axis, respectively. The vector X stores the values of both x_1 and x_2 . The output of the plot() function is as shown in Fig. 11.1.

```
#import statements
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt

#input values
x1 = np.array([3, 1, 1, 2, 1, 6, 6, 6, 5, 6, 7, 8, 9, 8, 9, 9, 8])
x2 = np.array([5, 4, 5, 6, 5, 8, 6, 7, 6, 7, 1, 2, 1, 2, 3, 2, 3])

#plot function to plot the dataset
plt.plot()
plt.xlim([0, 10])
```

Fig. 11.1 Data plot for clustering



```

plt.ylim([0, 10])
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()

# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

```

WSS method is used to determine the number of clusters with the code as shown below. First the k-means module is run on the data vector X and fitted into the kmean model. The clustering procedure is run through 10 iterations using the range (1,10) function. In each iteration, the centroid value is calculated using Euclidean distance measure and appended to the list wss[].

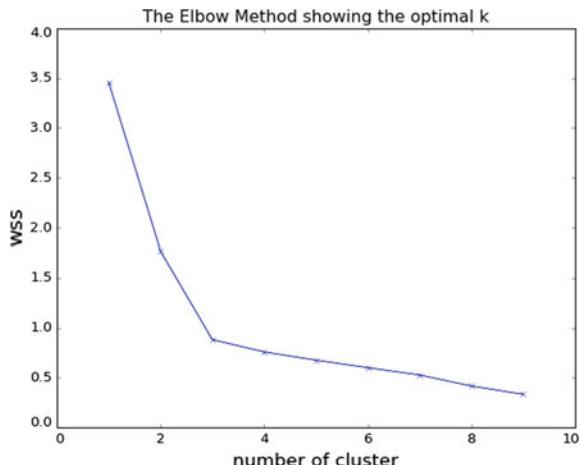
This wss[] list is used to plot the elbow curve as shown in Fig. 11.2. In Fig. 11.2, it can be observed that wss value starts to descend from $k = 3$ onward. From $k = 4,5,6$ onward, there is no much change in the value of wss. Thus, the number of clusters can be $k = 3$ for this example. In this way, the number of clusters is determined first for K -means clustering problems.

```

# WSS to determine k
wss = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X)

```

Fig. 11.2 Elbow curve for clustering



```

kmeanModel.fit(X)
wss.append(sum(np.min(cdist(
(X, kmeanModel.cluster_centers_, 'euclidean'), axis=1)) / X.shape[0]))

# Plot the elbow
plt.plot(K, wss, 'bx-')
plt.xlim([0, 10])
plt.ylim([0, 4])
plt.xlabel('number of cluster')
plt.ylabel('wss')
plt.title('The Elbow Method showing the optimal k')
plt.show()

```

11.1.3 Clustering Example

In the previous section, elbow curve method for determining the number of clusters was discussed but not the actual clustering. In this section, an example on clustering and the plot of it is discussed. The modules required for running this example in Python are as follows:

- *K*-means: It is the module that uses *k*-means algorithm for clustering.
- Numpy: It is the module required for initializing the array values for clustering.
- Cdist: It is the module that computes the Euclidean distance between the data points and the centroid.
- Matplotlib: It is the module required for plotting the data points and clusters.

Firstly, the modules are imported, and x_1 and x_2 data are initialized with the data values. The plot() function is used to plot the data points required for the cluster. It uses xlim() and ylim() to restrict the limits of the x -axis and y -axis, respectively. The vector X stores the values of both x_1 and x_2 . The output of the plot() function is as shown in Fig. 11.3. WSS method is used to determine the number of clusters with the code as shown below. First the *k*-means module is run on the data vector X and fitted into the *kmean* model. The clustering procedure is run through ten iterations using the range(1,10) function. In each iteration, the centroid value is calculated using Euclidean distance measure and appended to the list $wss[]$.

This $wss[]$ list is used to plot the elbow curve as shown in Fig. 11.4. In Fig. 11.4, it can be observed that wss value starts to descend from $k = 3$ onward. From $k = 4,5,6$ onward, there is no much change in the value of wss . Thus, the number of clusters can be $k = 3$ for this example.

```

from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt

```

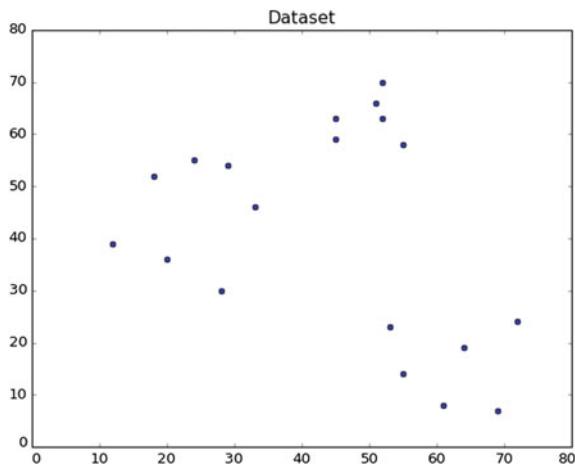
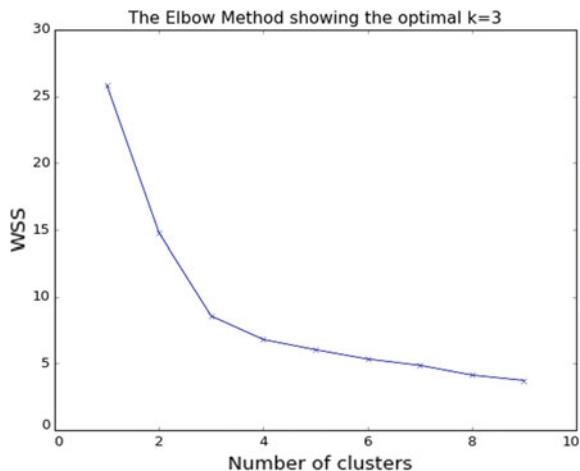


Fig. 11.3 Dataset for clustering example

Fig. 11.4 Elbow curve for clustering example



```
x1 = np.array([12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72])
x2 = np.array([39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24])

plt.plot()
plt.title('Dataset')
plt.scatter(x1, x2)
plt.show()
```

```

# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

wss = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans (n_clusters=k).fit(X)
    kmeanModel.fit(X)
    wss.append(sum(np.min(cdist(
        X, kmeanModel.cluster_centers_, 'euclidean'), axis=1)) / X.shape[0])

# Plot the elbow
plt.plot(K, wss, 'bx-')
plt.xlim([0, 10])

plt.xlabel('Number of clusters')
plt.ylabel('WSS')
plt.title('The Elbow Method showing the optimal k=3')
plt.show()

```

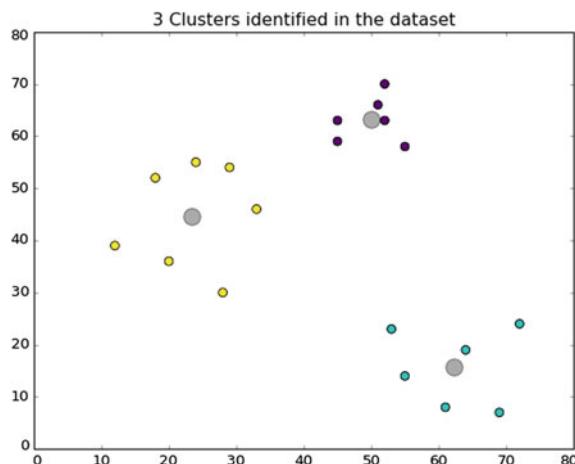
From Fig. 11.4, it can be observed that $k = 3$. Using $k = 3$ number of clusters, k -means algorithm is now executed to see the clusters in the dataset. The function k-means() is used to accomplish this purpose, and fit() function is used to fit the k -means model from it. The scatter() function is used to plot the centers of the clusters with the help of kmeanMode.cluster_centers_. The three clusters can be observed in Fig. 11.5 represented with colors yellow, blue, and purple. The gray-colored center represents the centroid for each cluster. Hence, in this way the clustering can be done first by examining the elbow curve and then plotting the number of clusters.

```

#Clustering
kmeanModel = KMeans(n_clusters=3).fit(X)
kmeanModel.fit(X)
y_kmeans = kmeanModel.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeanModel.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.title('3 Clusters identified in the dataset')
plt.show()

```

Fig. 11.5 Clustering example



11.1.4 Clustering Case Study on Student Marks

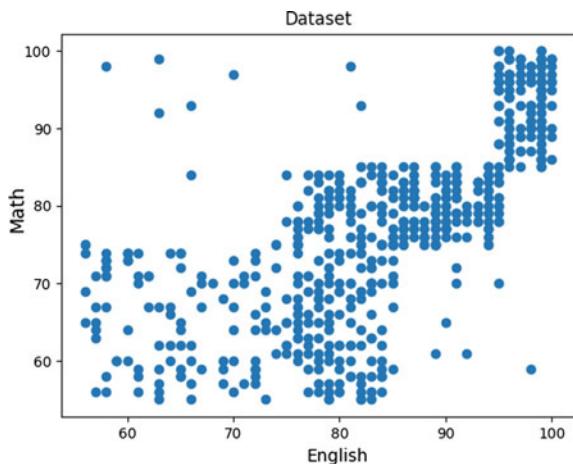
The example discussed in the previous section was on a random dataset for clustering. In this section, a case study on clustering for students and their marks is carried out. The dataset considered for this case study is as shown in Table 11.1. It shows only few rows of the dataset. The dataset consists of 619 students and their marks in English, Math, and Science. The goal of this case study is to present the clusters of students who excel in {English and Math}, {Math and Science}, and {Science and English}.

11.1.4.1 Clustering Case Study on Student Marks (English and Math)

Firstly, the code for plotting the dataset is as shown below. The modules required are same as in the case of elbow curve example. Here, the column names Student, English, Math, and Science are extracted into a list. The dataset is in csv format and thus, pandas module is used to read the csv file. Since, in this case only English and Math are required only those certain features are extracted and converted to an array of values using numpy as the module. The plot() function is used to visualize the

Table 11.1 Student data for clustering

Student	English	Math	Science
1	99	96	97
2	99	96	97
3	98	97	97
4	95	100	95
5	95	96	96
6	96	97	96

Fig. 11.6 Student dataset

dataset as shown in Fig. 11.6 that depicts the marks of students in English and Math. The task is to group these data points into clusters.

```
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist
import numpy as np
import matplotlib.pyplot as plt
import pandas

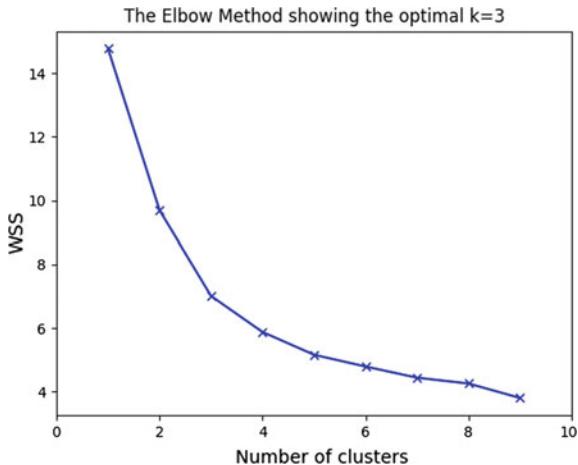
colnames=[‘Student’, ‘English’, ‘Math’, ‘Science’]
data=pandas.read_csv(‘Clustering.csv’, names=colnames)
English=data.English[1:].tolist()
Math=data.Math[1:].tolist()

x1 = np.array(English)
x2 = np.array(Math)

plt.plot()
plt.title(‘Dataset’)
plt.xlabel(‘English’)
plt.ylabel(‘Math’)
plt.scatter(x1, x2)
plt.show()
```

The task of executing clustering on the dataset with English and Math with code in Python is as shown below. First, the value of the k number of clusters is determined through elbow curve method. The same modules and wss method are used for determining the optimal number of clusters. The output of the wss plot is as

Fig. 11.7 Elbow curve for clustering case study on student marks (English and Math)



shown in Fig. 11.7. The number of clusters for this example is three which is inferred from the elbow curve because from $k = 3$ onward, the value of wss almost remains the same.

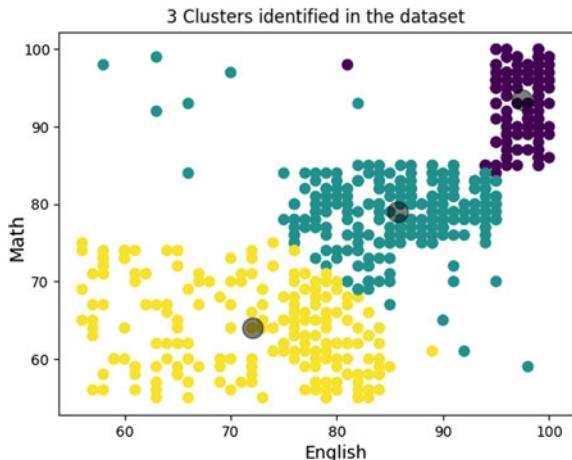
```
# create new plot and data
plt.plot()
X = np.array(list(zip(x1, x2))).reshape(len(x1), 2)
colors = ['b', 'g', 'r']
markers = ['o', 'v', 's']

wss = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans (n_clusters=k).fit(X)
    kmeanModel.fit(X)
    wss.append(sum(np.min(cdist
(X, kmeanModel.cluster_centers_, 'euclidean'), axis=1)) / X.shape[0])

# Plot the elbow
plt.plot(K, wss, 'bx-')
plt.xlim([0, 10])

plt.xlabel('Number of clusters')
plt.ylabel('WSS')
plt.title('The Elbow Method showing the optimal k=3')
plt.show()
```

Fig. 11.8 Clusters for case study on student marks (English and Math)



The code below shows the k -means clustering for the considered dataset. The number of clusters is set to 3, and k -means algorithm is executed to see the hidden patterns of clusters of students. The output of the plot() function of the clustering is as shown in Fig. 11.8.

It can be observed from Fig. 11.8 that there are three clusters represented by the colors yellow, blue, and purple. The purple color represents the students who have scored well (90–100) in both English and Math. The yellow-colored cluster represents the students who have scored average (60–70) in English and Math. The blue-colored cluster represents the students who have scored in the range of 70–90 marks in English and Math. The interesting point to be observed in the clustering plot is there are more number of blue points that are on the side of yellow-colored cluster. It infers that these students are good at Math and not in English. These hidden patterns can be found with the help of clustering.

```
#Clustering
kmeanModel = k-means(n_clusters=3).fit(X)
kmeanModel.fit(X)
y_kmeans = kmeanModel.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeanModel.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.title('3 Clusters identified in the dataset')
plt.xlabel('English')
plt.ylabel('Math')
plt.show()
```

11.1.4.2 Clustering Case Study on Student Marks (Math and Science)

The dataset also consists of another subject Science. The following code demonstrates clustering on Math and Science subjects. The output of the plot() function for the subjects Math and Science is as shown in Fig. 11.9. Here, the yellow-colored cluster represents the students who are excellent (90–100) in Math and Science. The purple-colored cluster represents the students who are good (75–90) in Math and Science. The blue-colored cluster represents the students who are average (50–70) in Math and Science. The dark-colored gray points represent the centroid of each cluster. The two purple-colored points in the plot represent the students who are excellent in Science but average in Math.

```
#Clustering
kmeanModel = KMeans (n_clusters=3).fit (X)
kmeanModel.fit(X)
y_kmeans = kmeanModel.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeanModel.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.title('3 Clusters identified in the dataset')
plt.xlabel('Math')
plt.ylabel('Science')
plt.show()
```

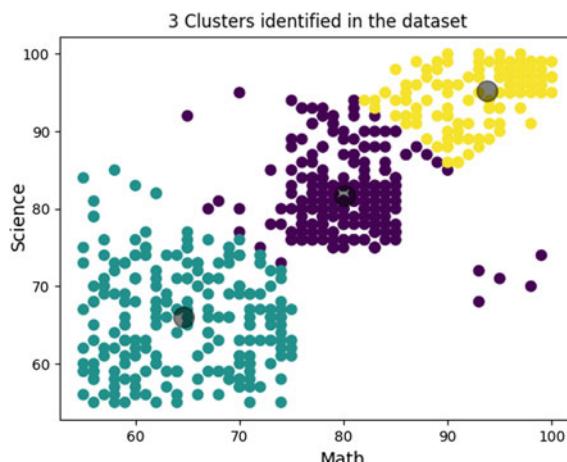


Fig. 11.9 Clusters for case study on student marks (Math and Science)

11.1.4.3 Clustering Case Study on Student Marks (Science and English)

The following code demonstrates clustering on Science and English subjects. The output of the plot() function for the subjects Science and English is as shown in Fig. 11.10. Here, the blue-colored cluster represents the students who are excellent (90–100) in Science and English. The purple-colored cluster represents the students who are good (75–90) in Science and English. The yellow-colored cluster represents the students who are average (50–70) in Science and English. The dark-colored gray points represent the centroid of each cluster. Some of the purple-colored points in the plot represent the students who are excellent in Science but weak in English as well as the students who are excellent in English but weak in Science.

```
#Clustering
kmeanModel = KMeans (n_clusters=3).fit (X)
kmeanModel.fit(X)
y_kmeans = kmeanModel.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeanModel.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5)
plt.title('3 Clusters identified in the dataset')
plt.xlabel('Science')
plt.ylabel('English')
plt.show()
```

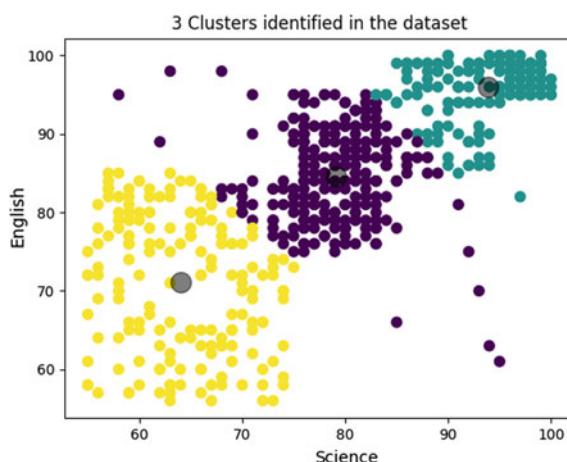


Fig. 11.10 Clusters for case study on student marks (Science and English)

11.2 Association Rule Mining

Association rule mining is one of the widely used machine learning methods for examining transactions in the areas of web mining, customer sales. The main goal is to find the interesting connections between the elements of the data in the transactions and correlate it with a result. In this section, a brief overview of association rule mining is discussed with examples.

11.2.1 Association Learning Basics

In association learning, there will be a set of transactions and items that are associated with each transaction as shown in Table 11.2. In transaction 1, product 1 and product 2 are purchased. Similarly in transaction 2, only product 3 is purchased and so on for other transactions. The main goal is to find the associations between the items. For example, if a customer purchases product 1 and product 2, is he likely to buy product 4 as well might be the question.

In association learning problem, there will be a set of items I and set of transactions T. An association rule will be of the form $X \rightarrow Y$, where X is called antecedent and Y is called consequent.

Some of terminologies related to association rule mining are discussed as below [3, 4]. For these terminologies, transactions are considered as shown in Table 11.2 and the item set as $X = \{P1, P2\}$.

- **Support:** It is the proportion of the transactions containing the item set as defined in Eq. 11.2.1.1.
- **Confidence:** It is the probability of finding transactions containing set X and also Y as shown in Eq. 11.2.1.2. For example, in the case of $(\{P1, P2\} \rightarrow P4)$, support $(\{P1, P2\} \cup P3) = 1/5 = 0.2$ and support $(\{P1, P2\}) = 0.4$. Therefore, the confidence of the rule $(\{P1, P2\} \rightarrow P3) = 0.2/0.4 = 0.5$

$$\text{SUPPORT}(X) = \frac{\text{No_of_transactions_containing_}X}{\text{Total_no_of_transactions}} \quad (11.2.1.1)$$

$$\text{CONF}(X \rightarrow Y) = \frac{\text{SUPPORT}(X \cup Y)}{\text{SUPPORT}(X)} \quad (11.2.1.2)$$

Table 11.2 Transactions and products purchase

Transactions	Products
T1	P1, P2
T2	P3
T3	P4
T4	P1, P2, P3

$$\text{SUPPORT}(X) = \frac{2}{5} = 0.4$$

- **Lift:** Given a rule $X \rightarrow Y$, lift is the number of times X and Y occurs together and they are independent of each other as shown in Eq. 11.2.1.3. The rule is considered to be more appropriate as the lift score is high.

For example, consider a scenario where among 500 transactions with items P1, P2, and P3, {P1,P2} appears in 200 transactions, P1 in 100 transactions and P2 in 100 transactions. Then Lift ($P1 \rightarrow P2$) = $0.4/(0.2*0.2) = 10$. Similarly, {P1, P3} appears in 100 transactions, P1 in 100 transactions and P3 in 75 transactions. Then Lift ($P1 \rightarrow P3$) = $0.4/(0.2*0.75) = 2.67$. Since, the LIFT($P1 \rightarrow P2$) is greater than LIFT($P1 \rightarrow P3$), P1 and P2 have stronger associations.

$$\text{LIFT}(X \rightarrow Y) = \frac{\text{SUPPORT}(X \cup Y)}{\text{SUPPORT}(X) * \text{SUPPORT}(Y)} \quad (11.2.1.3)$$

- **Leverage:** Given a rule $X \rightarrow Y$, leverage measures the probability of X and Y occurring together than expected if X and Y are independent of each other as shown in Eq. 11.2.1.4.

Consider the example of lift, Lift ($P1 \rightarrow P2$) = $0.4 - (0.2*0.2) = 0.36$ and Lift ($P1 \rightarrow P3$) = $0.4 - (0.2*0.75) = 0.25$. So, ($P1 \rightarrow P2$) scores high and hence P1 and P2 have stronger associations.

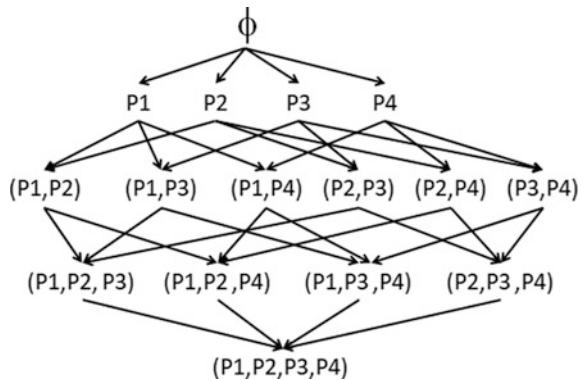
$$\text{LEVERAGE}(X \rightarrow Y) = \text{SUPPORT}(X \cup Y) - \text{SUPPORT}(X) * \text{SUPPORT}(Y) \quad (11.2.1.4)$$

11.2.2 *Apriori Algorithm*

Association rule mining can be carried out using Apriori algorithm. The main goal of the Apriori algorithm is to determine the frequent item set for a given set of transactions and items. Given an item set that is frequent, any subset of it is also frequent. This is known as Apriori property. For example, consider the frequent item set {P2,P3,P4} as shown in Fig. 11.11. All the subsets of it, namely {P2,P3}, {P2,P4}, {P3,P4}, P2,P3,P4, are also frequent item sets.

Apriori algorithm uses bottom-up approach in finding the frequent item sets starting from 1-item set, 2-item set, 3-item set, and so on. The algorithm is as shown below where

- C_k represents the item sets that can be used to generate frequent item sets.
- L_k represents the item sets that satisfy the minimum support S_t .
- In each iteration, L_{k+1} item sets are generated based on the L_k .

Fig. 11.11 Apriori property

- D represents the set of transactions.
- N_L is the optional parameter that specifies the length of the item set to be found.

```

Apriori (D, St, NL)
k=1
Lk={set of initial item candidates that satisfy the threshold St}
While Lc ≠ ∅
Begin
If ∃ NL
Ck+1 are the item sets generated from Lk
For each transaction in D
  Increment the count for Ck+1 if it is there in the transaction and
add it to Lk+1
End
Return Lk
  
```

11.2.3 Apriori Example

The following code represents the code for association rule mining using Apriori algorithm. The module apyori implements the Apriori algorithm as discussed in the previous section. The same module is used with method apriori. The method apriori () expects the argument as a list of transactions that consists of different products.

Here, a list of transactions along with the products are assigned first and given to the method apriori() for association rule mining. The minimum support given here is 0.5, i.e., 50%. The output of the apriori method consists of a list of records. These records contain the association rules and the frequent item sets.

```
from apyori import apriori

transactions = [
    ['A', 'B', 'C'],
    ['A', 'C'],
    ['B', 'C'],
    ['A', 'D'],
    ['A', 'C', 'D']

]

results = list(apriori(transactions, min_support = 0.5))
print(results[0])
print(results[1])
print(results[2])
```

Output:

```
RelationRecord(items=frozenset({'A'}),
support=0.8, ordered_statistics=[OrderedStatistic(items_base=
frozenset(), items_add=frozenset({'A'}), confidence=0.8, lift=1.0)])

RelationRecord(items=frozenset({'C'}),
support=0.8, ordered_statistics=[OrderedStatistic(items_base=
frozenset(), items_add=frozenset({'C'}), confidence=0.8, lift=1.0)])

RelationRecord(items=frozenset({'A', 'C'}),
support=0.6, ordered_statistics=[OrderedStatistic(items_base=
frozenset({'A'}), items_add=frozenset
({'C'}), confidence=0.7499999999999999, lift=0.9374999999999998),
OrderedStatistic(items_base=frozenset({'C'}), items_add=
frozenset
({'A'}), confidence=0.7499999999999999, lift=0.9374999999999998)])
```

Here the output consists of three relation records containing the item sets A, C, and {A,C} with support = 0.8, support = 0.8, and support = 0.6, respectively. In the relation record, the following terminologies are present.

- Items = frozenset(): It represents the items that have the minimum support.
- Items_base: It represents the antecedent of the association rule.
- Items_add: It represents the consequent of the association rule.

So for the example considered here, the association rule is $A \rightarrow C$ and the frequent item set is {A,C}. In this way, association rule mining can be implemented using apyori module in Python. It provides an easy way of identifying the association rules rather than spending time in iterations of determining the frequent item sets.

11.2.4 Association Rule Mining Case Study on Market Basket Analysis

In this section, a case study on market basket analysis is presented. A dataset on the market basket is considered that consists of a list of products as shown in Table 11.3. In each transaction, a list of products purchased is shown. For example in transaction T1, {milk and eggs} are purchased, T2, {spaghetti,milk,olive oil, brownies} are purchased, and so on. The Apriori algorithm is implemented on this dataset using the apyori module in Python.

The minimum support assumed is 0.003, confidence as 0.2, lift as 3, and minimum length = 3 restricting the number of frequent item sets to 3. Firstly, the modules required are numpy, pandas, and matplotlib. These modules are imported first, and the dataset is imported from csv file using pandas module. The transaction list is populated with the products list in the dataset.

The apriori method is populated with parameters min_support, min_confidence, min_lift, and min_length that return the association rules as a list of records. These lists of records are iterated to determine the list of rules that can be formed for the dataset. The output of the list of records is huge in number and consists of a number of relation records. In these relation records, the most suitable frequent item set was {}.

Table 11.3 Dataset for association rule mining

Transactions	Products
T1	Milk, eggs
T2	Spaghetti, milk, olive oil, brownies
T3	Burgers, red
T4	Wine, pepper, spaghetti, eggs, green
T5	Tea
T6	Burgers, whole wheat
T7	Pasta, soup, olive oil, cottage
T8	Cheese, energy drink
T9	Eggs, French fries
T10	Chocolate, cottage cheese, pancakes

```
# Apriori

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Data Preprocessing
dataset = pd.read_csv('Market_Basket_Optimisation.csv', header = None)
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i,j]) for j in range(0, 20)])

# Training Apriori on the dataset
from apyori import apriori
rules = apriori(transactions, min_support = 0.003,
min_confidence = 0.2, min_lift = 3, min_length = 3)

# Visualising the results
results = list(rules)
myResults = [list(x) for x in results]
```

In this section, unsupervised learning techniques clustering and association rule mining were discussed. Both techniques help in finding the relationships between the data points. Clustering helps in finding the patterns of data and finds a particular group of items that belong to a cluster. K -means algorithm is one of the earliest and fundamental algorithms for clustering. This chapter used an example on student marks for determining the groups of students in various subjects. Similarly, for association rule mining, a case study on the market basket analysis with products purchased was discussed. Other than these analytical techniques, some of the advanced techniques such as principal component analysis (PCA), random forest, and logistic regression, are discussed in the upcoming sections.

11.3 Principal Component Analysis

Usually, datasets are associated with high dimensionality consisting of many variables. The objective behind principal component analysis (PCA) is to reduce the dimensionality to suit the requirements. It is also noteworthy that a typical dataset consists of a number of variables associated with each other with a certain level of correlation which can be relatively analyzed as may be high or low degree of correlation. Along with the aim of reducing the dimensionality, it is of utmost

importance to keep the variation of the dataset intact [6]. PCA achieves this by transforming the existing variables into a new set of variables known as principal components and they are orthogonal. The higher the order, the greater the degree of variation present in the original dataset is preserved. Hence, it can be inferred that first principal component preserves the highest variation that was originally present in the components prior to PCA.

The main practical use of PCA is that in any dataset not all the attributes are helpful and many a times they tend to describe or measure related properties, and this creates a certain degree of redundancy. The principal component analysis helps to remove this redundancy and renders the analyst a summarized view which describes most of the relevant information which are proved to be helpful for further analysis.

In case of reduction from two dimensions to one dimension, it can be visualized as PCA picks up the line which passes through most of the points in the plane. Similarly in case of reduction from three dimensions to two dimensions, PCA chooses the most suitable plane. In this section, the key terms and the methodology for carrying out PCA are discussed.

11.3.1 Key Terms in PCA

- **Dimensionality:** It can be termed as the number of attributes or features or in mathematical terms the number of random variables that is present in the dataset chosen for analysis.
- **Correlation:** It represents the degree of relation between two variables so as to determine the relative strength of relation. The values range from -1 to $+1$, and positive values indicate positive correlation and negative values indicate negative correlation.
- **Positive Correlation:** It indicates that for a pair of correlated variables, if one of them increases the other increases as well. Positive slope in coordinate geometry can be considered as an analogy.
- **Negative Correlation:** It indicates that for a pair of correlated variables, if one of them increases the other in contrast decreases. Negative slope in coordinate geometry can be considered as an analogy.
- **Orthogonality:** It is a property in which two variables are uncorrelated to each other and hence the value of correlation is zero.
- **Covariance:** It refers to the joint variability of a pair of random variables. Statistically it means the mean value of the product of the deviations of two variables from their respective means.
- **Covariance matrix:** It is a $n \times n$ matrix which consists of covariance values between each pair of elements as in attributes in this case. The (i,j) th element refers to the covariance between attribute numbered i and j , respectively.

11.3.2 Steps Involved in Implementing PCA

- **Data Normalization:** Data normalization has to be carried out to ensure effective functioning of PCA. This procedure is essentially carried out before any kind of analysis. Data which is not normalized are sometimes considered meaningless and also result in higher values being overestimated.
- **Calculation of covariance matrix:** If a two-dimensional dataset is considered, the matrix can be constructed as follows. It results in an $n \times n$ matrix, and hence if two dimensions are considered, the result is a 2×2 matrix as shown below.

$$\text{Matrix(Covariance)} = \begin{bmatrix} \text{Var}[X_1] & \text{Cov}[X_1, X_2] \\ \text{Cov}[X_2, X_1] & \text{Var}[X_2] \end{bmatrix}$$

- **Arriving at eigenvalues and eigenvectors:** Eigenvalue can be determined as a solution of the characteristic equation of the covariance matrix. Here the determinant is applied on the quantity $\det(aI - A) = 0$ where,
 - ‘det’ represents the determinant operation
 - ‘a’ refers to the eigenvalue
 - ‘A’ is the covariance matrix
 - ‘I’ is an identity matrix of the same dimension as ‘A’
- **Determining the components and formulating the feature vector:** The number of eigenvalues and their corresponding eigenvectors is dependent on the number of variables or attributes chosen as they are evident in the dimension of the covariance matrix. Dimensionality reduction has to be carried out, and the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset. The number of components is chosen to suit the analysis experiment.

11.3.3 Principal Component Analysis in Python

In this section, PCA is discussed with an example in Python. The dataset considered for the PCA is diabetes dataset that is inbuilt in the `sklearn` module in Python. The dataset consists of ten dimensions and 442 sample instances. The following code shows how to reduce the ten dimensions of the diabetes dataset.

Initially, the dataset `diabetes` is loaded from the `sklearn` module. The *PCA* method is used to reduce the dimensions in Python. Here, in this case the number of dimensions chosen for reduction is 2. The converted values of the two dimensions are as shown in the final output. The values in the two dimensions represent range of values for the diabetes dataset considered.

```

from sklearn.decomposition import PCA
from sklearn.datasets import load_diabetes
import numpy as np

data = load_diabetes()
#dataset = list(zip(data.data, data.target))
features=np.array(data.data)
print("Number of dimensions of the dataset is", features.shape[1])

model=PCA(n_components=2)
model.fit(features)

print("10 dimensional features is converted to a two dimensional features
\nFor instance", features[0:1,:], "->", model.transform(features[0:1,:]))

model.explained_variance_ratio_

feature_in_2d=model.transform(features[0:1,:])
feature_in_10d=model.inverse_transform(feature_in_2d)
print("Two dimensional values =", feature_in_2d)
print("Ten dimensional values = ", feature_in_10d)
print("Original Ten dimensional values = ", features[0:1,:])

features_3d=features[:,0:3]
features_3d.shape[1]

```

Ouput:

```

Number of dimensions of the dataset is 10

PCA(copy=True,iterated_power='auto', n_components=2, random_state=None,
      svd_solver='auto', tol=0.0, whiten=False)

10 dimensional features is converted to a two dimensional features

For instance [[ 0.03807591  0.05068012  0.06169621
  0.02187235 -0.0442235 -0.03482076 -0.04340085 -0.00259226
  0.01990842 -0.01764613]] ->
[[ 0.02793062 -0.09260116]]

array([0.40242142, 0.14923182])

Two dimensional values = [[ 0.02793062 -0.09260116]]

```

```
Ten dimensional values =
[[ 0.0019362  0.04101691  0.02293928  0.02039249 -0.04347561
-0.03239287 -0.0547673   0.01829152  0.01300018  0.01686492]]
Original Ten dimensional values =
[[ 0.03807591  0.05068012  0.06169621  0.02187235 -0.0442235
-0.03482076 -0.04340085 -0.00259226  0.01990842 -0.01764613]]
```

11.3.4 PCA Visualization in Python

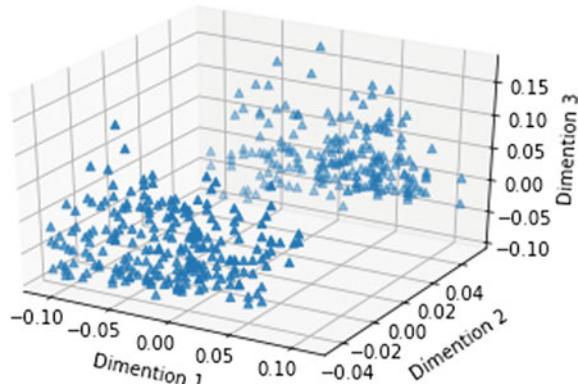
In this section, the PCA carried out in the previous section is visualized. The `matplotlib` module is used for the visualization. The variable `features_3d` holds the original ten-dimensional values of the diabetes dataset. These values are plotted as shown in Fig. 11.12. It can be observed from Fig. 11.12 that the values are scattered around the plot.

The PCA plot, i.e., after reducing it to two dimensions is as shown in Fig. 11.13. It can be observed from Fig. 11.13 that the data points scattered in the initial plot before dimensionality reduction are differentiable after PCA. A single line of plane can be drawn over the points for further analysis.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x,y,z=features_3d[:,0:1],features_3d[:,1:2],features_3d[:,2:]
ax.scatter(xs=x,ys=y,zs=z,marker='^')
ax.set_xlabel('Dimension 1')
```

Fig. 11.12 Diabetes dataset plot



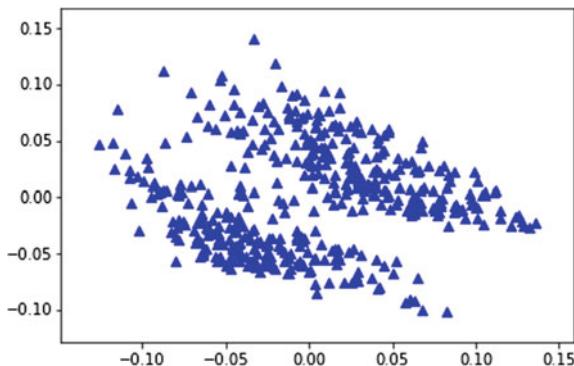


Fig. 11.13 Diabetes dataset plot after PCA

```

ax.set_ylabel('Dimension 2')
ax.set_zlabel('Dimension 3')
plt.show()

model=PCA(n_components=2)
model.fit(features_3d)
features_2d=model.transform(features_3d)

x,y=features_2d[:,0:1],features_2d[:,1:2]
plt.scatter(x,y,color='b',marker='^')
plt.show()

```

11.4 Random Forest

Random forest algorithm is one of the popular supervised classification algorithms in machine learning. It is one of the few algorithms which can be used for both classification and regression problems. Random forest gets its name because it creates a forest with lot of decision trees [5]. The higher the number of decision trees generated, higher is the accuracy of the classification or regression model.

To model multiple decision trees and create the forest, the same methods to create decision trees are used. The features of random forests are listed as follows.

- It is can be used for both classification and regression problems.
- It runs efficiently on large databases.
- It has effective methods for estimating missing data and maintains accuracy when a large portion of data is missing.
- It does not overfit the model when we have more trees.

- It is capable of extending to unsupervised clustering, data views, and outlier detection.

11.4.1 Classification Using Random Forest

Classification is a process of identifying different groups which target variable would likely fall into. Random forest working can be divided into two parts, where first stage is creation of forest and next stage is to making prediction from random forest classifier created in first stage. The steps followed for the creation of random forest are listed as follows.

Steps for Creation of Random Forest

- From the dataset, select k features from the given m features.
- The decision points among the k features are decided for splitting points in the tree.
- The nodes are split into level of trees.
- Repeat the above steps until only single node is reached.
- Build the random forest using the previous steps for the required number of n times.

Steps for Prediction of Random Forest

- The test features of the dataset are taken, and the rules are used for each randomly created decision tree to predict the outcome and store the predicted outcome (target).
- Calculate the votes for each predicted target.
- Consider the high-voted predicted target as the final prediction from the random forest algorithm.

The test feature is tested with all the decision trees in the random forest, and predicted outcome is stored. Then random forest just calculates the votes for each predicted outcome. The predicted outcome with most no of votes is final prediction from random forest algorithm. In this way, the random forests can be created and predicted for classification. A sample random forest can be seen in Fig. 11.14.

In Fig. 11.14, it can be observed that the few features are selected out of total features as a dataset. These randomly selected features are used to find the root node by using best split method. Then the daughter nodes are calculated using the same concept until we finally generate a decision tree by getting a leaf node. This process is carried to create many decision trees thus getting random forest.

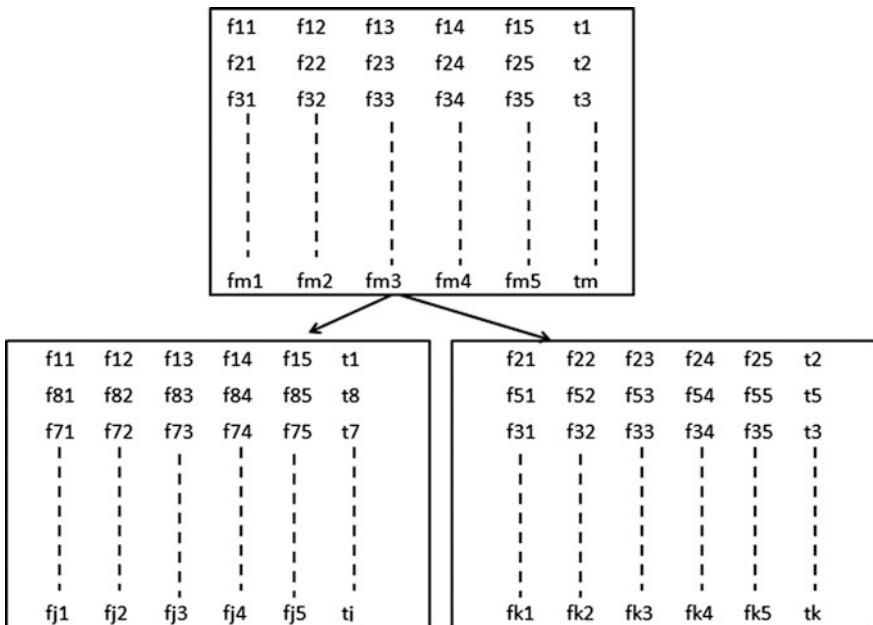


Fig. 11.14 Random forest

11.4.2 Random Forest and Regression

Simple decision trees can be used to find regression by using classification and decision trees (CART) algorithm. CART algorithm splits the plane into different regions and assigns a value to each region. It does this by a sequence of questions, the answer to which leads to next questions until terminal node is reached.

But simple regression tree can be very unstable as different regression trees can be generated for same data. Random forest solves this problem using bagging. The estimation of different regression trees is estimated to make final prediction by computing the average of the prediction across the trees. Even though different regression trees give different results, the instability is neutralized by bagging concept.

11.4.3 Missing Value Replacement for Training Set in Random Forest

One of the features of random forest is that it replaces large set of missing values in training dataset. This is an important feature as lot of other algorithms are not going to work properly, and missing values need to be taken care before we give the input. Random forest mainly uses two techniques to do the replacement of the missing values.

- (1) The first way is an easy and fast way to replace the missing values. If the m th variable is not categorical variable, then the median is computed of all values of this m th variable in that class j . Then it replaces all missing values in the training set of class j with the median. If the variable is categorical, then it replaces with most frequent non-missing value in class j . This replacement is called fills.
- (2) The second way is bit more expensive in terms of computational power but has better performance. It replaces the missing values only in training set with some rough and in accurate values. Then it computes proximities of the fills. If the missing variable is continuous value, then it fills as an average over the non-missing values of m th variables by the proximities between the n th case and non-missing value case. If it is a missing variable and is categorical, then replace it with most frequent non-missing values where the frequency is weighted by proximity.

Overfitting in Random Forest

Many other models tend to overfit when there is increase in the number of iterations. However, as there is increase in the number of trees, the overfitting by random forest decreases. The strength of random forest is that all trees are different. This is because all the trees start from scratch. Therefore, lot of new trees do not know overfitting trend which few decision trees knows. Random forest also takes average of all prediction, therefore canceling any biases. Due to this, random forest does not overfit the model when number of trees increases.

11.4.4 Random Forest Using Python

In this section, random forest example is demonstrated using Python. The dataset considered for the random forest is iris dataset that contains three classes of flowers, namely Setosa, Virginica, and Versicolor. The following code demonstrates the random forest in Python.

Initially, the required modules such as random forest classifier, `load_iris`, are imported first. The dataset is then split into training and test datasets. The features are created for the classification from the dataset. The testing dataset size is fixed to 25% for prediction of the classification.

The number of trees is initialized to 7 for the creation of the random forest. The method `RandomForestClassifier()` used is where the number of trees in the random forest is initialized to 7 and the method used is *entropy*. The information gain is used as the main criterion for the random forest creation. The random forest created is used to fit the model features of the training and the testing datasets.

The confusion matrix for the random forest classifier is validated in the end. It can be seen from the confusion matrix that 14 instances of the iris dataset are classified correctly as Setosa. Similarly, 15 instances are classified as Virginica and

another 9 instances are classified as Versicolor. In this way, the random forest classification can be used.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix

data = load_iris()
dataset = list(zip(data.data, data.target))
train , test = train_test_split(dataset, test_size=0.25)
train_features = []
train_labels = []
test_features = []
test_labels = []
for item in train:
    train_features.append(item[0])
    train_labels.append(item[1])
for item in test:
    test_features.append(item[0])
    test_labels.append(item[1])

number_of_trees=7
classifier = RandomForestClassifier(n_estimators=number_of_trees,bootstrap=True,criterion='entropy')
classifier = classifier.fit(train_features, train_labels)
results = classifier.predict(test_features)
print("model accuracy",accuracy_score(test_labels, results))
print("Confusion matrix :-\n",confusion_matrix(test_labels,results))
```

Output:

Confusion matrix

```
[[14  0  0]
 [ 0 15  0]
 [ 0  0  9]]
```

11.4.5 Visualization of Random Forest in Python

The random forest classifier used in the previous section can be visualized using the following code. The matplotlib module is used to visualize the random forests. The

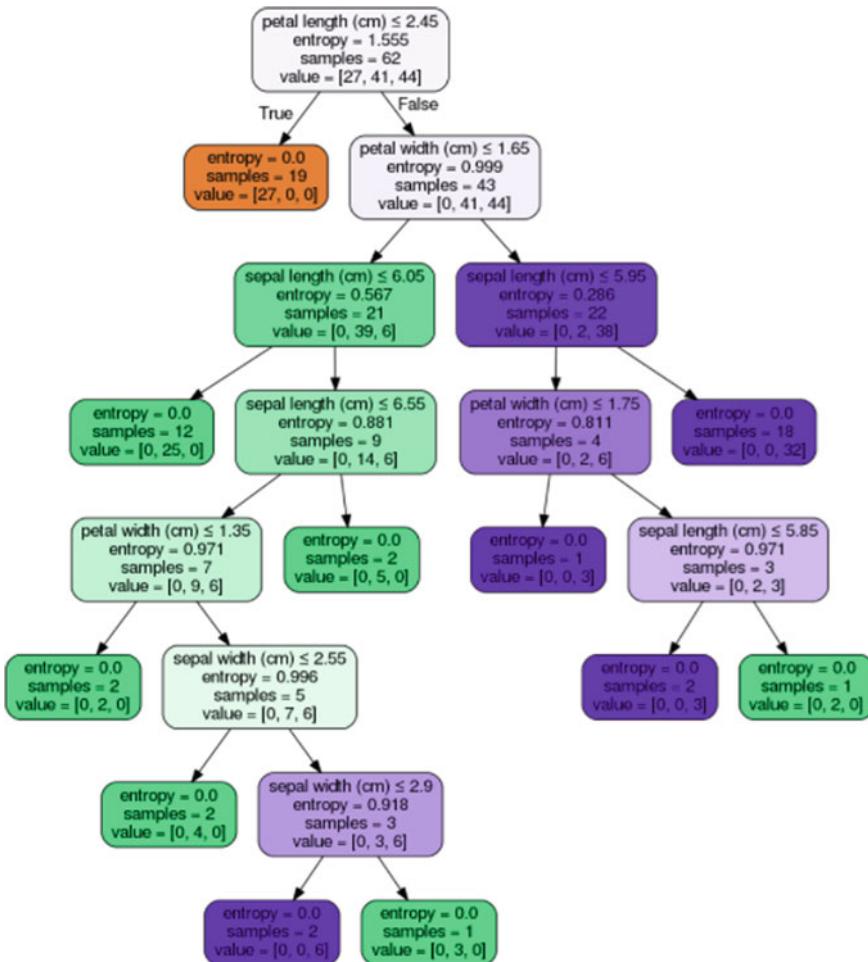


Fig. 11.15 Random forest tree for iris dataset

tree obtained by the classifier is passed to the plot function for the visualization. The output of the plot is as shown in Fig. 11.15.

```

from matplotlib import pyplot as plt
from matplotlib import image
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus

```

```

trees=[]
for tree in classifier.estimators_:
    dot = StringIO()
    export_graphviz(tree, out_file=dot,filled=True, rounded=True,
feature_names=data.feature_names,special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot.getvalue())
    trees.append(graph)

tree_number=1
Image(trees[tree_number].create_png())

```

11.5 Logistic Regression

This classification method is an application of regression analysis. It is a regression model in which the target variable is a categorical variable. Similar to any regression analysis, logistic regression computes the intercepts and coefficients based on the feature values called as the parameters of the model. These parameters are used to compute the probability of each data point to be classified into a specific class [7]. The probabilities are computed based on a function called logistic function. It is generally denoted by t where t is the input feature. The logistic function is defined as seen in the equation.

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

In any classification problem, the input feature vector, t is of the form $(x_1, x_2, x_3, x_4, \dots, x_n)$. This feature vector can be represented as follows:

$$t = \alpha + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n = \alpha + \sum_{i=1}^n \beta_i x_i$$

Thus, the logistic function can be rewritten as follows:

$$\sigma(t) = \frac{1}{1 + e^{-(\alpha + \sum_{i=1}^n \beta_i x_i)}}$$

The model parameters are obtained as any other regression model. The target variables are learnt from the feature vector and intercepts and the coefficients are determined. Once they are obtained, the logistic function is used to determine the probability of each class. Since $0 < t < 1$, the output of this function can be interpreted as probability. This method can be easily understood using a binary logistic regression (only two classes). Once multiple classes come into picture, the

understanding becomes quite complex. A simple binary logistic regression is discussed using an example in the next section followed by multiclass logistic regression, and then a case study is discussed.

11.5.1 Logistic Regression in Python

With the above overview of logistic regression, a simple example is discussed on logistic regression model for classification. A sample dataset considered contains one predicting variable and one target variable. The dataset is about the requirement of service by an automobile based on the number of days it has been used. The goal of the regression model is to predict the requirement of service. Even though it is quite simple, this will allow us to have an understanding on how logistic regression works. The dataset is as shown in Table 11.4.

While solving this regression problem, NO is considered as 0 and YES is considered as 1. First, using the logistic regression model is used to solve for intercept and coefficient. There will be only one intercept and coefficient because independent variable is a single-dimensional vector. Once computed, we will get the values as shown in Table 11.5.

Table 11.4 Logistic regression dataset

S.No.	Days	Service
1	50	NO
2	75	NO
3	100	NO
4	125	NO
5	150	NO
6	175	NO
7	175	YES
8	200	NO
9	225	YES
10	250	NO
11	275	YES
12	300	NO
13	325	YES
14	350	NO
15	400	YES
16	425	YES
17	450	YES
18	475	YES
19	500	YES

Table 11.5 Parameter estimation in logistic regression

Parameter	Value
Intercept	-1.4232019497184076
Coefficient	0.0062821669680195045

Once these parameters are computed, the probability of requiring the service can be estimated using the logistic function. Consider the feature vector 9 (225,YES). Its probability can be computed as shown below.

$$\text{probability of requirement} = \frac{1}{1 + \exp(-(\text{intercept} + \text{coefficient} * 225))} = 0.49$$

For programming in Python, *sklearn* provides logistic regression as part of the *linear_model* module. The above example is demonstrated using *sklearn* logistic regression mode in Python using the following code.

Initially, the modules required for the logistic regression modeling are imported, and the data prepared for the logistic regression are saved in a csv file. This csv file is read, and logistic regression model is built on it. The column *requires_service* is used to extract the values of the dataset, and the *days* are used to fit the regression model. The intercept and the coefficient of the model is obtained using the *fit()* function.

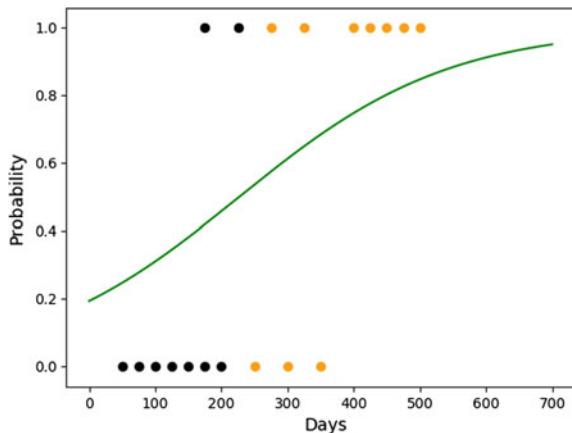
```
#import the libraries required.
import pandas as pd
import matplotlib.pyplot as plt
from math import exp
from sklearn.linear_model import LogisticRegression
import numpy as np

#define the logistic function.
def p(feature):
    global intercept,coef
    return 1/(1 + exp(-(coef*feature + intercept)))

#load the dataset as a pandas dataframe.
data = pd.read_csv('data.csv')
print(data)

#Train a logistic regression model and get the model parameters.
model = LogisticRegression()
model.fit((data['days'].values).reshape(-1,1),
          data['requires_service'].values)
global intercept
intercept = model.intercept_[0]
global coef
coef = model.coef_[0][0]
print(intercept,coef)
```

Fig. 11.16 Logistic regression model example



```
#plot the logistic function curve and the data points.
X = np.arange(0, 700, 0.1)
Y = list(map(p,X))
plt.xlabel('Days')
plt.ylabel('Probability')
plt.plot(X,Y,'g')
colors = {0:'black',1:'orange'}
Y_pred = model.predict((data['days'].values).reshape(-1,1))
for i in range(len(data)):
    plt.scatter(data['days'][i],
                Data['requires_service'][i],
                color = colors[Y_pred[i]])
plt.show()
```

The `plot()` function is used to see the logistic regression model. The output of the plot is as shown in Fig. 11.16. It can be seen from Fig. 11.16 that the data points can be easily classified into two classes without any ambiguity. In this way, the logistic regression is used for classification. In the next section, it is extended for classification of more than two classes using multinomial logistic regression.

Table 11.6 Intercept for logistic regression on iris data

-0.42881232	0.77961768	-2.171977	-0.87137675
-0.01187388	-1.79145715	0.43794731	-0.8268954
-0.54835467	-0.34274084	2.78417294	2.06613051

11.5.2 Multinomial Logistic Regression

If the target variable is multiclass variable, then multinomial logistic regression is used. Multinomial logistic regression is used when the dependent variable in question is nominal (equivalently *categorical*, meaning that it falls into any one of a set of categories that cannot be ordered in any meaningful way) and for which there are more than two categories. Multinomial logistic regression works on following assumptions:

- It assumes that each independent value is unique for each case in the data.
- It also works on the principle that we cannot classify the data perfectly from the feature vector in any case.

A multinomial logistic regression is trained as a set of independent binary logistic regression models. For N possible outcomes, we train $N - 1$ binary logistic regression models in a one-against-all method for all the classes. For all binary models, we consider one class as *pivot* (the outcome N) and train the model as *pivot versus rest*. Then, the probability of each class is determined as shown in the equation.

$$P(T = i) = P(T = N) * e^{\beta_i * X_i}$$

Where,

- T is the target, and X_i is the feature vector of the input i . Harnessing the fact that all these probabilities sum to 1, we can arrive at the following:

$$P(T = k - 1) = \frac{e^{\beta_{k-1} * x_i}}{1 + \sum_{i=1}^{k-1} e^{\beta_k * x_i}}$$

The unknown parameters in each vector β_k are typically jointly estimated by maximum a posteriori (MAP) estimation, which is an extension of maximum likelihood using regularization of the weights to prevent pathological solutions (usually a squared regularizing function, which is equivalent to placing a zero-mean Gaussian prior distribution on the weights, but other distributions are also possible).

11.5.3 Multinomial Logistic Regression Using Iris Dataset

In this section, logistic regression is applied to the well-known classification problem of the iris dataset. Sklearn library's logistic regression implementation is used to build our model. The aim of the regression is to predict the type of the flower based on the length and width of petal and sepal of a flower. The following code demonstrates the logistic regression in Python.

The dataset of iris consists of three classes, namely Setosa, Virginica, and Versicolor. The regression model built in this section is to classify the instances based on these three classes. Initially, the required modules for the logistic regression are imported first. The dataset iris is loaded first and split into training and testing sets. On the training set, logistic regression model uses the method `newton-cg` since it is a multiclass problem.

The model intercepts and coefficients can be obtained using the model. These parameters can be used for the estimation of the values in the dataset. The regression model is evaluated using the confusion matrix in the end.

```
#import the required libraries for model building and data handling.
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix
from sklearn.model_selection import train_test_split

#Load the dataset and generate the train and test datasets
#with features and labels.
data = load_iris()
dataset = list(zip(data.data, data.target))
train , test = train_test_split(dataset, test_size=0.25)
train_features = []
train_labels = []
test_features = []
test_labels = []
for item in train:
    train_features.append(item[0])
    train_labels.append(item[1])
for item in test:
    test_features.append(item[0])
    test_labels.append(item[1])

#Build the model with Newton conjugate gradient solver for
#multinomial classification.
model = LogisticRegression(solver='newton-cg') # since, multiclass
model.fit(train_features,train_labels)

#Get the model parameters.
print(model.coef_,model.intercept_)

#test the model.
results = model.predict(test_features)
acc = accuracy_score(test_labels,results)
cm = confusion_matrix(test_labels,results,labels=[0,1,2])
print(acc)
```

Table 11.7 Coefficients for logistic regression on iris data

6.63886903	3.77983127	-12.42753539
------------	------------	--------------

Table 11.8 Confusion matrix for logistic regression on iris data

10	0	0
0	15	4
0	0	9

```
#print the confusion matrix.
for row in cm:
    for item in row:
        print(item, end='\t')
    print()
```

The parameters obtained by the model in the code are listed in Table 11.6. In Table 11.6, the first column represents the intercepts and the second column represents the coefficients.

The intercept table consists of 12 values that are calculated by the model in the code. The first column represents the intercepts for the feature *sepal length*. Similarly, the other columns represent the intercepts for the features *sepal width*, *petal length*, and *petal width*, respectively. Similarly, the coefficients for each of the class are obtained for the dataset as shown in Table 11.7. The first column represents the coefficient for the *Setosa* class, second column value represents the value for the *class Virginica*, and the final one for the *class Versicolor*. These values can be used to calculate the logistic value of each data point as described for the earlier example of the logistic regression.

The confusion matrix as shown in Table 11.8 obtained by the logistic regression model can be interpreted in the following way. The first row depicts the class Setosa, second row depicts the class Virginica, and the third row represents Versicolor. The logistic regression model obtained has been classified into ten instances of Setosa class flowers correctly, similarly, 15 instances of Virginica flower and nine instances of Versicolor class flower correctly. However, the other four instances of the class Versicolor are classified incorrectly by the model as Virginica.

Logistic regression is a classification method for analyzing the multiclass problems. Since, linear regression cannot be used on categorical variables, logistic regression needs to be used. In this section, logistic regression was discussed with two examples. One example was on the requirement of the service of the vehicle based on the number of days and another on the iris dataset. In both examples, initially the parameters of intercepts and coefficients were obtained for the regression model. These parameters were then assessed for the prediction of the classification later. In this way, the logistic regression model can be used for classification in analytics.

11.6 Handling Missing Values

Missing data points are a common occurrence in real-world datasets and often play a crucial part in data wrangling. There are a number of ways to deal with missing data. The first and foremost important part is to find the number of missing values in each column of the dataset, and this would give a more clear picture on how to go about handling these missing values. In this section, the methods to overcome the missing values are discussed.

11.6.1 *Deleting Rows with Missing Values*

If the data considered are extremely rich and comparatively there are only a few rows with missing values, then these rows can be removed, which does make us lose data. It would not affect the overall analysis considering that there were very few of these rows as compared to the total number of rows in the dataset. This, however, is not an option in case where there are no many data points to begin with. Losing any data in this case can prove disastrous in the long run. In such cases, other techniques should be used to deal with missing data.

11.6.2 *Deleting Columns with Large Number of Missing Values*

If dataset contains a large number of columns and a lot of the missing values occurring in one single column, all the rows cannot be deleted at once with the missing values in this case. On the other hand, the alternative way is to delete the missing data column itself, but before doing this it is a good practice to make sure that the column in question is not very impactful to the dataset and its analysis.

This can be done in a number of ways but creating a heatmap of columns with missing values with the column to be predicted in case of classification is a good way to go. If the column with missing data is not highly correlated with the column being predicted, then it is alright to get rid of it and thus reducing the number of missing values in the dataset.

11.6.3 *Averaging Techniques*

Mean, median, and mode are the most popular averaging techniques, which are used to infer missing values. Approaches ranging from global average for the variable to averages based on groups are usually considered. Doing this gives a

quick estimate of the missing values, but artificially reduces the variation in the dataset as the missing observations could have the same value. This may impact the statistical analysis of the dataset since depending on the percentage of missing observations imputed, metrics such as mean, median, correlation may get affected. This might also make classification models infer the wrong things as it could see the same value repeat over and over again in the same column in the dataset.

11.6.4 Predictive Techniques

Imputation of missing values from predictive techniques assumes that the nature of such missing observations is not observed completely at random, and the variables chosen to impute such missing observations have some relationship with it, else it could yield imprecise estimates. It is useful when data are scarce and deleting data points is not an option. These techniques are not preferred in regression problems, because it often provides the model wrong insights.

11.6.5 Handling Missing Values in Python

The following code demonstrates how to handle missing values in Python. The dataset considered for this section is the Pima Indian diabetes.

In this example, it can be seen that the total number of missing data is comparatively high as compared to the total rows, therefore dropping fourth column, which has the most number of missing values and then removing the rows with the missing data seems like an appropriate strategy.

```
from pandas import read_csv
import numpy
data = read_csv('pima-indians-diabetes.data.csv', header=None)
print(data)

0   1   2   3   4   5   6   7   8
0   6  148  72  35   0  33.6  0.627  50  1
1   1   85  66  29   0  26.6  0.351  31  0
2   8  183  64   0   0  23.3  0.672  32  1
3   1   89  66  23  94  28.1  0.167  21  0
4   0  137  40  35  168  43.1  2.288  33  1
5   5  116  74   0   0  25.6  0.201  30  0
6   3   78  50  32  88  31.0  0.248  26  1
7  10  115   0   0   0  35.3  0.134  29  0
```

```
#0 is unacceptable value so it is considered as missing.  
We replace 0 with Nan  
data[[1,2,3,4,5]] = data[[1,2,3,4,5]].replace(0, numpy.Nan)  
# then we count the number of NaN values in each column to  
decide the strategy to handle to missing value  
print(data.isnull().sum())
```

```
0      0  
1      5  
2     35  
3    227  
4    374  
5     11  
6      0  
7      0  
8      0
```

#we can handle the missing data by removing the rows with missing data
data.dropna(inplace=True)

In this section, the different methods of handling the missing values were discussed with an example. As it was seen in the example of diabetes dataset, one of the columns of the data had more missing values and the strategy was to drop the column for analysis. In this way, the missing values in the dataset can be handled for the analysis.

11.7 Exercises

1. What is the Apriori property? What are the cons of Apriori?
2. What are the applications of association learning other than market basket analysis?
3. What is the n_estimators parameter in the RandomForestClassifier() function?
4. How can a particular tree be used and selected that is in the random forest?
5. How can you visualize the trees in the random forest?
6. What is the difference between linear regression and logistic regression?
7. What is the basic goal of logistic regression? How are the regression coefficients computed in logistic regression?
8. How do you identify missing values in a dataset?
9. What is pruning? How can pruning of the trees can be done in the random forest?
10. Following the example on the diabetes dataset provided, carry out a similar analysis on the iris dataset. Extract any two features of the dataset and reduce the dimensionality to 1.

References

1. Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Surveys (CSUR)*, 31(3), 264–323.
2. Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31 (8), 651–666.
3. Agarwal, R., & Srikant, R. (1994, September). Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference* (pp. 487–499).
4. Brin, S., Motwani, R., Ullman, J. D., & Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data. *ACM SIGMOD Record*, 26(2), 255–264.
5. Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32. *View Article PubMed/NCBI Google Scholar*.
6. Bro, R., & Smilde, A. K. (2014). Principal component analysis. *Analytical Methods*, 6(9), 2812–2831.
7. Harrell, F. E. (2001). Ordinal logistic regression. In *Regression modeling strategies* (pp. 331–343). Springer, New York, NY.

Part III

Advanced Analytics

Chapter 12

Text Analytics



12.1 Introduction to Text Analytics

Most of the information available on the Web is in textual format or in a semi-structured way. Analytics on this textual information plays an important role in gaining insights. Text analytics is a process of representation and modeling of textual content to gain insights into it. It helps in uncovering the interesting patterns underlying the large text information. For example, text analytics on product reviews can be helpful in making decisions around the good products and bad products. One of the main intentions of text analytics is to categorize documents into various classes [1]. This type of classification helps in applications such as spam classification, news classification, and story classification.

A conceptual view of text analytics is represented in Fig. 12.1. A collection of documents or in a single document large amount of information is present. This information needs to be segregated into separate classes so that it is easy to identify the different classes of information. For example, in Fig. 12.1 a collection of documents such as Avengers, Conjuring, and Batman are present. This collection can be categorized into three main classes, namely Food, Mobile Phones, and Movies. In this way, each text classification technique at the end produces different classes of documents. In this chapter, the different concepts around the text analytics are discussed with examples.

12.2 Steps Involved in Text Analytics

Since, text analytics play a vital role in gaining insights about various information and necessary steps need to be performed for carrying out analytics [1, 2]. The various steps that are involved in text analytics are as shown in Fig. 12.2. The process starts with collection of text, removal of stopwords, generating bag of

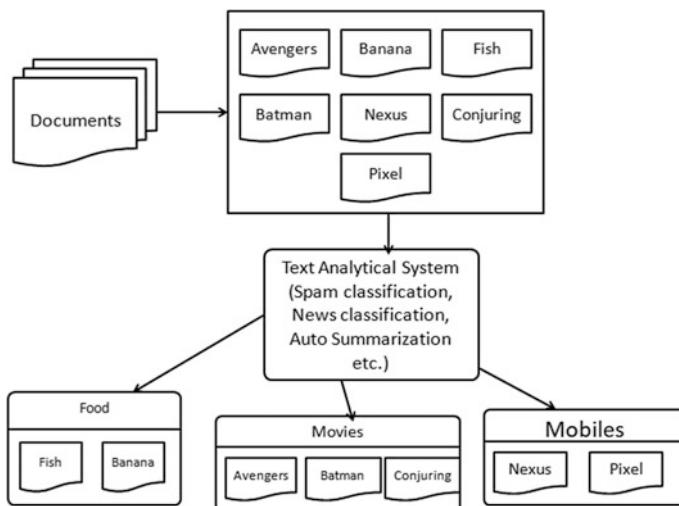


Fig. 12.1 Conceptual view of text analytics

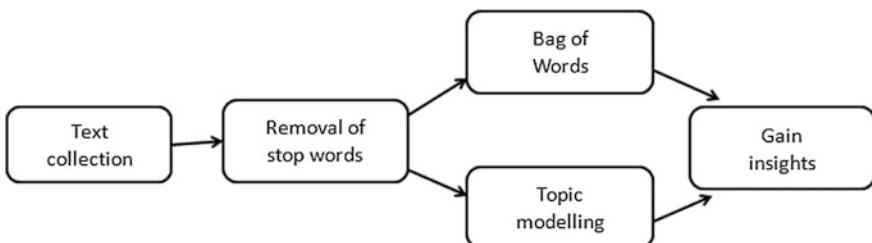


Fig. 12.2 Text analytics steps

words, topic modeling, and finally gain insights around it. The different types of information processed in each step are discussed as follows:

- **Text collection:** The very first step in text analytics is collection of text, i.e., either in raw format or in semi-structured format. The unstructured data such as tweets and RSS feed need to be pre-processed first and then analysis can be carried out. Some of the techniques for text collection are included as a part of web scraping.
- **Removal of stopwords:** Some of the phrases such as is, was, if, else, then, thus, so do not convey significant information about the text. Such phrases are called as stopwords. These stopwords are removed from the text initially for further processing.
- **Bag of words:** In the next phase of text analytics, a word cloud model and bag of words are built. A vector of words representing their frequencies in the text is depicted through the bag of words and word cloud model.

- **Topic modeling:** The pre-processed text in the earlier phases is used for categorization of documents based on the topic, for example, spam/non-spam, sports/politics/media.

For each of this phase of text analytics, a small example and its significance are presented in the upcoming sections of the chapter. The same steps are applied for two case studies discussed in this chapter.

12.3 Names, Numbers, and Stopwords Removal

In the text classification process, the first step is to pre-process the information of the text so that the significant content of the information is available for text analytics. In this section, a small example is presented in Python for removal of stopwords, names, and numbers in the text that does not carry necessary information [3]. The following code demonstrates the removal process of numbers and stopwords.

The main module required for text analytics and classification is nltk module. In this module, ‘stopwords’ module is used for removal process. A sentence is initialized first that contains numbers and stopwords. Some of the examples of stopwords are if, is, was, else, in, not. The sentence is converted to lower case first using the *lower()* function, and then the punctuations are removed using *punctuation()* method.

The stopwords are then removed from the sentence using the stopwords module by using the parameter *english* to it. The numbers in the sentence are removed using the regular expression *[0-9]* that identifies the numbers in the sentence. The original sentence and the pre-processed sentence are presented in the code.

```
import string
from nltk.corpus import stopwords
importre

sentence="PretzelBros, airbnb for people who like pretzels,
raises $2 million"

sentence=sentence.lower()
sentence

'pretzelbros, airbnb for people who like pretzels, raises $2 million'

symbols=string.punctuation
sentence="".join([x for x in sentence if x not in symbols])
sentence
```

```
'pretzelbrosairbnb for people who like pretzels raises 2 million'

sentence=" ".join([x for x in sentence.split() if x not in stopwords.words('english'))]

sentence=re.sub('[0-9]','',sentence)
print(sentence)
pretzelbros airbnb people like pretzels raises million
```

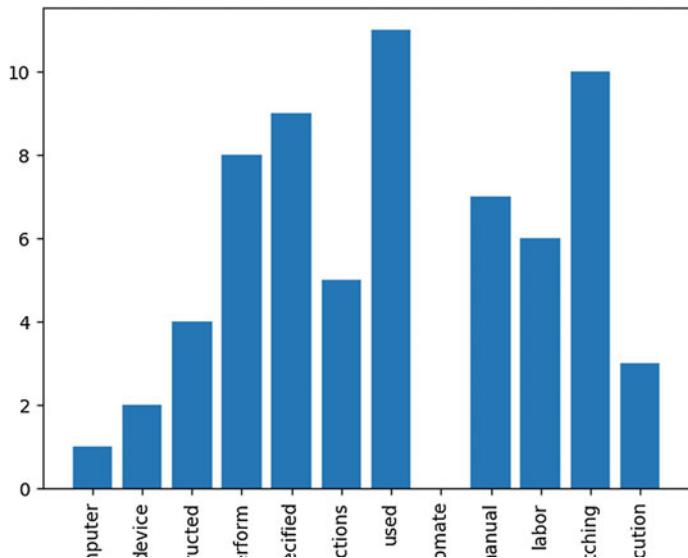
The last sentence in the code represents the sentence where the names, numbers, and stopwords are removed. The next step for text classification/analytics is to calculate the number of times/frequency of words/terms in the text. In the next section, the process of carrying out word frequency analysis is presented.

12.4 Word Frequency Analysis

Once the stopwords are removed from the text, the next step of text analytics is calculating the frequency of words in the text. The main aim of calculating the word frequency is to determine the sentences with highest frequency of the words. It is also referred to as term frequency analysis. The number of occurrences of the word is calculated based on the corpus.

The following code demonstrates word frequency analysis in Python [2, 3]. A text is initialized with two sentences as a list in the *Text*. The sentence is pre-processed first by removing the stopwords using the same procedure as discussed in the previous section. *CountVectorizer* module is used for counting the occurrences of the word in the text. A small model is built using the *tf_vectorizer()* function. The vocabulary of the model can be printed to see the occurrences of frequencies of each word in the document. For the same vocabulary, a graph can be plotted using the *matplotlib* module. The output of the plot is as shown in Fig. 12.3.

```
from nltk.corpus import stopwords
import string
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot as plt
Text = ["Computer is a device that can be instructed to
perform specified instructions." ,
        "Computer is used to automate manual labor through
unmatching instructions execution."]
def preprocess(sentence):
    sentence=sentence.lower()
    sentence="".join([x for x in sentence if x not in
string.punctuation])
```

**Fig. 12.3** Word frequency analysis model

```

sentence=[x for x in sentence.split(" ") if x not in stopwords.-
words('english')]
sentence=[x for x in sentence if x!=""]
return " ".join(sentence)

tf_vectorizer = CountVectorizer(lowercase=True,
preprocessor=preprocess)
model = tf_vectorizer.fit(Text)

print(model.vocabulary_)

x = [ i for i in range(len(model.vocabulary_)) ]
y = []
x_t = []
for item in model.vocabulary_.keys():
    x_t.append(item)
    y.append(model.vocabulary_[item])

plt.figure(figsize=(30,30))
plt.bar(x,y)
plt.xticks(x,x_t,rotation='vertical')
plt.show()

```

The above code generates the following output and plot:

```
{'computer': 1, 'device': 2, 'instructed': 4, 'perform': 8, 'specified': 9, 'instructions': 5, 'used': 11, 'automate': 0, 'manual': 7, 'labor': 6, 'unmatching': 10, 'execution': 3}
```

12.5 Generating Bag of Words

In text analytics, once the frequency of the words is calculated, a bag of words need to be generated for further analysis. In this section, the generation of bag of words is discussed with examples.

When it comes to text analysis, the biggest concern is to represent the textual data with numbers that make sense to the statistical models. One of the basic representations used is the ‘bag of words’ model. The main aim of it is to put the corpus of words into a common bag and then count the frequency of each word from the bag. Due to this, it is called the bag of words (BOW) model. One point that is to be noted in this model is that the context of the words in the corpus is completely ignored while adding them to the bag. It gives some context about the data being considered for analysis. While performing advanced tasks that involve the context of the words, the more powerful representations like word embeddings are used.

A bag of words model is built in two phases:

- First, a list of words is generated from the corpus.
- Then, for each word in the corpus, we generate the frequency of the occurrence of that word in the trained corpus.

The following corpus of information is based on artificial intelligence and is taken from Wikipedia

An artificial brain (or artificial mind) is software and hardware with cognitive abilities similar to those of the animal or human brain. Research investigating “artificial brains” and brain emulation plays three important roles in science: An ongoing attempt by neuroscientists to understand how the human brain works, known as cognitive neuroscience.

The following code demonstrates the use of bag of words from scikit-learn package of Python. Initially, the text data are loaded from using the file matter.txt from the Wikipedia. The data are cleaned and then used to fit the estimator to the data. Finally, the transform method is used to get the required representation from the count matrix.

```
#import the modules required.
from nltk.corpus import stopwords
import string
from sklearn.feature_extraction.text import CountVectorizer
```

```

from nltk.tokenize import sent_tokenize,word_tokenize

#Load the corpus from a text file and tokenize it into sentences.
with open('matter.txt','r') as f:
    data = f.read()
    Text = sent_tokenize(data)

# Total number of sentences in the data. Prints 14 for this text.
print(len(Text))

#Define the preprocessor routine for the data.
def preprocess(sentence):
    sentence=sentence.lower()
    sentence="".join([x for x in sentence if x not in
string.punctuation])

    sentence=[x for x in sentence.split(" ")]
    if x not in stopwords.words('english')]
        sentence=[x for x in sentence if x!=""]
    return " ".join(sentence)

#Fit a bag of words estimator and transform the count matrix.
bow_vectorizer = CountVectorizer(lowercase=True,preprocessor=pre-
process)
model = bow_vectorizer.fit(Text)
bag_of_words=model.transform(Text)

#Get the frequencies of the words.
bow = bag_of_words.todense()
#Get the words in the corpus.
words = model.get_feature_names()

#See the details of the estimator values.
print(bow.shape) # prints (14, 159)
print(len(words)) # prints 159

```

In the above code, a bag of words vector is generated for every sentence. In the last lines of the code, the shape of the BOW vector can be seen. It is a 2D matrix of shape 14×159 . Here, 14 indicates the sentences in the corpus, whereas 159 indicates the total number of words in the corpus.

Example on BOW

A simpler example to understand how BOW works is considered in this section. Consider the following sentences about computers:

Table 12.1 BOW vector for computer example

Words	Sentence 1	Sentence 2
Automate	0	1
Computer	1	1
Device	1	0
Execution	0	1
Instructed	1	0
Instructions	1	1
Labor	0	1
Manual	0	1
Perform	1	0
Specified	1	0
Unmatching	0	1
Used	0	1

*Computer is a device that can be instructed to perform specified instructions.
Computer is used to automate manual labor through unmatching instructions execution.*

On applying the bag of words model, there are 12 words in the two sentence corpus described above. Once the frequency of each word in the corpus is computed, following BOW vectors obtained as shown in Table 12.1.

Table 12.1 describes the BOW vectors for the provided corpus. Each column of the vector represents the frequency of the word in the document or sentence provided. In Table 12.1 of the BOW vector, most of the entries will be zeroes since all the words in the corpus are represented in each vector. This representation becomes quite unrealistic if the number of words in the data becomes large. For example, if we take a corpus of 1000 sentences and 5000 words, we would get a vector of 1000×5000 entries which would need 2 GB of memory which would be very hardware intensive task. In order to reduce the memory overhead, the sparse matrix representation is used. Sparse matrix representation is that we store the indices of the entries whose values are nonzero.

The following code represents the generation of BOW vector for another corpus of text. In this way, the BOW vector model can be built on the given corpus information of text.

```
from nltk.corpus import stopwords
import string
from sklearn.feature_extraction.text import CountVectorizer
#load dataset
Text = [
    "PretzelBros, airbnb for people who like pretzels, raises
    $2 million",
    "Top 10 reasons why Go is better than whatever language you use.",
    "Why working at apple stole my soul (I still love it though)",
```

```

“80 things I think you should do immediately if you use Python.”,
“Show HN: carjack.me -- Uber meets GTA”
]

def preprocess(sentence):
    sentence=sentence.lower()
    sentence="".join([x for x in sentence if x not in string.punctuation])
    sentence=[x for x in sentence.split(" ") if x not in stopwords.words
('english')]
    sentence=[x for x in sentence if x!=""]
    return " ".join(sentence)

bog_vectorizer = CountVectorizer(lowercase=True,
preprocessor=preprocess)
model = bog_vectorizer.fit(Text)

bag_of_words=model.transform(Text)
print(bag_of_words.todense())
print(model.get_feature_names())

Output:
['10', '80', 'airbnb', 'apple', 'better', 'carjackme', 'go', 'gta', 'hn',
'immediately', 'language', 'like', 'love', 'meets', 'million', 'people',
'pretzelbros', 'pretzels', 'python', 'raises', 'reasons',
'show', 'soul', 'still',
'stole', 'things', 'think', 'though', 'top', 'uber', 'use', 'whatever',
'working']

```

12.6 Word to Vector Model

In the pre-processing of text analytics, once the bag of words and word cloud is created, it becomes easy for further processing. One of the alternative ways to create a word cloud model is using a word to vector model. In this section, word to vector model is presented with visualization.

The basic procedure of getting a vector model using the words in the text is to first pre-process the sentences using the nltk module in Python. The following code demonstrates creating a word to vector model for text analytics. The corpus considered for the example is the inbuilt *abc* of the *sklearn* module. The corpus *abc* consists of random text where the sentences are extracted first and then tokenized into words.

Once the tokenized words are obtained, the stopwords are removed from the list and a final word list is formed for further text analytics. The output of the tokenized words for the *abc* corpus is as shown below. It is a nested list where each list represents the words in the sentence. In this way, first the token of words are obtained for each sentence in the corpus.

```
from nltk.corpus import abc,stopwords
from string import punctuation
from gensim.models import Word2Vec
from sklearn.manifold import TSNE
import pandas as pd
import matplotlib.pyplot as plt

sents = abc.sents()
#print(sents[:10])
puncs = list(punctuation)
stop = set(stopwords.words('english') + puncs + [“”, “”])
processed_sents = []
for sent in sents:
    temp = []
    for word in sent:
        if word not in stop:
            temp.append(word.lower())
    processed_sents.append(temp)
print(processed_sents[:10])

[['pm', 'denies', 'knowledge', 'awb', 'kickbacks', 'the', 'prime',
'minister', 'denied', 'knew', 'awb', 'paying', 'kickbacks', 'iraq', 'despite',
'writing', 'wheat', 'exporter', 'asking', 'kept', 'fully', 'informed', 'iraq',
'wheat', 'sales'], ['letters', 'john', 'howard', 'deputy', 'prime',
'minister', 'mark', 'vaile', 'awb', 'released', 'cole', 'inquiry', 'oil',
'food', 'program'], ['in', 'one', 'letters', 'mr', 'howard', 'asks',
'awb', 'managing', 'director', 'andrew', 'lindberg', 'remain', 'close',
'contact', 'government', 'iraq', 'wheat', 'sales'], ['the', 'opposition',
'gavan', 'o', 'connor', 'says', 'letter', 'sent', '2002', 'time', 'awb',
'paying', 'kickbacks', 'iraq', 'though', 'jordanian', 'trucking',
'company'], ['he', 'says', 'government', 'longer', 'wipe', 'hands',
'illicit', 'payments', 'totalled', '290', 'million'], ['the',
'responsibility', 'must', 'lay', 'may', 'squarely', 'feet', 'coalition',
'ministers', 'trade', 'agriculture', 'prime', 'minister', ',”',
```

```
'said'], ['but', 'prime', 'minister', 'says', 'letters', 'show',
'inquiring', 'future', 'wheat', 'sales', 'iraq', 'prove', 'government',
'knew', 'payments'], [it', 'would', 'astonishing', '2002', 'prime',
'minister', 'i', 'done', 'anything', 'i', 'possibly', 'could', 'preserve',
'australia', 'valuable', 'wheat', 'market', '",', 'said'], ['email',
'questions', 'today', 'inquiry', 'awb', 'trading', 'manager', 'peter', 'geary',
'questioned', 'email', 'received', 'may', '2000'], [it', 'indicated',
'iraqi', 'grains', 'board', 'approached', 'awb', 'provide', 'sales',
'service', ".']]
```

For each of the processed sentences, initially now the vector model is obtained using the function *Word2Vec*. The parameters passed to this function are *size*, *min_count*, *workers*, and *iter*. Here, the size is specified as 300 indicating the number of sentences in the corpus considered. The *min_count* refers to the total number of words in the sentence. If the number of words in the sentence is 20, then the word to vector model is obtained.

Here, for interpreting the results, the word *government* is used. All the words in the corpus that are most similar to these words are added to the embeddings to the get the word vector model for it. It can be seen in the output that the words *court*, *governments*, *federal*, *industry*, *opposition* are taken into the word vector model with their occurrence of frequencies.

```
embeddings = Word2Vec(sentences=processed_sents, size=300, min_count=20,
workers=4, sg=0, iter=5, hs=0)
print(embeddings.wv.most_similar('government'))

[('court', 0.8545933365821838), ('governments', 0.7608055472373962),
('federal', 0.7528774738311768), ('industry', 0.7445326447486877),
('the', 0.7218483686447144), ('opposition', 0.7107964158058167),
('funding', 0.6966238021850586), ('inquiry', 0.6842386722564697),
('review', 0.6813172698020935), ('trade', 0.6796253323554993)]
```

Next, the words obtained in the embedding list are collected as one list called as *vocabulary*. The function *TSNE()* is used to obtain the word vector model for all the words in the corpus. It gives the distributed stochastic neighbor embedding model for the words in the corpus considered. The words that are similar are grouped into

one and stored in the vector. The output of the TSNE is as shown below where the *x* and *y* values can be used to see position of the words in the corpus.

```
vocab = list(embeddings.wv.vocab)
X = embeddings[vocab]
tsne_model = TSNE(n_components=2)
X_tsne = tsne_model.fit_transform(X)

data = pd.DataFrame(X_tsne, index=vocab, columns=['x', 'y'])
data = data[:100] # use only first 100 words.
print(data)
```

=====x===== == == == y

denies	41.000854	-38.010197
knowledge	-1.009654	11.519361
awb	33.187195	41.710598
kickbacks	34.419754	-36.837307
the	35.065098	38.840485
prime	50.779877	30.259434
minister	52.725185	38.467823
knew	-5.418348	-57.856190
paying	5.399355	-14.157794
iraq	18.697643	35.002403
despite	23.604540	37.188904
writing	32.363739	-21.261209
wheat	29.105671	42.581383
exporter	29.166904	39.915119
asking	-33.615059	12.525192
kept	-25.664696	-2.553294
fully	-16.702623	-36.173401
sales	0.915510	42.332363
letters	41.420280	1.516451
john	52.070244	37.122097
howard	49.365101	28.190823
deputy	43.894024	25.362448

mark	49.109886	31.493637
vaile	49.242855	27.371328
released	31.614101	35.871021
cole	25.016020	31.850077
inquiry	33.530052	41.403614
oil	22.474443	33.175205
food	22.632797	33.002705
program	32.533459	18.267725

Figure 12.4 represents the word to vector model for the code in Python. It can be observed from Fig. 12.4 that the words *released*, *exporter* are grouped closely than the other words like *denies*, *knew* in the word to vector model. The model shown in Fig. 12.4 is for only 25 words in the corpus. For the entire corpus, the visualization needs to be divided into subsets.

In this section, the example discussed demonstrated the usage of word to vector model. The word cloud model for applications like spam classification and sentimental analysis is discussed in the next section.

12.7 Word Cloud Model

Word clouds commonly referred to as tag clouds represent graphically the word frequency appearing in a text file. It gives prominence to the words that appear most frequently in the text. Larger the font size in the word cloud, more frequently the word appears in the text. Visualization of content of a file using word cloud helps in identifying a set of most frequent words in major documents like interviews, formal

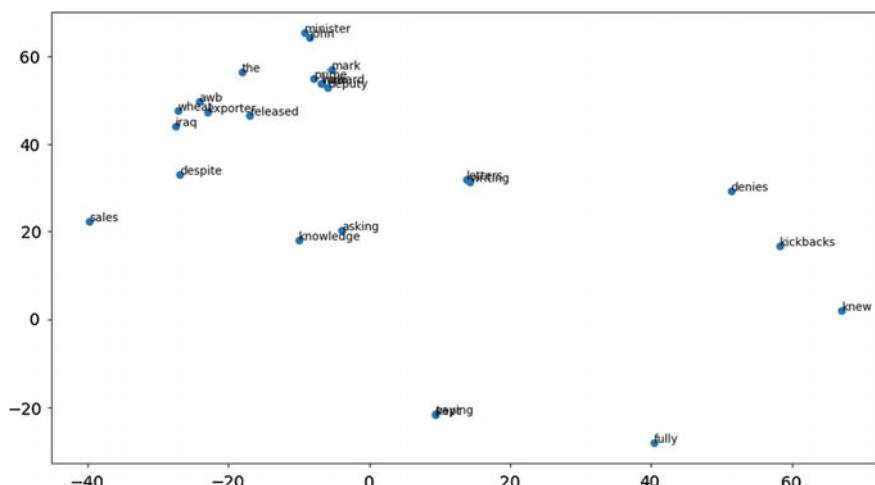


Fig. 12.4 Word to vector model

reports, etc. The main aim of word cloud is it can be used to communicate the thematic points of the text.

Various applications such as spam classification and automatic summarization include word cloud as the beginning phase for exploration of the data. For example, in the case of spam classification building the word cloud model gives the word that occurs most frequently in the text that can be either spam/non-spam. In the same way, reviews of different products for recommendation systems can be used to build a word cloud model for identifying the keywords used in the recommendation. These applications are dealt separately in the upcoming chapters of this part.

12.7.1 Word Cloud Model in Python

Word cloud models are very helpful in analyzing the various insights about a document. For example, in the case of a small documentary on a natural disaster like ‘tsunami’, a typical word cloud would be helpful in searching for the meanings of some important words related to ‘tsunami’. Since, various applications can make use of word cloud for analysis in this section a small example on the word cloud model in Python is discussed.

The following code demonstrates the word cloud model in Python. Initially, the modules required for the word cloud model are imported. The corpus of words from the nltk module is imported for stopwords and tokenization of words. A file consisting of random text is considered here for building the word cloud model. All the punctuation marks are removed from the text in the file as the first step. Each line of the file is read into word cloud for generating the data. The data are then fed into the word cloud module of nltk. The matplotlib module is used to plot the word cloud model as shown in Fig. 12.5.

In Fig. 12.5, the words that appear more frequently appear in large font size than the other words in the text file. For example, the words brain and artificial appear large in size than the other words. Another analysis that can be found from the word cloud model is the content of the file is more related to AI, machine, cognitive, human, and brain.

```
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from nltk.tokenize import sent_tokenize,word_tokenize
from nltk.corpus import stopwords
from string import punctuation
```

Fig. 12.5 Word cloud model



```
from nltk.probability import FreqDist
puncs = (list(punctuation))
puncs.extend(['“', “”, “”'])

with open('files/matter.txt') as f:
    data = f.read()
wordcloud = WordCloud()
wordcloud.generate(data)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In the previous code of building the word cloud model, the sentences read from the file were not tokenized into individual words. It will be interesting to see what happens when the sentences are broken down into words and then the word cloud model is built. The following code demonstrates building the word cloud model using tokenization approach.

Once a line is read from the file, each word in the line is extracted into tokens using the code below. The `sent_tokenize()` is the method used by converting the sentences into lower case. Each word extracted now might be the case of stopwords/punctuations. Hence, first the words are appended to a list, and then the stopwords and words with punctuations are removed for generating the data for word cloud. For the final word list, the word cloud model is built using the `Wordcloud()` module.

```
with open('files/matter.txt') as f:
    data = f.read()
sentences = sent_tokenize(text=data.lower().strip())
words = []
for i in sentences:
    words.extend(word_tokenize(i))
stop = set(stopwords.words('english') + puncs )
final_words = []
for word in words:
    if word not in stop:
        final_words.append(word)
table = FreqDist(final_words)
wordcloud = WordCloud()
wordcloud.generate_from_frequencies(table)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Fig. 12.6 Word cloud model with tokenization



The output of the word cloud model is as shown in Fig. 12.6. It can now be observed in Fig. 12.6 that machine is not highlighted with larger font size as in the previous word cloud model. However, other words artificial, brain, and human font sizes remain the same. In this way, the word cloud model can be built efficiently using the tokenization approach.

So far, in the previous sections the basic steps necessary for text analytics like word cloud model, word to vector model, bag of words were discussed with examples. These steps are pre-processing steps that are needed for every text analytics problem that need to be studied. Once these steps are executed, further machine learning techniques can be employed for advanced analytics.

In the upcoming sections, case studies on text analytics such as automatic summarization, spam classification, and question classification are discussed with Python examples. For these case studies, open datasets are used and hence can be used as examples for simple analysis. However, it can be further extended to larger problems for analysis.

12.8 Automatic Summarization Case Study

Natural language processing (NLP) is a domain area in computer science that is related to human and computer interaction. It is now tending toward artificial intelligence (AI) and has become the core integral part of AI and automation. The current world of AI in every aspect requires large volume of information and understanding. For example, speech recognition needs lot of information about the taxonomy and semantics of processing. In this chapter, the focus is on summarization which is one of the key roles of NLP.

Summarization can be defined as the creation of small version of text from large version using automation. The increase in information around the Internet has lead to the topic of summarization. When information is required and searched in the Internet, voluminous information is presented instead of quick short summary of it [4]. For example, the key term ‘network data analytics’ refers to large amount of information, and a search in the Internet does not give a short summary that gives the key concepts on it.

A summary of text can be defined as ‘text that conveys the information about one or more documents and usually half the size of the original document.’ The general examples of summarization are news headlines, movie reviews, meeting minutes, etc. In the current era of Internet world, summary of information plays a key role in defining the qualitative information. In this section, a case study on the summarization is discussed in Python. Before diving into the code, some of the key aspects of summarization are discussed as below.

12.8.1 Supervised and Unsupervised Summarization

In supervised method, the summarization of text is obtained from the training text dataset. The corpus of the information includes information elements like dictionary and grammar text. The basic approach followed in the supervised approach is that certain key phrases are characterized and ranked. Summary of information is obtained based on these key phrases in the testing document. The drawback of this approach is that supervised methods are domain specific. For example, if the training set includes corpus of information on scientific discoveries, summary will be concentrated on it only. When other texts based on movies, news are given summary obtained may not be accurate.

In the case of unsupervised method, statistical techniques are used for obtaining the summary of the text. The content of the text is divided into small pieces of information where each chunk of the information is ranked according to a weight function matrix. Based on the ranking of the contents, the information of the summary is obtained. The drawback of the domain basis in supervised approach is eliminated here using ranking approach. Since each time the corpus of information is built around the different information in the text, it is domain independent.

12.8.2 Abstract and Extractive Summarization

The abstraction of information from various documents is a complex approach. For example, if we read an article in a newspaper and rewrite an abstract for the same, sometimes the context and the meaning might not appear right. In the same way when a machine learning model needs to be used for summarization, it involves a set of steps with complex process. The abstractive way of summarization searches for the general phrases in the text and a lexical model is built first for summarization. The summary of the text is obtained from the lexical chain of information of the text. For example, in a compiler lexical analysis is performed to identify the errors in a program and point to the line of error.

In extractive method of summarization, a subset of words, sentences, and paragraphs are taken from the original text to form the summary of the text. Some of the typical phases of extractive summarization methods are tokenization,

sentence extraction, ranking of sentences, and then finally the summary. The substeps of the extractive summarization method are integrated one by one for forming a summary. For example, the words that are extracted in the phase of tokenization are used in sentence extraction for forming the summary.

12.8.3 Sentence Extraction

The different types of summary as discussed in the previous section mainly involve sentence extraction as the basic entity for summarization. In this step of summarization, the sentence extracted need to be scored and ranked for generating the summary. The key concepts that are involved in sentence extraction are as follows:

- **Keyword occurrence:** Certain keywords in the sentences highlight the importance of the sentence and thus can be used for scoring and ranking. For example, words such as ‘because’, ‘hence’, ‘therefore’ are some of the keywords that mark the sentence as important and can be used for summarization.
- **Location-heuristic:** In certain domain-specific documents, the location of the documents represented by the headings can be used for finding useful information about the text. Generally, abstract and conclusion are the key locations where most of the information is present and the same can be used for summary.
- **Pronouns:** The sentences with more pronouns are not used as a part of the summary as generally it involves redundant information. For example, consider the text ‘Data Analytics is a process of analyzing data with machine learning methods. It involves various steps in arriving the final results’, where the second sentence with pronoun can be excluded in the final summary.
- **Scoring:** The sentences with the key phrases are scored according to the corpus information available in the text. Once the summary is generated using the scoring approach, then if the summary is not satisfied, rescoring needs to be done once again for the summary.

With the prerequisites of different methods of summarization, autosummari-zation in Python is presented in the next section.

12.8.4 Auto Summarization in Python

The following code demonstrates autosummari-zation in Python using extractive technique of summarization and nltk module.

Firstly, the modules required for summarization are snt_tokenize, stopwords, punctuation, and FreqDist. These modules are imported first using the nltk library. In the text, punctuations need to be removed and thus a list of punctuations are imported from the inbuilt module of Python and initialized into a variable puncs.

The file with the text to be summarized is opened, and the lines are read. Each line reads from the file is converted to lower case and tokenized into words.

All the words extracted from the sentence are appended to a list of words. From this list of words, stopwords need to be removed. The stopwords are removed using the words method, and the parameter passed to it is ‘english’. The final word list is prepared by comparing the ‘words’ list and ‘stop’ list. This list is used for generating the summary of a text.

In the next phase, a frequency distribution of the final words is prepared for ranking the sentences. A table of sorted words with their frequencies is prepared. Using this table, each sentence in the extracted text is compared. If the sentence contains the word, then rank is increased. In this way, a dictionary ‘sent_ranks’ is prepared that contains the rank for each sentence.

The last phase contains the code for generating summary. The dictionary ‘sent_ranks’ is first sorted based on the items. The dictionary is then reversed to get the descending order of sentences, i.e., highest ranked sentence is at the beginning of the dictionary. For each sentence in the dictionary that is ranked first, sentences are extracted one by one for generating the summary.

```
from nltk.tokenize import sent_tokenize,word_tokenize
from nltk.corpus import stopwords
from string import punctuation
from nltk.probability import FreqDist
puncs = (list(punctuation))

with open('matter.txt','r') as f:
    content = f.read()
sentences = sent_tokenize(text=content.lower())

words = []
for i in sentences:
    words.extend(word_tokenize(i))

stop = set(stopwords.words('english') + puncs )

final_words = []
for word in words:
    if word not in stop:
        final_words.append(word)

table = FreqDist(final_words)
ranked_words = sorted(table,key=table.get)

sent_ranks = {}
for sent in sentences:
    w = word_tokenize(sent)
```

```

rank = 0
for word in w:
    if word in ranked_words:
        rank = rank + ranked_words.index(word)
sent_ranks[rank] = sent

final_sents = sorted(sent_ranks.items())
final_sents.reverse()
final = []
for item in final_sents[0:10]:
    final.append(sentences.index(item[1]))
for index in sorted(final):
    print(sentences[index] + '\n')

```

The outputs of the summarization are as shown in the following Figs. 12.7, 12.8, and 12.9. In the plots, the underlined sentences are selected for the summarization. Figure 12.7 shows the summarization of AI corpus information. Figure 12.8 shows the summarization of Google corpus information. Figure 12.9 shows the summarization of brain corpus information. It can be seen clearly from the plots that specific sentences are picked up for the summarization which are underlined. In this way, automatic summarizations can be obtained for other corpus information of the text using Python.

Output of the summarization

In artificial intelligence, an intelligent agent (IA) is an autonomous entity which observes through sensors and acts upon an environment using actuators (i.e. it is an agent) and directs its activity towards achieving goals (i.e. it is "rational", as defined in economics). Intelligent agents may also learn or use knowledge to achieve their goals. They may be very simple or very complex: a reflex machine such as a thermostat is considered an example of an intelligent agent.

Intelligent agents are often described schematically as an abstract functional system similar to a computer program. *For this reason, intelligent agents are sometimes called abstract intelligent agents (AIA) to distinguish them from their real world implementations as computer systems, biological systems, or organizations. Some definitions of intelligent agents emphasize their autonomy, and so prefer the term autonomous intelligent agents. Still others (notably Russell & Norvig (2003)) considered goal-directed behavior as the essence of intelligence and so prefer a term borrowed from economics, "rational agent".*

Intelligent agents in artificial intelligence are closely related to agents in economics, and versions of the intelligent agent paradigm are studied in cognitive science, ethics, the philosophy of practical reason, as well as in many interdisciplinary socio-cognitive modeling and computer social simulations.

Intelligent agents are also closely related to software agents (an autonomous computer program that carries out tasks on behalf of users). In computer science, the term intelligent agent may be used to refer to a software agent that has some intelligence, regardless if it is not a rational agent by Russell and Norvig's definition. For example, autonomous programs used for operator assistance or data mining (sometimes referred to as bots) are also called "intelligent agents".

Fig. 12.7 Automatic summarization for AI corpus

Google Inc. is an American multinational technology company that specializes in Internet-related services and products. These include online advertising technologies, search, cloud computing, software, and hardware. Google was founded in 1995 by Larry Page and Sergey Brin while they were Ph.D. students at Stanford University, in California. Together, they own about 14 percent of its shares, and control 56 percent of the stockholder voting power through supervoting stock. An initial public offering (IPO) took place on August 19, 2004, and Google moved to its new headquarters in Mountain View, California, nicknamed the Googleplex. Upon completion of the restructure, Sundar Pichai was appointed CEO of Google; he replaced Larry Page, who became CEO of Alphabet. The company's rapid growth since incorporation has triggered a chain of products, acquisitions, and partnerships beyond Google's core search engine (Google Search). It offers services designed for work and productivity (Google Docs, Sheets and Slides), email (Gmail/Inbox), scheduling and time management (Google Calendar), cloud storage (Google Drive), social networking (Google+), instant messaging and video chat (Google Allo/Duo), language translation (Google Translate), mapping and turn-by-turn navigation (Google Maps/Waze), video sharing (YouTube), notetaking (Google Keep), and photo organizing and editing (Google Photos). The company leads the development of the Android mobile operating system, the Google Chrome web browser, and Chrome OS, a lightweight operating system based on the Chrome browser. Google has moved increasingly into hardware: from 2010 to 2015, it partnered with major electronics manufacturers in the production of its Nexus devices, and in October 2016, it released multiple hardware products (including the Google Pixel smartphone, Home smart speaker, WiFi mesh wireless router, and Daydream View virtual reality headset). The new hardware chief, Rick Osterloh, stated: "a lot of the innovation that we want to do now ends up requiring controlling the end-to-end user experience". Google has also experimented with becoming an Internet carrier. In February 2010, it announced Google Fiber, a fiber-optic infrastructure that was installed in Kansas City; in April 2015, it launched Project Fi in the United States, combining Wi-Fi and cellular networks from different providers; and in 2016, it announced the Google Station initiative to make public Wi-Fi available around the world, with initial deployment in India. Alexa, a company that monitors commercial web traffic, lists Google.com as the most visited website in the world. Several other Google services also figure in the top 100 most visited websites, including YouTube and Blogger. Google's mission statement, from the outset, was "to organize the world's information and make it universally accessible and useful", and its unofficial slogan was "Don't be evil". In October 2015, the motto was replaced in the Alphabet corporate code of conduct by the phrase "Do the right thing".

Fig. 12.8 Automatic summarization for Google corpus

An artificial brain (or artificial mind) is software and hardware with cognitive abilities similar to those of the animal or human brain.
Research investigating "artificial brains" and brain emulation plays three important roles in science: An ongoing attempt by neuroscientists to understand how the human brain works, known as cognitive neuroscience.
A thought experiment in the philosophy of artificial intelligence, demonstrating that it is possible, at least in theory, to create a machine that has all the capabilities of a human being.
A long term project to create machines exhibiting behavior comparable to those of animals with complex central nervous system such as mammals and most particularly humans. The ultimate goal of creating a machine exhibiting human-like behavior or intelligence is sometimes called strong AI.
An example of the first objective is the project reported by Aston University in Birmingham, England where researchers are using biological cells to create "neurospheres" (small clusters of neurons) in order to develop new treatments for diseases including Alzheimer's, motor neurone and Parkinson's disease.
The second objective is a reply to arguments such as John Searle's Chinese room argument, Hubert Dreyfus' critique of AI or Roger Penrose's argument in The Emperor's New Mind. These critics argued that there are aspects of human consciousness or expertise that can not be simulated by machines. One reply to their arguments is that the biological processes inside the brain can be simulated to any degree of accuracy. This reply was made as early as 1950, by Alan Turing in his classic paper "Computing Machinery and Intelligence".
The third objective is generally called artificial general intelligence by researchers. However, Ray Kurzweil prefers the term "strong AI". In his book The Singularity is Near, he focuses on whole brain emulation using conventional computing machines as an approach to implementing artificial brains, and claims (on grounds of computer power continuing an exponential growth trend) that this could be done by 2025. Henry Markram, director of the Blue Brain project (which is attempting brain emulation), made a similar claim (2020) at the Oxford TED conference in 2009.

Fig. 12.9 Automatic summarization for brain corpus

12.9 Spam Classification Case Study

The data that revolve around the Internet are generally unstructured. The sources of such data are images, e-mail, audio, and video. Algorithms that scrape and analyze on such data need to have more capabilities in understanding the domain. For example, consider a mailbox where there is a lot of mails and if one has to segregate manually into spam/non-spam. It becomes a tedious process for humans to carry out such process. Machine learning methods of classification can be employed here for analysis of spam/non-spam.

The basic methods of spam classification can be categorized into two approaches, namely content based and non-content based. In the content-based approaches, text classification methods such as clustering, SVM, logistic regression are employed. The contents are first classified as spam/non-spam first, and then the actual spam classification is carried out. The basic examples of classification methods were discussed as a part of this book in Part 2. In the non-content-based approaches, the contents are not available in hand for analysis. For example, in the case of social networks since the content is unstructured, the information cannot be categorized into spam/non-spam easily. It requires many steps for classification.

In this section, a spam filter is constructed for a dataset [7]. A spam filter is constructed with Naïve Bayes, SVM, and neural network methods. Each of the methods for classification is discussed with results. A comparison among all the three methods is finally made in the end where neural network outperforms all the other methods. The main aim of this section is to get familiarize with the machine learning methods for spam classification.

12.9.1 *Spam Classification with Naïve Bayes in Python*

In this section, classification model for identifying spam/non-spam messages is discussed with an example in Python [5, 6]. Initially, as explained in the earlier sections the text needs to be pre-processed first by removing the stopwords and building the word cloud. Firstly, the word cloud in Python is built and then the spam filter is modeled. The dataset is considered for the spam classification from [7].

12.9.2 *Spam Word Cloud in Python*

The word cloud for the dataset considered is implemented before pre-processing the text and after pre-processing. This gives a better clarity of pre-processing and word cloud. In the first section, the code for implementing word cloud before pre-processing is presented, and then the word cloud is presented for the after pre-processing part.

12.9.2.1 Word Cloud Before Pre-processing

In the following code, the modules of nltk, WordCloud, TfidfVectorizer, Stemmer, and GaussianNB are imported first. The dataset from the file *spam.csv* is loaded, and the unwanted columns are removed. These columns are ignored for building the word cloud as they do not signify any important meaning.

The word cloud is built for the sentences that are identified as spam in the training dataset using the spam text. In the same way, the word cloud is built for the sentences that are identified as ham in the training set. The word cloud model in Fig. 12.10 is for spam data points, and the word cloud model in Fig. 12.11 is for ham data points.

In this section of the code, the text from the dataset was not pre-processed, i.e., stopwords and punctuations were not removed. So, it can be seen from the word cloud model of spam data points in Fig. 12.10 that some of the stopwords such as will and please are present. These words do not have a significant meaning, and thus pre-processing needs to be done.

```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
import string
import pandas as pd
from nltk.stem.porter import PorterStemmer
from nltk.stem import SnowballStemmer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix
import matplotlib.pyplot as plt
from wordcloud import WordCloud
```

Fig. 12.10 Word cloud model before pre-processing for spam data points



Fig. 12.11 Word cloud model before pre-processing for ham data points



```

dataset=pd.read_csv("spam.csv",encoding='latin')
dataset = dataset.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"],axis=1)

x=dataset.copy()
spam=x[x.v1=="spam"]
spam=spam.v2
spam_text=".".join(spam)
wordcloud_spam = WordCloud().generate(spam_text)
plt.imshow(wordcloud_spam)
plt.axis("off")
print("The spam word cloud is:-")
plt.show()

ham=x[x.v1=="ham"]
ham=ham.v2
ham_text=".".join(ham)
wordcloud_ham = WordCloud().generate(ham_text)
plt.imshow(wordcloud_ham)
plt.axis("off")
print("The not spam word cloud is:-")
plt.show()

```

12.9.2.2 Word Cloud After Pre-processing

In this section, the code first pre-processes the text considered in the previous section by removing the stopwords and punctuation. The word cloud models for spam and ham data points are modeled using the same approach as seen in the earlier section. The PortStemmer() method from the nltk module is used for stemming the words in the text. For example, the words fishing and fisher are reduced to the root word ‘fish’.

The word cloud model after pre-processing with spam and ham data points is as shown in Figs. 12.12 and 12.13. From Fig. 12.12, it can be observed that the stopwords are not present in the word cloud model. Once the pre-processing is done, the word cloud appears more clear than the one without pre-processing. We can see from Fig. 12.12 of spam data points where the words ‘call’ and ‘free’ are highlighted with greater font size. These are the generally used phrases that identify the message as spam. Hence, using the word cloud model these types of certain phrases identifying as spam can be seen.

Fig. 12.12 Word cloud model after pre-processing for spam data points



Fig. 12.13 Word cloud model after pre-processing for ham data points



```
def preprocess(sentence):
    stemmer=PorterStemmer()
    sentence=sentence.lower()
    sentence="".join([x for x in sentence if x not in string.punctuation])
    sentence=[x for x in sentence.split(" ") if x not in stopwords.words('english')]
    sentence=[x for x in sentence if x!=""]
    sentence=[stemmer.stem(x) for x in sentence]
    return " ".join(sentence)

content=dataset['v2'].copy()
content=content.apply(preprocess)

x=dataset.copy()
spam=x[x.v1=="spam"]
spam=spam.v2
spam=spam.apply(preprocess)
spam_text=".".join(spam)
wordcloud_spam = WordCloud().generate(spam_text)
plt.imshow(wordcloud_spam)
plt.axis("off")
print("The spam word cloud is:-")
plt.show()

ham=x[x.v1=="ham"]
ham=ham.v2
ham=ham.apply(preprocess)
ham_text=".".join(ham)
wordcloud_ham=WordCloud().generate(ham_text)
plt.imshow(wordcloud_ham)
plt.axis("off")
print("The not spam word cloud is:-")
plt.show()
```

12.9.3 Spam Filter in Python

In this section a spam filter is modeled for the dataset considered. The following code demonstrates the spam filter in Python using Naïve Bayes classification method. Initially, the modules in nltk, Portstemmer, Snowballstemmer, and GaussianNB are loaded. The csv file ‘spam.csv’ is loaded as the main text file for classification. From the dataset, the columns 2, 3, and 4 are dropped as it is not needed for classification. The first ten rows of the data are as shown in Fig. 12.14.

The sentences in the third column from the dataset are extracted for classification. Initially, the PortStemmer() module is used for stemming the sentences with whitespace and other characters. The sentence is then converted into lower case, and then the punctuation and stopwords are removed. The sentences are then rejoined back for training the dataset for classification.

```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
import string
import pandas as pd
from nltk.stem.porter import PorterStemmer
from nltk.stem import SnowballStemmer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix

dataset=pd.read_csv("spam.csv",encoding='latin')
dataset = dataset.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"],axis=1)
dataset.head(10)

def preprocess(sentence):
    stemmer=PorterStemmer()
    sentence=sentence.lower()
```

Fig. 12.14 Dataset for spam classification

	v1	v2
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
5	spam	FreeMsg Hey there darling it's been 3 week's n...
6	ham	Even my brother is not like to speak with me. ...
7	ham	As per your request 'Melle Melle (Oru Minnamin...
8	spam	WINNER!! As a valued network customer you have...
9	spam	Had your mobile 11 months or more? UR entitle...

```

sentence="".join([x for x in sentence if x not in string.punctuation])
sentence=[x for x in sentence.split(" ") if x not in stopwords.words
('english')]
sentence=[x for x in sentence if x!=""]
sentence=[stemmer.stem(x) for x in sentence]
return "".join(sentence)

content=dataset['v2'].copy()
content=content.apply(preprocess)

tfidf_vectorizer = TfidfVectorizer("english")
tfidf_vectorizer.fit(content)
features = tfidf_vectorizer.transform(content)
features = features.todense()
features_train, features_test, labels_train, la-
bels_test = train_test_split(features, dataset['v1'], test_size=0.3,
shuffle=True)
model=GaussianNB()
model.fit(features_train,labels_train)
GaussianNB(priors=None)
test=dataset.sample(10).copy()
test_features=test['v2']
test_labels=test['v1']
test.rename(columns={'v1':'Actual_Class','v2':'Email Content'},
inplace=True)
test=test.reindex_axis(['Email Content','Actual_Class'],axis=1)
test

```

The joined sentences are applied for pre-processing, and the term frequency (TF) and document frequency (DF) are calculated. The nltk module of the vectorizer itself is used for this purpose, and then features for spam classification are extracted. The pre-processed dataset with no stopwords and punctuations is split into training and testing for classification. The test dataset size is set to 0.3 here.

A Naïve Bayes classification model is built using the GaussianNB() function. In order to see classification, a sample row is selected from the dataset with actual class as seen in Fig. 12.15. The pre-processed test features are applied to the model with TF and DF for classification. The predicted class of the dataset is as shown in Fig. 12.16. In Fig. 12.16, only sample rows are selected for viewing the output. The actual class and predicted class values are equal. But it does not mean the same for all the rows in the dataset. Hence, the accuracy score and confusion matrix are printed out. We can see the accuracy score is 88%, and the confusion matrix represents that

	Email Content	Actual_Class
1722	Thought praps you meant another one. Goodo! I...	ham
813	I borrow ur bag ok.	ham
671	SMS. ac sun0819 posts HELLO:\You seem cool	spam
2116	It's that time of the week again, ryan	ham
3568	She's fine. Sends her greetings	ham
2243	Boo I'm on my way to my moms. She's making tor...	ham
2228	Those were my exact intentions	ham
2696	Nope but i'm going home now then go pump petro...	ham
1312	I love u 2 babe! R u sure everything is alrite...	ham
3981	His frens go then he in lor. Not alone wif my ...	ham

Fig. 12.15 Dataset of spam classification with actual class

	Email Content	Actual_Class	Predicted_Class
1722	Thought praps you meant another one. Goodo! I...	ham	ham
813	I borrow ur bag ok.	ham	ham
671	SMS. ac sun0819 posts HELLO:\You seem cool	spam	spam
2116	It's that time of the week again, ryan	ham	ham
3568	She's fine. Sends her greetings	ham	ham
2243	Boo I'm on my way to my moms. She's making tor...	ham	ham
2228	Those were my exact intentions	ham	ham
2696	Nope but i'm going home now then go pump petro...	ham	ham
1312	I love u 2 babe! R u sure everything is alrite...	ham	ham
3981	His frens go then he in lor. Not alone wif my ...	ham	ham

Fig. 12.16 Dataset of spam classification with predicted class

```

test_features=test_features.apply(preprocess)
test_features=tfidf_vectorizer.transform(test_features)
test_features=test_features.todense()
model.predict(test_features)
test['Predicted_Class']=model.predict(test_features)
test

print("The confusion matrix:-\n",confusion_matrix(labels_test,model.
predict(features_test)))
print("accuracy ",accuracy_score(labels_test,model.predict
(features_test)))

```

The confusion matrix:-

```
[ [1284 163]
```

```
[ 28 197]]
```

```
accuracy 0.885765550239
```

In this section, spam classification was carried out using Naïve Bayes method and the accuracy was 89%. The confusion matrix shows that 1284 data points are correctly classified as ham whereas 163 are incorrectly classified as 163. In the next section, the same dataset is considered for spam classification using SVM method.

12.9.4 *Spam Classification with SVM*

SVM machine learning technique was discussed in the in Part 2 of this book. SVM technique is used for classification of datasets where multiple classes are involved. In this regard, spam classification is carried out using SVM method. Even though Spam classification appears as a binary classification problem, i.e., identifying whether it is spam/non-spam, SVM gives a better accuracy over the other machine learning techniques.

The same dataset considered in the earlier section is used. The procedure for classification remains the same except the model used. Initially, the pre-processing of the text is carried out using the stemmer, stopwords, and tokenizer modules. A linear support vector classification is used as a classifier model for this task. In addition to its computational efficiency, another advantage of this approach is interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier.

```
import nltk
from nltk.tokenize import word_tokenize
import numpy as np
import random
import pandas as pd
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
vectorizer=CountVectorizer(analyzer = "word",tokenizer = None,
preprocessor = None,stop_words = None,max_features = 5000)
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

df=pd.read_csv("spam.csv",encoding='latin-1')
#Read in the data into a dataframe
df=df[['v1','v2']]
df.columns=['label','sms']
```

```
#Function to split the text messages into into words
def create_lexicon(sent):
    sent=re.sub("[^a-zA-Z]","",sent)
    sent=sent.lower()
    all_words = word_tokenize(sent)
    lex=list(all_words)
    lex = [lemmatizer.lemmatize(i) for i in lex]

#Converting each word to its root word
lex = [w for w in lex if not w in stop_words]

#Removing stop words
lex = [w for w in lex if(len(w)>1)]

#Removing single letter words
return ("".join(lex))

rows,col=df.shape

split_msg=[]

#Creating new column in the dataframe
for i in range(0,rows):

    #which contains the lemmatized words of
    broken=create_lexicon(df['sms'][i])

    #each message with the stop words removed
    split_msg.append(broken)
df['split_msg']=split_msg

#Splitting data into training and testing set, 80% training
#data and 20% testing data
X_train,X_test,y_train,y_test=train_test_split(df['split_msg'],
df['label'],test_size=0.2)

train_data=vectorizer.fit_transform(X_train)
train_data=train_data.toarray()

test_data=vectorizer.transform(X_test)
test_data=test_data.toarray()

model=svm.SVC(kernel='linear')
model.fit(train_data,y_train)
```

Table 12.2 Accuracy scores of spam classification

Method	Accuracy score (%)
Naïve Bayes	88
SVM	95

```
#Testing the model
predicted = model.predict(test_data)
print ("Accuracy")
print (accuracy_score(y_test, predicted))
```

It can be seen from Table 12.2 the accuracy scores of the each of the method for spam classification case study. The results show that the classification accuracy is more with SVM than the Naïve Bayes classification method. This is because Naïve Bayes classification has many drawbacks.

The important drawbacks of Naïve Bayes classification method are listed as follows.

- Naïve Bayes is based on the assumption that data distribution is correct for any two independent features given in the dataset.
- If there are no class labels for the given input dataset, then the prediction is not accurate. For example, if there is a class as ‘Play = No’ and no instances are present in the training data, then for another feature that uses *Play* prior probability, the conditional probability will be 0.
- Naïve Bayes classification will be difficult for continuous variables of the dataset. In that case, the values need to be converted into discrete set using the probability density functions.

The advantages of SVM over Naïve Bayes are listed as follows.

- SVM avoids overfitting the data and helps to generalize the model in a better way.
- It can be used for real-world problems such as text and image classification, handwriting recognition, and bioinformatics and biosequence analysis.
- There is no upper limit on the number of attributes.

Since SVM outperforms Naïve Bayes in the listed points above, there is a better accuracy with SVMs in the spam classification case study. In this section of text analytics on the spam classification case study initially, the bag of words and word cloud model were created, and then the classifiers were modeled for the problem. In this modeling classification, SVM outperformed Naïve Bayes classification method as it has advantages over Naïve Bayes.

12.10 Question Classification Case Study

In this section, a case study is discussed on the question classification. The dataset considered here consists of a list of question and their categories [8, 9]. It is as shown in Table 12.3. It consists of only few questions in the dataset. As seen in Table 12.3, for each row, a question id, question, and category are present. The main aim of this case study is to classify the questions based on the category.

Table 12.3 Question classification dataset

Question	Category
Who did Arthur H. Bremer try to assassinate on May 15, 1972?	1
Doesn't ROS stand for return on sales?	0
How do you buy stocks?	3
Doesn't AAOS mean American Academy of Orthopedic Surgeons?	0
What was the infamous feat of Germany's U-2 submarine?	4
How fast is a 45 MHz processor?	3
Does Ahvaz lie in Iran?	2
What is the highest amount that can be got by using exactly 5 zeros?	5
What is the largest lake in North America?	4
Is there not a drug test that can detect any drug that is in your system?	4
What was the Great Britain population from 1699 to 1722?	5
What do I call the sons and daughters of my first cousins?	4
Is that lady a scientist?	1
Is there not a way to prevent someone from seeing my answers on Quora?	3
Is there any way for a student to get an internship at Microsoft or at Facebook from high school?	3
Does CASLPA mean Canadian Association of Speech-Language Pathologists and Audiologists?	0
Is that boy a celebrity?	1
What is the highest number that can be got by using exactly 5 ones?	5
Is there not a way to get Quora (or Web sites in general) to display with fonts of my own choosing in Chrome on Linux?	3
What does the acronym CPR mean?	3

The steps followed for the classification of the case study are listed as follows.

- Pre-processing
- Creating features
- Classification using SVM.

12.10.1 *Pre-processing*

In the pre-processing stage, the data are read from the csv file by removing the *id* and *label* columns. These columns are not necessary for analysis as they cannot be used for the classification on a standalone basis. All the words are tokenized using the tokenize function of nltk module. If the words are digits and punctuations, they should be ignored. In this regard, the tokens are checked to see whether there are digits and punctuations.

The text document of the questions was first split according to its constituent sentences. These sentences were further split into its constituent words, and then the question number or any punctuation was also removed in order to standardize the questions. The numbers in the questions itself were ignored because some of them included years such as ‘1680’ and these contributed classification of the sentences in most cases. After this, the words were reconstituted into a sentence and placed into a list of strings.

```
import nltk
import csv
import pickle
from nltk.classify.scikitlearn import SklearnClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC, LinearSVC

f=open('train_questions.txt','rU')
with open('train_labels.csv','rb') as k:
    reader=csv.reader(k)
    train_labels=list(reader)

#Reads in the training labels file
train_labels.remove(train_labels[0])

#removes 'id' and 'label' from the label file

train_data=f.read()

train_sent=train_data.splitlines()
train_sent.remove(train_sent[0])

#split the training set into its corresponding
#print len(train_sent)
final_set=[]
all_words1=[]
token=nltk.RegexpTokenizer(r'\w+')

#the word tokenizer that does not read in punctuation
all_words=token.tokenize(train_data)

#All words in the file are tokenized
for j in all_words:
    if j.isdigit() is False:

#Read in only non numerical words present in the entire train set
    all_words1.append(j)
e=0
for i in train_sent:
```

```

# Creates a list of list of lists with words of each question and the
words=[]

# corresponding label [0-6]
set1=[]
set2=[]
words=nltk.word_tokenize(i)
set1.append(words[2:])
set1.append(train_labels[e][1])
final_set.append(set1)
e=e+1

all_words2=nltk.FreqDist(all_words1)

#The frequency distribution of all of the words present in the train file
word_features=list(all_words2.keys())

#print len(word_features)

```

12.10.2 Creating Features

Since, the classifier is based on the bag of words model, this requires a numeric representation of the frequency of every word in all the questions. Bag of words is a rudimentary approach to text classification in this case question classification but because of its robust nature and its ability to work well smaller-sized data made it the perfect choice for the model for this particular task. For this the questions were then passed to the CountVectorizer function which split them into its corresponding matrix of token counts. This gives a representation of the frequency of all the words that occur in all the questions and thus gives a representation of what kind of words would put a particular question under a particular category or label. None of the features were removed due to the fact that most words occurred in more than 90% of the documents and losing these features would drastically affect the output of the classifier. Considering also the small size of the training data, any loss of features would affect the classifier performance. The training data were split into a 90:10 ratio for the actual training data and the testing data.

```

def find_features(sent):
    # Finding the features of each question and storing
    it as a dictionary
    words2=set(sent)
    features={}
    for w in word_features:
        features[w]=(w in words2)
    return features

```

```

featuresets=[(find_features(rev),category) for (rev, category)
in final_set]
# Finds all the features of all the questions present in the
training set and puts it in the form of a list

```

12.10.3 SVM Classifier

A linear support vector classification was used as a classifier model for this task. The ability of LinearSVC to handle multi label data by using the one-vs-rest scheme makes a better choice for the case study considered. Also known as one-vs-all, this strategy consists of fitting one classifier per class or label. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency, another advantage of this approach is interpretability. Since each class is represented by one and one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This solution works particularly well in this case because of the multilabel nature of the classification of questions (Table 12.4).

Table 12.4 Predicted category of questions

Question	Category
Who did Arthur H. Bremer try to assassinate on May 15, 1972?	1
Doesn't ROS stand for return on sales?	0
How do you buy stocks?	0
Doesn't AAOS mean American Academy of Orthopedic Surgeons?	0
What was the infamous feat of Germany's U-2 submarine?	4
How fast is a 45 MHz processor?	3
Does Ahvaz lie in Iran?	2
What is the highest amount that can be got by using exactly 5 zeros?	5
What is the largest lake in North America?	4
Is there not a drug test that can detect any drug that is in your system?	4
What was the Great Britain population from 1699 to 172?	5
What do I call the sons and daughters of my first cousins?	4
Is that lady a scientist?	1
Is there not a way to prevent someone from seeing my answers on Quora?	3
Is there any way for a student to get an internship at Microsoft or at Facebook from high school?	3
Does CASLPA mean Canadian Association of Speech-Language Pathologists and Audiologists?	0
Is that boy a celebrity?	1
What is the highest number that can be got by using exactly 5 ones?	5
Is there not a way to get Quora (or Web sites in general) to display with fonts of my own choosing in Chrome on Linux?	3
What does the acronym CPR mean?	3

```

training_set=featuresets[:2900]
testing_set=featuresets[2900:]
#Split of 80:20 for training and testing set
print "Training Classifier ....."
LinearSVC_classifier = SklearnClassifier(LinearSVC())
LinearSVC_classifier.train(training_set)
print "Accuracy"
print nltk.classify.accuracy(LinearSVC_classifier, testing_set)

```

Output

Training Classifier

Accuracy

0.88

12.11 Sentimental Analysis

12.11.1 Introduction

Social computing involves constructing the models for different activities that take place in the social networks. Innovative and intellectual application development is the main focus of the social network analysis. People share their opinions and views on different types of products or issues through social networking sites like Facebook, Google, Twitter, etc. Gaining insights into such comments on different products and views gives interesting conclusions in various domain areas like online retail, marketing, scientific surveys, job marketing, health care, customer marketing, and other fields [10, 11].

Opinions form the core part of human behaviors. Sometimes, when decision needs to be taken by the humans, they are dependent on the others' opinions. For example, if a television needs to be purchased, an opinion is asked among our friends to see whether the brand, configuration, warranty, and others are reliable. Similar in enterprises, the customer opinions on products play a key role in long-term assessment of the company.

Sentimental analysis is an ongoing field of research in computer science that analyzes the different types of content available in social networking sites. Information generated by the users in the form of tweets, status on social networking platforms like Google, Facebook are used to know the sentiments of the users on different issues of the world. Some of the companies use sentimental analysis for endorsing their brands for creating the awareness and reputation. Most of the data in the social networking sites are unstructured. It poses a great challenge in converting the unstructured data into a specific format for analysis.

Generally, when an opinion is needed for business application, a survey is collected for a set of products and this survey is posted back o the company Web site for the consumer to know the insights. But due to the growth of social media, it

is easy to approach to the consumers for getting the survey of products without the actual questionnaire. It is no longer needed to prepare a questionnaire, review it, select a set of users and ask them to provide the survey. However, the large amount of text in the social networking sites needs to be collected first and then analyzed to see the positive and negative opinions. The average human reader cannot get a bigger insight into such information.

Due to proliferation of social media as discussed, automated sentiment analysis systems are needed. In this section, sentimental analysis is discussed with the examples in Python. Before diving into the actual sentimental analysis using Python, some of the concepts related to sentimental analysis are discussed in the upcoming sections.

12.11.2 *Different Types of Sentimental Analysis*

Opinions can be shared in the form of documents, status, short text, audio, and video. Each format of the data shared has its own structure and needs to be broken down into small tokens for opinion mining. The conventional approach of extracting the data and storing them in relational database systems will not be helpful. The schema of the database system might need to be changed every time. Thus, alternative platforms and technologies are needed. However, the basic categories of sentimental analysis remain the same irrespective of the platform/technology. In this regard, the different levels of analysis that can be carried out are summarized as follows [11]:

- **Document level:** In this level of sentimental analysis, the main aim is to categorize opinion of the document whether it expresses positive or negative. For example, if there are a bunch of documents about the reviews of different products, can the system be able to categorize it into positive and negative opinions? The drawback of this analysis is that it can be carried on a single entity but not with multiple entities. The review has to be for a single rather and not for multiple products.
- **Sentence level:** In this type of analysis, each sentence is extracted from the document and classified as positive, negative, or neutral. Neutral opinion means there is no opinion given. It is based on the subjectivity of the information present in the sentence. Sometimes, there can be difficulty in understanding the objectivity and subjectivity of the sentence. For example, if the sentence is '*The car was bought last week but the wiper had fallen off*', here even though the subject is car, it does not imply the review for the car but only the objective of buying is present. In Python, *nltk* module has inbuilt methods for extracting the sentences, and corpus information is also available for building the word model for analysis.
- **Entity and aspect level:** In this level of analysis, particular entity in the sentence is analyzed. For example, consider a sentence '*the phone quality is good but not battery life*'. Here, the entity targets are phone quality and battery life. It

indicates a positive opinion on the phone quality and negative opinion on the battery life of it. In this way, for each sentence the entity targets need to be found first and then the opinions can be formed.

Other than these types of different sentimental analysis, *corpus-based analysis* can be carried out. Here, the sentimental analysis is carried out only on the basis of certain corpus information and specific to it. So, if any information outside the corpus is present in the product reviews, they can be ignored for analysis. In this way, different sentimental analysis can be carried out. In the next section, sentimental analysis is discussed with a sample dataset. The method used is sentence-level analysis. Although it is sentence-level analysis, initially the documents are collected as the sources for analysis.

12.11.3 Sentimental Analysis in Python

In this section, a case study on the sentimental analysis is discussed for a dataset obtained from [12]. The dataset contains sentences from three different Web sites such as IMDb, Amazon, and Yelp. For each Web site, there exist 500 positive and 500 negative sentences. Those were selected randomly for larger datasets of reviews. The attributes are text sentences, extracted from reviews of products, movies, and restaurants. The main aim of the case study is to build a model on these set of sentences and carry out sentimental analysis for other datasets.

The following code demonstrates the sentimental analysis in Python. Initially, the modules such as *nltk*, *word_tokenize*, *random*, *wordnetlemmatizer*, *wordcloud*, *stopwords* are imported. The main module used is *nltk* because sentimental analysis involves natural language processing. Initially, the reviews collected by each site such as Yelp, Amazon, and IMDb are extracted into a data frame using *df()* function. The data frames are concatenated to obtain the training set for sentimental analysis.

```
import nltk
from nltk.tokenize import word_tokenize
import numpy as np
import random
import pickle
from collections import Counter
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import TensorFlow as tf
import pandas as pd
import re
import csv
from wordcloud import WordCloud, STOPWORDS
```

```

import matplotlib as mpl
import matplotlib.pyplot as plt

df=pd.read_table('yelp_labelled.txt',names=('review','sentiment'))
df2=pd.read_table('imdb_labelled.txt',names=('review','sentiment'))
df3=pd.read_table('amazon_cells_labelled.txt',names=('review',
'sentiment'))
df=pd.concat([df,df2])
df=pd.concat([df,df3])

```

Once, the data frames are concatenated, lexical analysis is performed on each sentence in the concatenated dataset. To accomplish this, `word_tokenize()` is used to get each words in each sentence. The set of words is included in a list `lex`. A list of positive and negative words is prepared initially that identifies the sentiment of the sentence. In the dataset, each row is marked with 0 or 1, where 1 indicates a positive sentiment and 0 indicates a negative sentiment.

The sentences are iterated one-by-one to gather whether the sentence is tagged with positive or negative sentiment. Lemmatization is performed in the next step. The main aim of lemmatization is to map the sentences morphologically. For example, if the sentence is *the boy's car are different colors* then in lemmatization, it is mapped as *the boy car be different colors*. In this way, lemmatization is carried out for each sentence once the tokenized words are obtained.

```

def create_lexicon(sent,lex):
    sent=re.sub("[^a-zA-Z]","",sent)
    sent=sent.lower()
    all_words = word_tokenize(sent)
    lex+= list(all_words)
    return list(all_words)

lexicon = []
pos_words=[]
neg_words=[]
for index, row in df.iterrows():
    if(row['sentiment']==1):
        pos_words+=create_lexicon(row['review'],lexicon)
    else:
        neg_words+=create_lexicon(row['review'],lexicon)

lexicon = [lemmatizer.lemmatize(i) for i in lexicon]
w_counts = Counter(lexicon)
l2 = []
for w in w_counts:

```

```

if 2000 > w_counts[w] > 50:
    12.append(w)

13 = []
for i in 12:
    if len(i) > 1:
        13.insert(0, i)

```

In this module, the lexicons are prepared for the data in data frame 2, i.e., for Amazon reviews. Initially, the reviews are obtained from the data frame and tokenized into words. These words are then first converted into lowercase and appended to the features of the words to be compared for sentimental analysis.

```

def create_feature(df2, lexicon):
    featureset = []
    for l in df2['review']:
        current_words = word_tokenize(l.lower().decode('utf-8'))
        current_words = [lemmatizer.lemmatize(i)
    for i in current_words]
        features = np.zeros(len(lexicon))
        for word in current_words:
            if word.lower() in lexicon:
                index_value = lexicon.index(word.lower())
                features[index_value] += 1
        features = list(features)
        featureset.append(features)
    return featureset

```

The codes in the previous sections deal with the pre-processing of the text. Once the pre-processing is carried out, the data are split into training set and testing set where first 2500 sentences are used for training and another 2500 sentences are used for testing. The dataset is then iterated over the sentences to get the sentiment associated with it. From this iteration, a word cloud model of positive words and negative words are obtained in the next phase.

```

X_train=create_feature(df[:2500],13)
X_test=create_feature(df[2500:],13)

y_train=list(df['sentiment'][:2500])
for i in range(len(y_train)):
    l=[0]*2
    l[int(y_train[i])]=1
    y_train[i]=l

```

```

y_test=list(df[“sentiment”][2500:])
for i in range(len(y_test)):
    l=[0]*2
    l[int(y_test[i])]=1
    y_test[i]=l

wordcloud = WordCloud(background_color='white',
                      stopwords=STOPWORDS,
                      max_words=200,
                      max_font_size=40,
                      random_state=42
                     ).generate(str(pos_words))

print wordcloud
plt.imshow(wordcloud)
plt.axis('off')
plt.title("Positive word cloud")
plt.show()

wordcloud = WordCloud(background_color='white',
                      stopwords=STOPWORDS,
                      max_words=200,
                      max_font_size=40,
                      random_state=42
                     ).generate(str(neg_words))

print wordcloud
plt.imshow(wordcloud)
plt.axis('off')
plt.title("Negative word cloud")
plt.show()

```

The output of the plots for positive word cloud model and negative word cloud model is as shown in Figs. 12.17 and 12.18, respectively. It can be observed from Fig. 12.17 of positive word cloud model where the words such as *good*, *wonderful* indicate positive sentiments. Similarly, in Fig. 12.18 of the negative word cloud model, the words like *terrible* and *bad* indicate the negative sentiments of the sentences in the dataset.

The actual training of the model is done using the following code. A TensorFlow model is used for training and modeling of sentimental analysis where the three hidden layers are used. For each of the hidden layer, 2000 nodes are used. The *epochs* used is 500 which tunes the network to the best possible extent. All the hidden layers are encoded with the required training weights.

Fig. 12.17 Positive word cloud for sentimental analysis

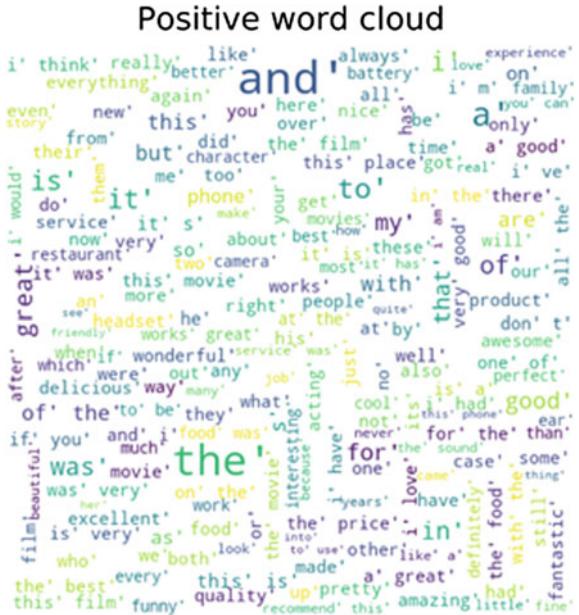
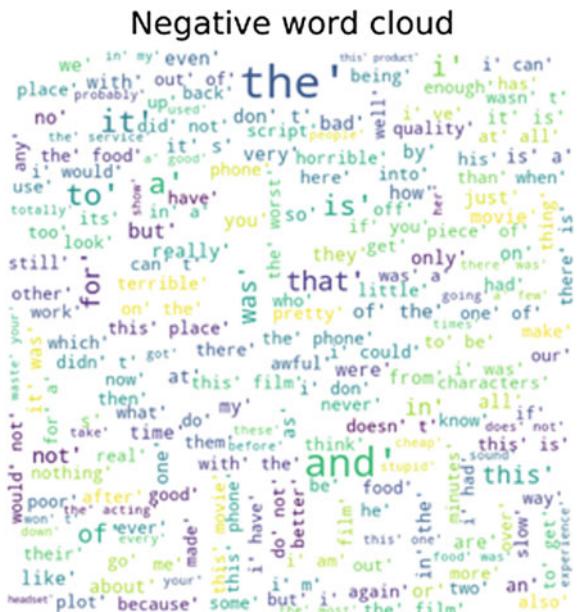


Fig. 12.18 Negative word cloud for sentimental analysis



```
hiddden_layer_1 = 2500
hidden_layer_2 = 2500
hidden_layer_3 = 2500

n_classes = 2
batch_size = 100
epochs = 500

x = tf.placeholder('float')
y = tf.placeholder('float')

hidden_1_layer = {'f':hiddden_layer_1,
                  'weight':tf.Variable(tf.random_normal
([len(X_train[0]), hidden_layer_1])),
                  'bias':tf.Variable(tf.random_normal
([hiddnen_layer_1]))}

hidden_2_layer = {'f':hidden_layer_2,
                  'weight':tf.Variable(tf.random_normal
([hiddnen_layer_1, hidden_layer_2])),
                  'bias':tf.Variable(tf.random_normal
([hidden_layer_2]))}

hidden_3_layer = {'f':hidden_layer_3,
                  'weight':tf.Variable(tf.random_normal
([hidden_layer_2, hidden_layer_3])),
                  'bias':tf.Variable(tf.random_normal
([hidden_layer_3]))}

output_layer = {'f':None,
                 'weight':tf.Variable(tf.random_normal
([hidden_layer_3, n_classes])),
                 'bias':tf.Variable(tf.random_normal
([n_classes])),}

def layers(data):
    layer_1 = tf.add(tf.matmul(data,hidden_1_layer['weight']),
                    hidden_1_layer['bias'])
    layer_1 = tf.nn.relu(layer_1)
```

```

layer_2 = tf.add(tf.matmul(layer_1,hidden_2_layer[‘weight’]),
hidden_2_layer[‘bias’])
layer_2 = tf.nn.relu(layer_2)

layer_3 = tf.add(tf.matmul(layer_2,hidden_3_layer[‘weight’]),
hidden_3_layer[‘bias’])
layer_3 = tf.nn.relu(layer_3)

output = tf.matmul(layer_3,output_layer[‘weight’]) + output_layer
[‘bias’]

return output

def train_model(x):
    pred = layers(x)
    cost = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits
(logits=pred,labels=y) )
    optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize
(cost)

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch in range(epochs):
            epoch_loss = 0
            i=0
            while i < len(X_train):
                start = i
                end = i+batch_size
                batch_x = np.array(X_train[start:end])
                batch_y = np.array(y_train[start:end])

                _, k = sess.run([optimizer, cost], feed_dict={x:
batch_x,y: batch_y})
                epoch_loss += k
                i+=batch_size

            print(‘Epoch’, epoch+1, ‘completed out of’, epochs, ‘loss:’,
epoch_loss)
            correct = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
            accuracy = tf.reduce_mean(tf.cast(correct, ‘float’))
            print(‘Accuracy:’ ,accuracy.eval({x:X_test, y:y_test}))
            return pred

# In[86]:
p=train_model(x)

```

In this way, the sentimental analysis can be carried out based on the reviews. The model considered here for the sentimental analysis has an accuracy score of 93%. In this section, a case study presented on the sentimental analysis showed the different steps involved in the analysis with an example.

12.12 Exercises

1. Why is removing stopwords an important step in the bag of words model?
2. Why do we remove very high-frequency and very-low frequency words when creating the feature set for question classification? What would happen if we did not remove them?
3. How does the pre-processing step in the spam classification using SVM help reduce the number of duplicate features?
4. How does SVM avoid overfitting of data and thus provide better and more reliable results as compared to other traditional machine learning models?
5. For the question classification case study, how would the accuracy be affected if there is a change to the type of penalty used from L2 to L1 in LinearSVC classifier that was used?
6. How does the ‘one versus rest’ classification tactic give higher accuracy when it comes to multi label classification? What is the result ‘one versus one’ classifier is used?
7. How does lemmatization of words affect the feature set? What would happen if words were not lemmatized?
8. For the sentimental analysis case study, try with the dataset provided by the link <http://ai.stanford.edu/~amaas/data/sentiment/>
9. Create the bag of words model for the dataset provided in <https://archive.ics.uci.edu/ml/datasets/bag+of+words>
10. Create the set of positive and negative word cloud models for the dataset provided in the question 8.

References

1. Aggarwal, C. C., & Zhai, C. (Eds.). (2012). *Mining text data*. Springer Science & Business Media.
2. Bird, S., & Loper, E. (2004, July). NLTK: The natural language toolkit. In *Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions* (p. 31). Association for Computational Linguistics.
3. Bird, S., Klein, E., & Loper, E. (2005). NLTK tutorial: Introduction to natural language processing. *Creative Commons Attribution*.
4. Neto, J. L., Freitas, A. A., & Kaestner, C. A. (2002, November). Automatic text summarization using a machine learning approach. In *Brazilian Symposium on Artificial Intelligence* (pp. 205–215). Springer, Berlin, Heidelberg.

5. Kim, S. B., Han, K. S., Rim, H. C., & Myaeng, S. H. (2006). Some effective techniques for naive bayes text classification. *IEEE Transactions on Knowledge and Data Engineering*, 18 (11), 1457–1466.
6. Youn, S., & McLeod, D. (2007). A comparative study for email classification. In *Advances and innovations in systems, computing sciences and software engineering* (pp. 387–391). Dordrecht: Springer.
7. Spam classification dataset: <http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/>.
8. Metzler, D., & Croft, W. B. (2005). Analysis of statistical question classification for fact-based questions. *Information Retrieval*, 8(3), 481–504.
9. Zhang, D., & Lee, W. S. (2003, July). Question classification using support vector machines. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 26–32). ACM.
10. Yu, Y., Duan, W., & Cao, Q. (2013). The impact of social and conventional media on firm equity value: A sentiment analysis approach. *Decision Support Systems*, 55(4), 919–926.
11. Fang, X., & Zhan, J. (2015). Sentiment analysis using product review data. *Journal of Big Data*, 2(1), 5.
12. <https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences>.

Chapter 13

Internet of Things and Analytics



In the past years, sensors were used for tracking data in retail stores, oil and gas industry. But, over the recent years a sudden hype is made on IoT. It is because, the sensors cost have drastically reduced and easily accessible. The bandwidth costs of Internet usage have drastically came down where each individual in the world can access all the information in the smartphone. In this chapter, an introduction to Internet of things (IoT) and analytics is discussed.

13.1 Introduction to IoT

IoT refers to the network of physical things that are connected to each other using the Internet as the backbone for communication. It is the connectivity and the communication among the objects that form the word ‘Internet of Things’ or ‘IoT.’ It is estimated that the number of things that are connected to each other will be at least more than 20 billion by 2020. As the volume of the data increases, the main concern lies in the storage, analysis, and making decisions based on the analysis. The scope of IoT is not only limited to this but it also involves concerns such as types of devices added to the network, types of data generated, scalability of the network, etc. [1, 2]. In this section, a brief overview of IoT and its components are discussed before analytics is discussed in the next sections.

The main aim of IoT is to gather information from various devices and different domains and offer service based on the data available. Everyday objects termed as ‘Things’ in IoT can offer services based on the data they collect by addition of capabilities such as assigning a virtual address space, self-organization and communication among the other devices. For improving the quality of services, additional capabilities such as awareness of the context, autonomous, and others need to

be incorporated. The services need to be offered without human intervention. RFID is one of the popular sensors used for identifying things, understand the context of the environment of things, and take necessary actions. For example, an RFID can be used in a warehouse for a product tracking. It can be used to see if it has left the warehouse and automatically update the inventory in the warehouse database without the human intervention. Thus, sensors such as RFID transfer the physical entities into virtual address space for services. The application logic part of the device is then transformed into *smart environment* such as smart homes, smart city, smart grid, smart power. In this way, IoT helps to offer services with things communicating to each other.

Long back the computer cost was high when it was invented and took more space, an entire room to fit in and was also complex to use. But, due the advances in both hardware and software, computers can be carried ranging from homes, cars, and offices. People around the globe are able to leverage the benefits of such advances in technology which are easy to use and less expensive. IoT helps in enabling the people to use more services from the advances in technology by connecting the existing machines and things to the cloud for streamline processing. The important question that will arise is why IoT now?

It is mainly because due to the advances in the hardware, IoT has become more popular now [2]. However, some of the key highlights that made IoT popular now days are as follows:

- **Innovation in mobiles:** One of the main reasons for the adoption of IoT in many fields of computing and engineering is the innovations in mobile technology. People around the globe started migrating from the traditional desktop browsing to mobile browsing and no company could turn down this trend. So, it was a crucial development of migration of software services that run in the backend to adapt to the mobility and deliver the services dynamically. Though, it may be overstated or hyped in the digital world, it is still the critical innovations that lead to the adoption of IoT.
- **Open APIs:** Due to the proliferation of social media and other e-commerce platforms such as Facebook, Twitter, Amazon, Instagram, Flickr, large number of APIs were used by these platforms. As the number of users grew in each one of these platforms, new innovations and creative ideas were needed to expand the usage of the platforms and thus APIs used by them were opened for the developers and integrate with their applications. Thus, open APIs played a major role in developing android-based applications that require for integration of services into various other applications.
- **Ease of coding:** Earlier years of computing in the areas of embedded programming, mobile device application development needed experts from both the areas for developing an application. But, due to the advances in programming languages and scripting such as Python, Julia, R, node.js, and others, it has now become easy for programmers to learn the programming language on their own and need not depend on others while developing an application.

- **Easy of availability:** Ten years ago, IoT was used with the help of small motes which was expensive. But now, sensors are integrated into a single board where multiple sensors can be integrated on a single board and analysis can be carried out easily. The data services needed to be used with cloud also used to be expensive compared to the current services available with Amazon EC2, S3 services on the cloud.
- **Rise of data science:** Many areas of computing were addressed in diverse fields of engineering such as data science, cloud computing, machine learning, artificial intelligence, and others. Due to the diverse interest in interdisciplinary engineering fields, IoT also came into the picture of main role and continued to be integrated into many other fields of computing.

Even though, there are many other reasons of IoT coming into picture of current world, the above key points highlight the important ones. The other challenges such as privacy, security, interoperability, user-friendly design coexist with IoT development. These challenges are research based and thus not encountered as a part of this book.

13.2 Components of IoT

The architectural elements of IoT system depend on the domain and the applications involved. Some of the examples of IoT where architecture reference model is used for developing applications are smart home, smart traffic, smart health, and smart transportation. Rather, than calling it as the architectural framework or the reference model, the components become a necessary part of IoT. In this section, the different components of IoT are discussed based on Fig. 13.1.

The necessary components of IoT are sensors, cloud, networking, and finally the analytics [3]. The basic IoT devices and sensors form the low-level component of IoT where various sensors such as camera, video surveillance, GPS, smart home sensors are involved. These sensors communicate with each other with the help of computer networks such as Wifi, Bluetooth, 3G, 4G, and other communication standards. The data gathered from the sensors are moved to the cloud further using the APIs and gateways. The APIs and gateways follow standard communication protocols such as TCP/IP for communication between the sensors and the cloud. From the cloud, the different Big data analytical applications extract the data and build various machine learning models for further analytics. The different kinds of dashboards and APIs are used in Big data analytical applications for communicating with cloud.

Some of the Big data analytical applications include smart home, smart grid, and smart transportation. For example, in the case of smart home application, the data are gathered from various devices inside the home and stored in the cloud. The analytics application built at the top layer gathers all the data related to home from the cloud and then carries out analysis. Alerts and notifications are sent to the users

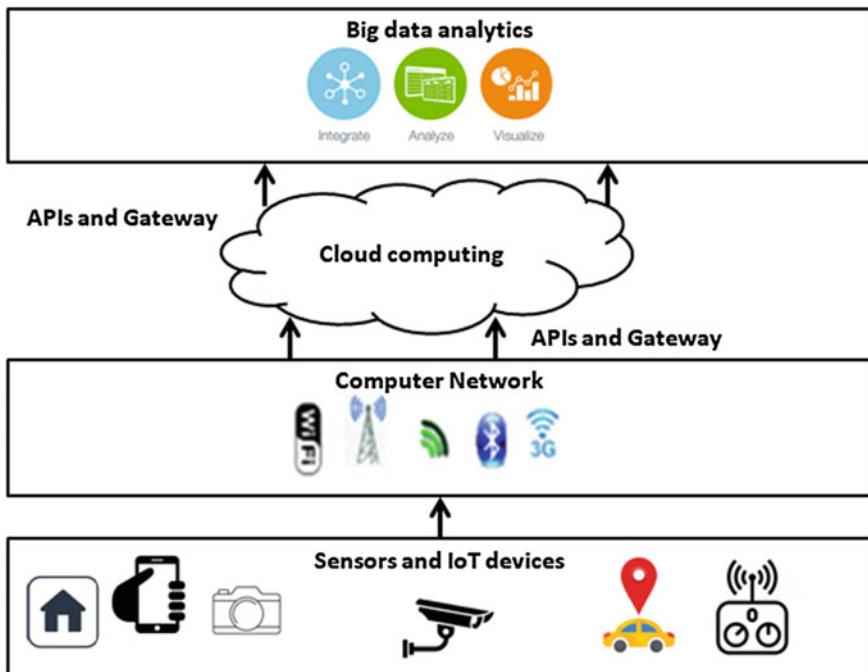


Fig. 13.1 Components of IoT

if any necessary actions that need to be taken care of like switching off lights, fans. In this way, the different components of IoT help in carrying out analytics. Typically, the components specified in this section can be narrowed down further for domain-specific analytics in IoT systems.

13.3 Analytics and IoT

In the previous sections, an overview of IoT and its components were discussed. The data that are generated by the sensor devices needs to be analyzed and cannot be ignored. Since it is estimated that 20 billion devices will be connected by 2020, the volume and variety of data to be analyzed will also become more [4]. Here, variety refers to the heterogeneity of the data coming from various sources of sensors in the IoT environment.

The main difference between IoT and analytics is the data sources for analysis. In the case of IoT, the data sources can be sensors like camera, surveillance, RFID, temperature, humidity, Co₂. The data captured from these sources are analyzed to see the impacts like air quality (Co₂ sensor), asset tracking (RFID and surveillance), smart home (electric bulb and switches sensors), and other areas. In the case Big

data analytical applications, the data sources are from social media (Twitter, Facebook, Instagram), databases (customer data, inventory data, and historical data) [5]. Even though the data sources are different in both of the areas, heterogeneity remains the same. The data formats are different and may need to be converted to a uniform format for analysis. In this section, a brief overview of different analytics that fall into the category of IoT systems is first discussed before diving into the actual case study of analytics in IoT.

13.3.1 Big Data Analytics

Big data refers to the data that are characterized with volume, variety, and velocity, generally referred to as 3 Vs. Analytics with such type of data require advanced techniques rather than the traditional way of analysis, i.e., migrating from the SQL-based analysis to more complex analysis. Advanced types of analytics are required because of different types of data that are involved in Big data, namely unstructured, semi-structured, and structured data. For example, in order to do sentimental analysis on Twitter data, access to tweets (unstructured), user information (structured), and tagging (semi-structured) are needed. The analytics needs to be carried out fast in real time. Thus, IoT interconnecting with Big data analytics helps in fast analytics, and the information can be shared among different platforms in a unified way [5, 6].

Both Big data analytics and IoT are inter-dependent on each other. As the number of the devices that get added into the IoT network, more are the chances and opportunities for Big data analytics to be carried out. An interface of relationship between IoT and Big data analytics can be depicted as shown in Fig. 13.2.

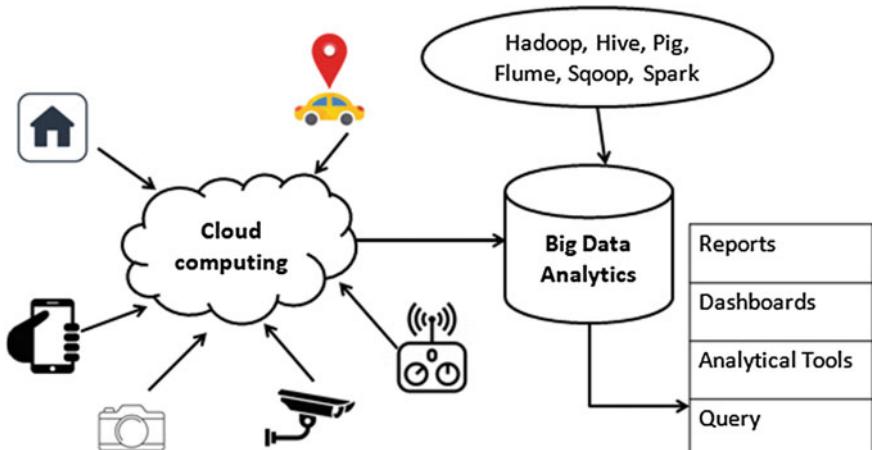


Fig. 13.2 Relationship between IoT and Big data analytics

As shown in Fig. 13.2, firstly the data are gathered from various devices and stored in the cloud. Since, the data are gathered from different sources and varies in formats, it needs to be converted into a uniform source for data analytics. The various data analytical methods that can be performed on this uniform dataset can be clustering, regression, classification etc. These methods are discussed in the earlier part of the book in Part 2.

13.3.2 *Real-Time Analytics*

Earlier attempts in streaming data analytics suggested that such type of analytics can be carried out only on high-performance computing systems. The primary techniques that are employed in streaming data analytics using distributed cloud platform are data parallelism and incremental processing. In data parallelism, the data is divided into smaller chunks, analytics is carried out on such parallel data. Incremental processing is batch analytics where the data are divided into a number of batches and analytics are carried out incrementally in a batch. IoT in streaming data analytics play an important role in bringing the devices that generate the data at faster rate to the cloud platform for analysis. For real-time analytics with IoT, incremental processing and data parallelism are less sensible as the data generated at the sources can be analyzed rapidly. Fog computing, one of the recent advanced technologies, is gaining impact in such analysis. However, with the integration of IoT with necessary storage and analytical platforms real-time streaming analytics can be formed.

13.3.3 *Bringing Analytics to IoT*

As discussed in the earlier sections of the chapter, IoT and analytics play an important role in developing smart solutions. IoT involves components like Arduino, Raspberry pie, and hardware programming. The scope of this book is limited to analytics and its types and thus specific hardware implementations; sensors information is not included in this chapter. However, a case study on the dataset collected from various sensors for analyzing air quality is presented. Using this case study as the example, the readers of the book are suggested to carry out analytics using the machine learning techniques that are discussed in Part 2 of the book.

13.4 Air Quality Analysis

Air pollution monitoring is gaining worldwide importance because of dense pollutants concentration in many areas. The sources of air pollution are due to the increase in vehicle population, fuel usage from the factories, etc. Even though many

programs are initiated for regulatory analysis of air quality by the government agencies, there is still lack of manpower, instruments, and cost viability. However, machine learning methods are helpful in analyzing such pollutants in the air for air quality monitoring.

Machine learning techniques are used for air quality monitoring and forecasting by using the earlier data of air quality. Most of the machine learning models used for air quality analysis use the historical data and come up with a precise knowledge representation based on the relationship among the data. In this section, air quality analysis using regression modeling approach is discussed. The dataset is gathered from UCI machine learning repository [7]. Since, the scope of the book is limited to data analytics, any sensor or hardware programming to gather the data is not discussed as a part of this section. The main aim is to show how machine learning can be used as an integral part of IoT.

Dataset Information

The dataset consist of 9358 instances collected from an air quality sensor device with five metal oxide sensors. The device was located in an Italian city where areas were polluted for the period of 2004–2005. The attributes that are present in the dataset are as follows:

- Date (DD/MM/YYYY);
- Time (HH.MM.SS);
- True hourly averaged concentration CO in mg/m³ (reference analyzer);
- PT08.S1 (tin oxide) hourly averaged sensor response (nominally CO targeted);
- True hourly averaged overall Non-Metanic HydroCarbons concentration in microg/m³ (reference analyzer);
- True hourly averaged benzene concentration in microg/m³ (reference analyzer);
- PT08.S2 (titania) hourly averaged sensor response (nominally NMHC targeted);
- True hourly averaged NOx concentration in ppb (reference analyzer);
- PT08.S3 (tungsten oxide) hourly averaged sensor response (nominally NOx targeted);
- True hourly averaged NO₂ concentration in microg/m³ (reference analyzer);
- PT08.S4 (tungsten oxide) hourly averaged sensor response (nominally NO₂ targeted);
- PT08.S5 (indium oxide) hourly averaged sensor response (nominally O₃ targeted);
- Temperature in °C;
- Relative humidity (%);
- AH absolute humidity.

13.4.1 Normalize Air Quality Data for Regression

The dataset considered for the regression analysis of air quality data needs to be normalized first. Normalization of the data is important because the scales in which the values of the data are present will be inconsistent. In this section, air quality data are first extracted from the UCI repository and normalized.

The data are first extracted from the repository and converted to a list. The different attributes of the data present are:

```
'CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'PT08.S2(NMHC)', 'NOx  
(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)', 'PT08.S5(O3)', 'T', 'RH', 'AH'
```

These features of the data are used for the analysis of air quality data. Initially, all the features are collected in a matrix where each entry in the matrix represents the feature value. The data are first converted to a data frame, and the minimum and the maximum attribute values are obtained. The normalization of the values is carried out using the expression $\text{value}/(\text{max}-\text{min})$.

Since the features of the data considered are in different range of values, normalization is needed. Otherwise, the plot of the values will be scattered and not clearly visible. Thus, in this case, the normalization of the values is important in this case of air quality analysis.

```
import pandas as pd
import numpy as np
import csv
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
from sklearn import preprocessing, model_selection
from sklearn.linear_model import LinearRegression

csv_file=open('AirQualityUCI_req.csv','r')
data = list(csv.DictReader(csv_file))

attr_list=['CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'PT08.S2(NMHC)',  
'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)',  
'PT08.S5(O3)', 'T', 'RH', 'AH']

matrix=np.zeros([9357,len(attr_list)])
i=0
j=0
for item in data:
    for attr in attr_list:
        matrix[i][j]=item[attr]
        j=j+1
    i=i+1
    j=0
dframe=pd.DataFrame(matrix,columns=attr_list)
n_attr=len(attr_list)
```

```
min_attr=np.zeros([n_attr])
max_attr=np.zeros([n_attr])
attr_values=np.zeros([13,9357])

for i in range(13):
    attr_values[i]=dframe[attr_list[i]]

for i in range(n_attr):
    #print(matrix[i])
    min_attr[i] = np.min(attr_values[i])
    max_attr[i] = np.max(attr_values[i])

print(min_attr)
print(max_attr)

for i in range(len(attr_values)):
    attr_values[i]=(attr_values[i]-min_attr[i])/(max_attr[i]-
min_attr[i])
    #print(attr_values)
    print(attr_values.shape)
attr_values_new=attr_values.transpose()
print(attr_values_new.shape)
#print(attr_values)
df=pd.DataFrame(attr_values_new,columns=attr_list)
axes = scatter_matrix(df, alpha=0.2, figsize=(45, 30), diagonal='histo')
corr = df.corr().as_matrix()

for i, j in zip(*plt.np.triu_indices_from(axes, k=1)):
    axes[i, j].annotate("%.3f" %corr[i,j], (0.8, 0.8),
xycoords='axes fraction', ha='center', va='center')
plt.show()
```

To see the normalized values as a plot, the correlation of the data features is calculated initially. The *corr()* method is used on the data frame for obtaining the correlation among the features of data. The correlations obtained are used to plot the dataset. A scatterplot of the normalization of the data is as shown in Fig. 13.3.

It can be observed in Fig. 13.3 where in each cell, the correlation among the features is displayed. For example, the correlation between ‘PT08.S1(CO)’ and other features is 94%, C6H6(GT) and other features is -0.04. The highest correlation values between the features are taken, and those features are considered for the regression analysis.

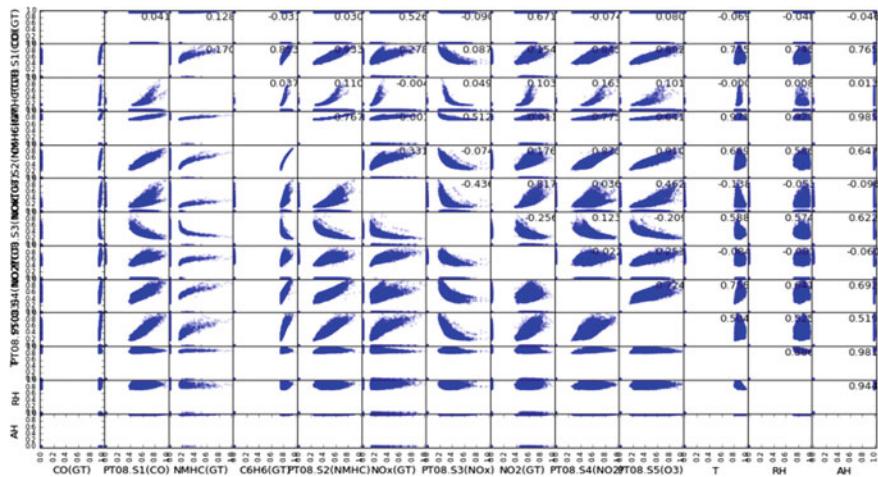


Fig. 13.3 Normalization for air quality data

13.4.2 Regression Model for Air Quality Data

In this section, the regression modeling is done for the normalized air quality data. Initially, the following modules required for the regression model are listed as follows.

- matplotlib
- numpy
- pandas
- sklearn
- linearregression

The csv file is first read and converted into a list of values with the features as listed in the dataset described in the previous section. A matrix of values is created from the attribute list in the dataset. From the scatter plot shown in Fig. 13.3, the features selected for the regression is T and $C6H6(GT)$. These features are selected for the analysis because the correlation between them is 56%. Since it is the highest correlation obtained in the scatter plot, these two data features are selected.

The regression model is first trained on the dataset by splitting the test and the training dataset. The `fit()` function is used to fit the model of the regression obtained using the `model()` function. A plot is obtained for the regression model obtained using the `matplotlib` module. The output of the plot is as shown in Fig. 13.4. It can be observed from Fig. 13.4 that the model obtained is fitting the values.

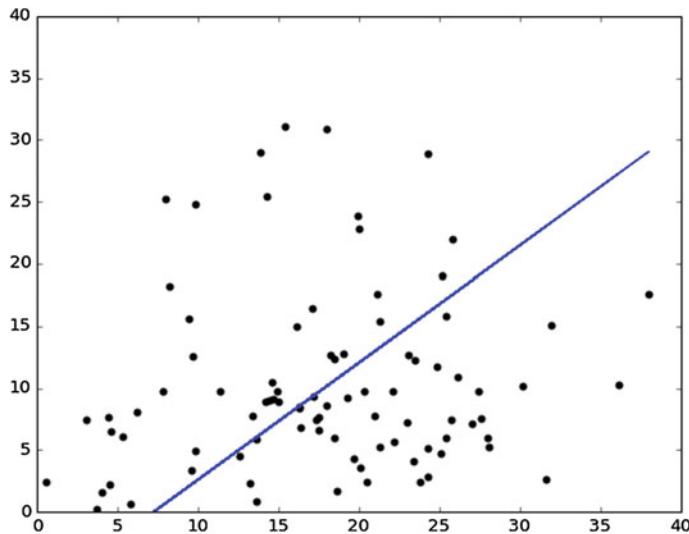


Fig. 13.4 Air quality regression model

```

import matplotlib.pyplot as plt
import numpy as np
import csv
import pandas as pd
from sklearn import preprocessing,model_selection
from sklearn.linear_model import LinearRegression

csv_file=open('AirQualityUCI_req.csv','r')
data = list(csv.DictReader(csv_file))

attr_list=['CO(GT)', 'PT08.S1(CO)', 'NMHC(GT)', 'C6H6(GT)', 'PT08
.S2(NMHC)', 'NOx(GT)', 'PT08.S3(NOx)', 'NO2(GT)', 'PT08.S4(NO2)',
'PT08.S5(O3)', 'T', 'RH', 'AH']
matrix=np.zeros([9357,len(attr_list)])
print(data[1]['CO(GT)'])

i=0
j=0
try:
    for item in data:
        for attr in attr_list:
            matrix[i][j]=float(item[attr])

            j=j+1
        i=i+1
        j=0

```

```

except Exception:
    pass

dframe=pd.DataFrame(matrix,columns=attr_list)

x=np.array(dframe[‘T’].values.reshape(9357,1))
y=np.array(dframe[‘C6H6(GT)’].values.reshape(9357,1))
x_train,x_test,y_train,y_test = model_selection.train_test_split(x,y,test_size=0.99)

clf=LinearRegression()
clf.fit(x_train,y_train)
accuracy=clf.score(x_test,y_test)
print("Accuracy: "+str(accuracy))

plt.scatter(x_train,y_train,color='black')
pred=clf.predict(x_train)
plt.plot(x_train,pred,color='blue')
plt.xlim(0,40)
plt.ylim(0, 40)
plt.show()

y_axes=np.concatenate([np.array(i) for i in y_train])
y_pred=np.concatenate([np.array(i) for i in pred])
X=[]
for i in range(len(y_axes)):
    X.append([y_axes[i],y_pred[i],y_axes[i]-y_pred[i]])
print(X)

```

In this section, air quality analysis was carried out using the regression model. The regression model was chosen for the data analytical technique because forecasting of values can be done using it. However, initially the values of the data features are normalized so that consistent values are used for regression. Normalization need not be applied for all the cases of analytics in IoT. It is carried out only when needed. In the next section, another case study on activity analysis relating to IoT is discussed.

13.5 Activity Analysis

Wearable devices in the world are increasing day by day because of IoT solutions getting smarter. Wearable devices are used for activities like running, walking, treadmill tracking, monitoring heart beat rate, blood pressure monitoring. The data generated from these devices can be used for analytics. For example, the data generated from the wearable devices can be used to analyze the number of kilometers covered during walking, running, jogging, etc.

In this section, a case study is discussed on the activity analytics. A random dataset is created for the analytics purpose here. In reality, the wearable devices use sensors like gyroscope, accelerometer for fetching the data during the activities. In this case study, the random dataset assumed consists of measurements obtained by the gyroscope and accelerometer. These measurements are created on the basis of analysis of real-time values from the devices.

Activity analysis is carried out in two parts namely:

- Plotting activities,
- Modeling activities.

13.5.1 Activity Plots

In this section, the plots of the different activities are obtained using the matplotlib module. Initially, the data are read from the csv file and converted to a data frame. From the data frame, the accelerometer values along x -axis, y -axis, and z -axis are obtained and plotted versus the time as shown in Fig. 13.5.

The measurements indicate that the graph indicates a walking pattern. The pattern of walking is obtained because of the accelerometer values obtained along all the axes. This is evident by the fact that the spacing between the peaks is about constant. If someone is walking at an irregular pace (i.e., slow-fast-slow progression), then there can be a change of frequency (more on frequency later).

A similar plot is obtained as shown in Fig. 13.6 for the gyroscope analysis. It can also be seen in Fig. 13.6 that the peaks indicate that the person is walking. Since the frequency is not changing much, it can be inferred that activity is walking and not others. In this way, activity analysis can be carried out in the context of IoT.

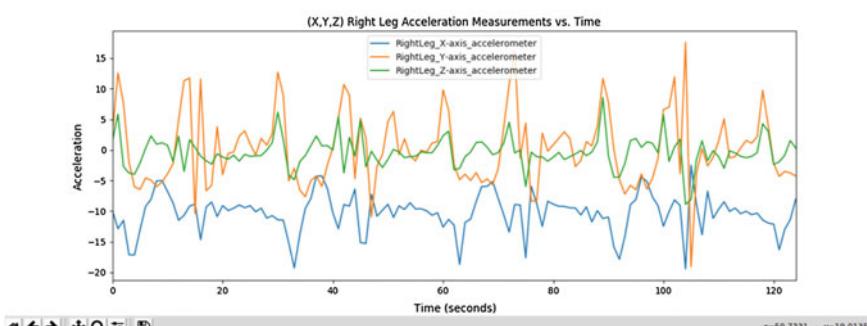


Fig. 13.5 Log acceleration in activity analysis

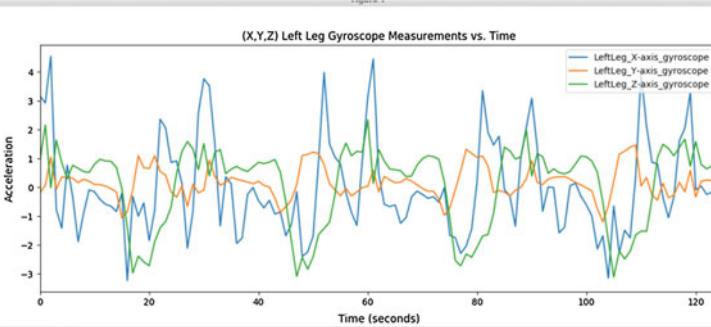


Fig. 13.6 Gyroscope measurements in activity analysis

```

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('activity-data.csv')
print(df)

ax = \
df[['RightLeg_X-axis_accelerometer", "RightLeg_Y-
axis_accelerometer", "RightLeg_Z-
axis_accelerometer"]].plot(title = "(X,Y,Z) Right Leg Accel-
eration Measurements vs. Time",
figsize=(16,5));

ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Acceleration");
plt.show()

ax = \
df[['LeftLeg_X-axis_gyroscope", "LeftLeg_Y-axis_gyroscope",
"LeftLeg_Z-axis_gyroscope"]].plot(title = "(X,Y,Z) Left Leg
Gyroscope Measurements vs. Time",
figsize=(15,5));
ax.set_xlabel("Time (seconds)")
ax.set_ylabel("Acceleration");
plt.show()

g = sns.PairGrid(df[['Torso_X-axis_accelerometer", "Torso_Y-
axis_accelerometer", "Torso_Z-axis_accelerometer']],size
=2.5, aspect=2.0)
g.map_upper(plt.scatter)
g.map_lower(sns.kdeplot, cmap="Blues_d")

```

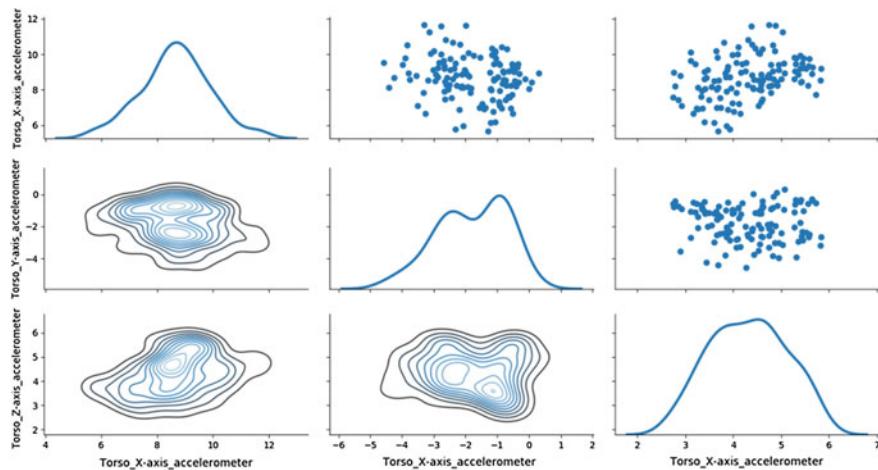


Fig. 13.7 Conditional probability estimation in activity analysis

```
g.map_diag(sns.kdeplot, lw=3, legend=False);
plt.show()
```

Figure 13.7 shows a pair of grid plot for the conditional probabilities along the X, Y, Z dimensions of the person’s acceleration. It shows the correlation with each other. It can be seen that the distributions are centered close to each other. The top triangle shows the conditional relationship between the dimensions as a scatter plot. Since the signals are approximately normal, these features can be used in feature modeling further.

13.5.2 Activity Count

In the previous section, analytics was carried out only from the perspective of different activities. But now, in this section, the number of times the activities are done is calculated. For example, if user is walking, how many steps were taken from the user is counted. To accomplish this, initially, the data frame is loaded and the different features are obtained.

The different features obtained here are leg accelerometer, gyroscope analysis, and torso acceleration. For each of these feature names, the data are split into different sections first such as *data_home* and *user_home*. These section of the data is analyzed one by one using the *split()* function. Using this function, the length of each data is computed that gives the number of times each activity has occurred.

```
import os
import csv
import re

def generate_feature_names():
    """Creates feature names for dataframe header"""
    feat_names = []
    for unit_label in ["Torso", "RightArm", "LeftArm",
"RightLeg", "LeftLeg"]:
        for sensor in ["accelerometer", "gyroscope", "magnetometer"]:
            for position in ['X', 'Y', 'Z']:
                feat_names.append(unit_label + "_" + position
+ "-axis_" + sensor)
    return feat_names

def load_segment_names(home, data):
    """Loads activity data for a specified subset"""
    return [filename for filename in os.listdir(home + data)]


feat_names = generate_feature_names()
print(feat_names)

data_home = "/.../data/"
user_data = "a09/p7/"

# load data for a single user that is walking in a parking
lot
file_names = load_segment_names(data_home, user_data)
walk_file = data_home + user_data + file_names[0]
#df = pd.read_csv(walk_file, names = feat_names)
csv_file_orig=open(walk_file,'r')
data=csv_file_orig.read()
#print(data)
cs_data=re.split('\n',data)
print(cs_data)
print(len(cs_data))
for i in range(len(cs_data)):
    try:
        cs_data[i]=float(cs_data[i])
    except:
        print(cs_data[i])
print(len(cs_data))
print(cs_data)
print(feat_names)
```

```
csv_file=open("activity-data.csv","w")
writer = csv.writer(csv_file, delimiter=',')
writer.writerow(feat_names)
print(len(feat_names))

i=0
while i <len(cs_data):
    writer.writerow(cs_data[i:i+45])
    i=i+45
```

13.6 Exercises

1. Describe the different components of IoT.
2. Create a chart of data sources for different IoT applications.
3. For the air quality analysis, experiment the regression analysis without normalization.
4. Create a chart of sensors and its bandwidth for the activity analysis case study.
5. Describe the different types of analytics that can be carried out in an IoT system.

References

1. Kopetz, H. (2011). Internet of things. In *Real-time systems* (pp. 307–323). Boston, MA.: Springer.
2. Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, 54(15), 2787–2805.
3. Kortuem, G., Kawsar, F., Sundramoorthy, V., & Fitton, D. (2010). Smart objects as building blocks for the internet of things. *IEEE Internet Computing*, 14(1), 44–51.
4. Da Xu, L., He, W., & Li, S. (2014). Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4), 2233–2243.
5. Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29 (7), 1645–1660.
6. Khan, R., Khan, S. U., Zaheer, R., & Khan, S. (2012, December). Future internet: the internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on* (pp. 257–260). IEEE.
7. Lichman, M. (2013). UCI Machine Learning Repository <http://archive.ics.uci.edu/ml>.

Chapter 14

Advanced Analytics with TensorFlow



14.1 Introduction to TensorFlow

TensorFlow is a deep learning library that is available as open source and created by Google. It is used for extensive numerical computing and machine learning tasks. It supports major languages like C++, Python, Java, Julia, R, and Go. Some of the major examples where TensorFlow is used are smart reply features available in Gmail and other Google products. For example, if there is a mail in the box suggesting *when to meet on Monday or Tuesday*, then automatically Gmail suggests for a reply like '*Lets do it on Monday*' or '*Monday works for me*' or '*Either of the days works for me*'. In this way, TensorFlow library helps in suggesting automatic replies to the mails.

The basic building block of TensorFlows is a tensor which is a multidimensional array with a set of primitives [1, 2]. The primitives in the multidimensional array are based on the number of dimensions considered in the dataset. The different variables used in the TensorFlow program holds tensors in the memory. Initially before starting the session, tensors are assigned constant values and act as variable placeholders. Once the session of TensorFlow is started, a computation graph is created that depicts a graph of nodes where each node takes the input as tensors and output as tensors. The node in the graph that does not take any input acts a constant and its output values are stored internally.

In the run() function, the graph is created and the operations specified are executed. The data may be fed into graph from outside and acts as placeholders in the graph. This facilitates the program to be run multiple times with different input. The model weights for the intermediate layers can be incremented and updated between running sessions in the TensorFlow. The main aim of using TensorFlow for computational problems is to exploit the parallel processing capabilities that are present in GPU. It helps in increasing the speed of training the dataset and reduces the overall time for execution. TensorFlow programs can be easily scaled up to more GPU processors and cloud platforms [1, 2]. The various applications of TensorFlow where it can be used are:

- Language Translation,
- Speech Recognition,
- Sentiment Analysis,
- Text Summarization,
- Object detection in images and videos,
- Chatbots and virtual assistants,
- Medical diagnosis, and
- Image recognition.

14.2 Installing TensorFlow

Installing TensorFlow Using Pip

1. To install TensorFlow via pip in Ubuntu 16.04, ensure that pip version 8.1 or above is installed on your system.
2. To verify pip installation, type the following command in the terminal:

```
pip -V
```

- 2a. In the event that pip is not installed, follow the instructions below:
- 2b. Update your system software by typing the following line in the terminal:

```
sudo apt-get update && sudo apt-get -y upgrade
```

- 2c. Install pip:

```
sudo apt-get install Python-pip
```

- 2d. Now check that your pip version is above 8.1 using step 2.
3. Install TensorFlow using the following command in the terminal:

```
pip install tensorflow
```

4. In case step 3 fails, issue the following command in the terminal:

```
sudo pip install --upgrade
```

5. Once installation is complete, validate your installation. This can be done by navigating to a directory which does not contain TensorFlow source code and running a short TensorFlow program.
6. To validate your installation, issue the following commands.
 - 6a. Enter the Python environment by typing the following in the terminal.
 - 6b. Enter the following code in the Python environment

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

- 6c. If the code executes and prints “Hello, TensorFlow!”, then the installation has been successful and has been validated.

Installing with Virtualenv

Take the following steps to install TensorFlow with Virtualenv:

1. Install pip and Virtualenv on your system by using one of the following commands in the terminal:

```
$ sudo apt-get install python-pip python-dev python-virtualenv # for Python 2.7
$ sudo apt-get install python3-pip python3-dev python-virtualenv # for Python 3.n
```

2. Create a Virtualenv environment:

```
$ virtualenv --system-site-packages targetDirectory # for Python 2.7
$ virtualenv --system-site-packages -p python3 targetDirectory # for Python 3.n
```

where targetDirectory specifies the top of the Virtualenv tree. These instructions assume that targetDirectory is `~/tensorflow`, but any directory may be chosen.

3. Activate the Virtualenv environment:

```
$ source ~/tensorflow/bin/activate # bash, sh, ksh, or zsh
$ source ~/tensorflow/bin/activate.csh # csh or tcsh
```

The source command issued should change your prompt to the following:

```
(tensorflow)$
```

4. Ensure that you have `pip ≥ 8.1` installed:

```
(tensorflow)$ easy_install -U pip
```

5. Use one of the following commands to install TensorFlow in the activated Virtualenv environment:

```
(tensorflow)$ pip install --upgrade tensorflow # for Python 2.7  
(tensorflow)$ pip3 install --upgrade tensorflow # for Python 3.n  
(tensorflow)$ pip install --upgrade tensorflow-gpu # for Python 2.7 and GPU  
(tensorflow)$ pip3 install --upgrade tensorflow-gpu # for Python 3.n and GPU
```

In case of successful installation, skip Step 6 and proceed. In case of failure, perform Step 6 before proceeding.

6. If Step 5 failed (if the native pip version is less than 8.1), install TensorFlow in the activated Virtualenv environment using the one of the following commands in the terminal:

7. (tensorflow)\$ pip install --upgrade tfBinaryURL # Python 2.7
(tensorflow)\$ pip3 install --upgrade tfBinaryURL # Python 3.n

In this case, tfBinaryURL identifies the URL of the TensorFlow Python package. The appropriate value depends on the operating system, Python version, and GPU support. Below is a list of the values for Python 3.4. For other versions of Python, it depends on the version of Python and the type of TensorFlow installation for Ubuntu operating systems.

Python 3.4
CPU only:

https://storage.googleapis.com/tensorflow/linux/cpu/tensorflow-1.5.0-cp34-cp34m-linux_x86_64.whl

GPU support:

https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-1.5.0-cp34-cp34m-linux_x86_64.whl

After installing TensorFlow, the installation can be validated as follows. To use TensorFlow installed in the via Virtualenv, the Virtualenv environment must be invoked every time. In case the environment is not active, one of the following commands may be issued.

```
$ source ~/tensorflow/bin/activate # bash, sh, ksh, or zsh  
$ source ~/tensorflow/bin/activate.csh # csh or tcsh
```

When the Virtualenv environment is active, TensorFlow will work as intended. Your prompt will become the following, indicating that the TensorFlow environment is active:

```
(tensorflow)$
```

Installing with Docker

Take the following steps to install TensorFlow through Docker:

- Install Docker on your machine as described below: For an Ubuntu Xenial 16.04 (LTS) OS, the instructions are as follows.
 - a. Uninstall old versions of docker by issuing the following commands in the terminal. This ensures that previous docker engines are uninstalled while the docker library is preserved.
sudo apt-get remove docker docker-engine docker.io
 - b. Install Docker using the repository.
 - c. Before installing docker for the first time, the docker repository must be set up. Docker can be installed and updated from this repository in the future.
- Set up the repository.
 - a. Update the apt package index using the command *sudo apt-get update*
 - c. Install packages to allow apt to use a repository over HTTPS using the following commands.

```
$ sudo apt-get install\  
apt-transport-https\  
ca-certificates\  
curl\  
software-properties-common
```

- Add Docker's official GPG key using the following command.
**\$ curl -fsSL
https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -**
- A Docker container is launched that contains one of the TensorFlow binary images using the following commands.

CPU only

```
$ docker run -it -p hostPort:containerPort TensorFlowCPUImage
```

GPU

To launch a Docker container with NVidia GPU support, the following command is used.

```
$ nvidia-docker run -it -p hostPort:containerPort TensorFlowGPUImage
```

where:

- -p hostPort:containerPort is optional. If the TensorFlow programs are to be run as Jupyter notebooks, both hostPort and containerPort to 8888.
- TensorFlowGPUImage specifies the Docker container.

14.3 TensorFlow Case Study Using MNIST Data

The MNIST data [3] consists of 60,000 training samples and 10,000 test samples on handwritten digits that are formatted as 28×28 pixel images in the dataset. A sample of the dataset is as shown in Fig. 14.1. In this section, the TensorFlow library is used to build a neural network for recognizing the handwritten digits. In this section, the code is divided into sections, where in each section the specific module is explained.

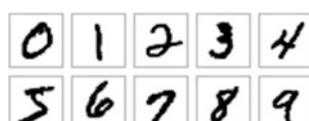
The below code first imports the TensorFlow model. The MNIST dataset is extracted from the example files. The MNIST dataset is a collection of handwritten digits ranging from 0 to 9. `one_hot = True` makes sure the labels we read are one hot encoded that means that a image with the digit ‘2’ will be represented as `[0,0,1,0,0,0,0,0,0,0]`. We can see that the addresses cannot be directly used for classification.

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist=input_data.read_data_sets("/tmp/data/", one_hot=True)
```

```
Extracting/tmp/data/train-images-idx3-ubyte.gz
Extracting/tmp/data/train-labels-idx1-ubyte.gz
Extracting/tmp/data/t10 k-images-idx3-ubyte.gz
Extracting/tmp/data/t10 k-labels-idx1-ubyte.gz
```

The basic neural network model will have three hidden layers each with 1000 nodes. The number of classes is the number of different classes we need to differentiate the numbers into, in this case its from 0 to 9 so 10 different classes. As the dataset is relatively big, due to memory restrictions, training it all at once would not be simple. Hence, we train it in batches, 100 images in each batch.

Fig. 14.1 MNIST data



```
hidden_nodes_1=1000
hidden_nodes_2=1000
hidden_nodes_3=1000
number_classes=10
batch=100
```

The placeholder variables are defined before starting the tensorflow session. X will hold our images which have been flattened down. As each image is 28x28, it gets flattened to a vector with 784 values. Y will hold the labels of the data.

```
X=tf.placeholder('float',[None,784])
Y=tf.placeholder('float')
```

Here the actual neural network is built. For each of the layers their weights and biases are defined, each layer has a set of there which are varied during the backpropogation step. Each layer is defined by DataxW + B, where W is the weights and B is the biases.

```
def network_model(data):
    layer_1 = {'weight':tf.Variable(tf.random_normal([784, hidden_nodes_1])),
               'bias':tf.Variable(tf.random_normal([hidden_nodes_1]))}

    layer_2 = {'weight':tf.Variable(tf.random_normal([hidden_nodes_1, hidden_nodes_2])),
               'bias':tf.Variable(tf.random_normal([hidden_nodes_2]))}

    layer_3 = {'weight':tf.Variable(tf.random_normal([hidden_nodes_2, hidden_nodes_3])),
               'bias':tf.Variable(tf.random_normal([hidden_nodes_3]))}

    output_layer = {'weight':tf.Variable(tf.random_normal([hidden_nodes_3, number_classes])),
                   'bias':tf.Variable(tf.random_normal([number_classes]))}

    l1 = tf.add(tf.matmul(data,layer_1['weight']), layer_1['bias'])
    l1 = tf.nn.relu(l1)
```

```

12 = tf.add(tf.matmul(l1,layer_2[‘weight’]), layer_2[‘bias’])
12 = tf.nn.relu(12)

13 = tf.add(tf.matmul(l2,layer_3[‘weight’]), layer_3[‘bias’])
13 = tf.nn.relu(13)

output = tf.matmul(l3,output_layer[‘weight’]) + output_layer[‘bias’]

return output

```

Here, the cost function is defined, which basically tells how far off the expected target we are. And we also define the optimizer; in this case, we use the Adam Optimizer to optimize the cost. We begin the TensorFlow session at this stage. We define the number of epochs as 50 but this can be changed to any number, higher the epochs higher the accuracy but it will also take a longer time to train. On each step, we print what is the current loss and the current epoch. After it has done training, we evaluate the accuracy by comparing prediction made by our model with the given testing set.

```

def train(x):
    pred=network_model(x)

    cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits
    (logits=pred, labels=Y))
    optimizer=tf.train.AdamOptimizer().minimize(cost)
    n_epochs=50
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch in range(n_epochs):
            loss=0
            for _ in range(int(mnist.train.num_examples/batch)):
                epoch_x,epoch_y=mnist.train.next_batch(batch)

    _,c=sess.run([optimizer,cost],feed_dict={X:epoch_x, Y:epoch_y})
                loss+=c
                print(‘Epoch’,epoch,’loss’,loss)
                correct = tf.equal(tf.argmax(pred, 1), tf.argmax(Y, 1))
                accuracy = tf.reduce_mean(tf.cast(correct, ‘float’))

    print(‘Accuracy:’,accuracy.eval({X:mnist.test.images,
    Y:mnist.test.labels}))

```

The function train(x) is called to see the accuracy of the neural network model built for the MNIST data.

```
train(X)
('Epoch', 0, 'loss', 4364316.3502044678)
('Accuracy:', 0.91350001)
```

14.4 Spam Classification Revisited Using TensorFlow

In this section, the spam classification problem is revisited once again in the context of TensorFlow. Spam classification case study was discussed in the text analytics section where Naïve Bayes and SVM methods were used. The same dataset is considered once again but the analysis is carried out with TensorFlow [3–5]. The case study is carried out in three stages, namely

- Preprocessing
- Creating features
- Building the classifier with TensorFlow

14.4.1 *Preprocessing*

Initially, the text in the spam dataset is preprocessed using the nltk module in Python. The features of the spam dataset are extracted one by one and converted into a data frame. The stop words are removed from the dataset using the lemmatizer module in the Python. In this way, the preprocessing of the text is done using the nltk module in Python. The preprocessing of the text is done in the same way using the methods as used in the spam classification with Naïve Bayes and SVM methods.

```
import nltk
from nltk.tokenize import word_tokenize
import numpy as np
import random
import pickle
from collections import Counter
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import tensorflow as tf
import pandas as pd
import re
```

```

df=pd.read_csv("spam.csv",encoding='latin-1')

df=df[['v1','v2']]
df.columns=['label','sms']

def create_lexicon(sent,lex):
    sent=re.sub("[^a-zA-Z]","",sent)
    sent=sent.lower()
    all_words = word_tokenize(sent)
    lex+= list(all_words)

lexicon = []
for i in df.sms:
    create_lexicon(i,lexicon)

lexicon = [lemmatizer.lemmatize(i) for i in lexicon]
w_counts = Counter(lexicon)
l2 = []
for w in w_counts:
    if 2000 > w_counts[w] > 50:
        l2.append(w)

l3=[]
for i in l2:
    if(len(i)>1):
        l3.insert(0,i)

```

14.4.2 Creating Features

Before the classifier model is built using the TensorFlow, the features are created for the classifier. A bag of words is created for this purpose. The data points that are spam or ham are passed to CountVectorizer function which split them into its corresponding matrix of token counts. This gives a representation of the frequency of all the words that occur in all the messages and thus gives a representation of what kind of words would put a particular message under a particular category or label, i.e., spam or ham.

```

def create_feature(df2,lexicon):
    featureset = []
    for l in df2['sms']:
        current_words = word_tokenize(l.lower())
        current_words = [lemmatizer.lemmatize(i)

```

```

        for i in current_words]:
            features = np.zeros(len(lexicon))
            for word in current_words:
                if word.lower() in lexicon:
                    index_value = lexicon.index(word.lower())
                    features[index_value] += 1

            features = list(features)
            featureset.append(features)

    return featureset

y_train=list(df[‘label’] [:4450])
y_train=[[1,0] if i==‘ham’ else [0,1] for i in y_train]
y_test=list(df[‘label’] [4450:])
y_test=[[1,0] if i==‘ham’ else [0,1] for i in y_test]

X_train=create_feature(df[:4450],13)
X_test=create_feature(df[4450:],13)

```

14.4.3 *TensorFlow Classifier*

A classifier model is built using the TensorFlow as follows. There are three hidden layers used for the classifier here. Each of the layers is initialized with 1500 nodes first with epochs as 100. Since the classification problem deals with spam/non-spam, the number of classes is initialised to 2. Random weights are assigned to each of the hidden layer in the neural network initially using the training dataset values as shown in the following code.

The neural network model is built using the function *nnmodel()* where the bias is multiplied with the weights in each hidden layer in the neural network. The multiplication of the bias function is achieved using the inbuilt multiplication function *matmul()* in the TensorFlow module. For each of the hidden layer, the bias is multiplied with the random weights of the neural network model.

Finally, the classification is trained on the dataset using the train function on the spam dataset. The TensorFlow model is run with the required number of epochs and the hidden layers. The confusion matrix of the classifier is obtained to validate and to see the optimized accuracy of the TensorFlow model.

```

hidden_layer_1 =2500
hidden_layer_2 = 2500
hidden_layer_3 = 2500

```

```
n_classes = 2
batch_size = 100
epochs = 500

x = tf.placeholder('float')
y = tf.placeholder('float')

hidden_1_layer = {'f':hidden_layer_1,
                  'weight':tf.Variable(tf.random_normal([len(X_train[0]),
                                                       hidden_layer_1])),
                  'bias':tf.Variable(tf.random_normal([hidden_layer_1]))}

hidden_2_layer = {'f':hidden_layer_2,
                  'weight':tf.Variable(tf.random_normal([hidden_layer_1,
                                                       hidden_layer_2])),
                  'bias':tf.Variable(tf.random_normal([hidden_layer_2]))}

hidden_3_layer = {'f':hidden_layer_3,
                  'weight':tf.Variable(tf.random_normal([hidden_layer_2,
                                                       hidden_layer_3])),
                  'bias':tf.Variable(tf.random_normal([hidden_layer_3]))}

output_layer = {'f':None,
                'weight':tf.Variable(tf.random_normal([hidden_layer_3, n_classes])),
                'bias':tf.Variable(tf.random_normal([n_classes])),}

def layers(data):
    layer_1 = tf.add(tf.matmul(data,hidden_1_layer['weight']),
                    hidden_1_layer['bias'])
    layer_1 = tf.nn.relu(layer_1)

    layer_2 = tf.add(tf.matmul(layer_1,hidden_2_layer['weight']),
                    hidden_2_layer['bias'])
    layer_2 = tf.nn.relu(layer_2)
```

```
layer_3 = tf.add(tf.matmul(layer_2,hidden_3_layer[‘weight’]),  
hidden_3_layer[‘bias’])  
layer_3 = tf.nn.relu(layer_3)  
  
output = tf.matmul(layer_3,output_layer[‘weight’]) + output_layer  
[‘bias’]  
  
return output  
  
def train_model(x):  
    pred = layers(x)  
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits  
(logits=pred,labels=y))  
    optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize  
(cost)  
  
with tf.Session() as sess:  
    sess.run(tf.global_variables_initializer())  
    for epoch in range(epochs):  
        epoch_loss = 0  
        i=0  
        while i < len(X_train):  
            start = i  
            end = i+batch_size  
            batch_x = np.array(X_train[start:end])  
            batch_y = np.array(y_train[start:end])  
  
            , k = sess.run([optimizer, cost],  
feed_dict={x: batch_x,y: batch_y})  
            epoch_loss += k  
            i+=batch_size  
  
        print(‘Epoch’, epoch+1, ‘completed out of’, epochs, ‘loss:’,  
epoch_loss)  
        correct = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))  
        accuracy = tf.reduce_mean(tf.cast(correct, ‘float’))  
        print(‘Accuracy:’,accuracy.eval({x:X_test, y:y_test}))  
return pred  
  
p=train_model(x)
```

```

pred=[]
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    output= sess.run(tf.argmax(p,1),feed_dict={x:X_test})

y_pred=pd.Series(output,name='Predicted')
y_tst=[1 if i==[1,0] else 0 for i in y_test]
y_actual=pd.Series(y_tst,name='Actual')

df_confusion=pd.crosstab(y_actual,y_pred)
print df_confusion

```

// Confusion matrix for spam classification with tensorflow.

The confusion matrix:-

```
[ [1384  183]
```

```
[  38   207]]
```

In this section, the spam classification problem was revisited with TensorFlow. The model now used a neural network model with three hidden layers and epochs as 100. The confusion matrix obtained from the TensorFlow is as shown above. It can be observed that the TensorFlow model has classified more number of data points as spam and ham correctly compared to the Naïve Bayes and SVM classification considered in the earlier sections of the book. In this way, the TensorFlow model can be used for improving the accuracy of the classifier. In the next section, the question classification with TensorFlow is discussed.

14.5 Question Classification Revisited with TensorFlow

In the previous section, the optimized accuracy was obtained using the TensorFlow model for the spam classification. The question classification considered in the text analytics is revisited once again with the TensorFlow model [3]. Now, the classification is carried out in three steps, namely

- Preprocessing
- Creating features
- Classification

14.5.1 Preprocessing

Initially, the text in the question classification dataset is preprocessed using the nltk module in Python. The features of the question classification dataset are extracted one by one and converted into a data frame. The stop words are removed from the dataset using the lemmatizer module in the Python. In this way, the preprocessing of the text is done using the nltk module in Python. The preprocessing of the text is done in the same way using the methods as used in the question classification with SVM method.

```
import nltk
from nltk.tokenize import word_tokenize
import numpy as np
import random
import pickle
from collections import Counter
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
import tensorflow as tf
import pandas as pd
import re
import csv

f=open('train_questions.txt','rU') #Reads in the training questions text file
with open('train_labels.csv','rb') as k:
    reader=csv.reader(k)
    train_labels=list(reader)

df=pd.read_table('train_questions.txt',sep=',',names=('ID','Ques'))
df=df.iloc[1:]

df_label=pd.DataFrame(train_labels)
df_label=df_label .iloc[1:]
df_label.columns=['ID','Label']

df=pd.merge(df,df_label, on='ID')

def create_lexicon(sent,lex):
    sent=re.sub("[^a-zA-Z]","",sent)
    sent=sent.lower()
    all_words = word_tokenize(sent)
    lex+= list(all_words)
```

```

lexicon = []
for i in df.Ques:
    create_lexicon(i,lexicon)

lexicon = [lemmatizer.lemmatize(i) for i in lexicon]
w_counts = Counter(lexicon)
l2 = []
for w in w_counts:
    if 2000 > w_counts[w] > 50:
        l2.append(w)
l3=[]
for i in l2:
    if(len(i)>1):
        l3.insert(0,i)

```

14.5.2 Creating Features

Before the classifier model is built using the TensorFlow, the features are created for the classifier. The data points that belong to one of the categories, i.e., from 1 to 5, are passed to CountVectorizer function which split them into its corresponding matrix of token counts. This gives a representation of the frequency of all the words that occur in all the messages and thus gives a representation of what kind of words would put a particular message under a particular category.

```

def create_feature(df2,lexicon):
    featureset = []
    for l in df2[“Ques”]:
        current_words = word_tokenize(l.lower())
        current_words = [lemmatizer.lemmatize(i) for i in current_words]
        features = np.zeros(len(lexicon))
        for word in current_words:
            if word.lower() in lexicon:
                index_value = lexicon.index(word.lower())
                features[index_value] += 1

        features = list(features)
        featureset.append(features)

    return featureset
X_train=create_feature(df[:2500],l3)
X_test=create_feature(df[2500:],l3)

```

```

y_train=list(df[‘Label’][:2500])
for i in range(len(y_train)):
    l=[0]*6
    l[int(y_train[i])]=1
    y_train[i]=l

y_test=list(df[‘Label’][2500:])
for i in range(len(y_test)):
    l=[0]*6
    l[int(y_test[i])]=1
    y_test[i]=l

```

14.5.3 *TensorFlow Classifier*

A classifier model is built using the TensorFlow as follows. There are three hidden layers used for the classifier here. Each of the layers is initialized with 1500 nodes first with epochs as 100. Since the classification problem deals with different categories, the number of classes is initialised to 6. Random weights are assigned to each of the hidden layer in the neural network initially using the training dataset values as shown in the following code.

The neural network model is built using the function *nnmodel()* where the bias is multiplied with the weights in each hidden layer in the neural network. The multiplication of the bias function is achieved using the inbuilt multiplication function *matmul()* in the TensorFlow module. For each of the hidden layer, the bias is multiplied with the random weights of the neural network model.

Finally, the classification is trained on the dataset using the train function on the question classification dataset. The TensorFlow model is run with the required number of epochs and the hidden layers. The confusion matrix of the classifier is obtained to validate and to see the optimized accuracy of the TensorFlow model.

```

hidden_layer_1 =2500
hidden_layer_2 = 2500
hidden_layer_3 = 2500

n_classes = 2
batch_size = 100
epochs = 500

x = tf.placeholder(‘float’)
y = tf.placeholder(‘float’)

```

```
hidden_1_layer = {'f':hidden_layer_1,
                  'weight':tf.Variable(tf.random_normal([len(X_train[0]),
                                                      hidden_layer_1])),
                  'bias':tf.Variable(tf.random_normal([hidden_layer_1]))}

hidden_2_layer = {'f':hidden_layer_2,
                  'weight':tf.Variable(tf.random_normal([hidden_layer_1,
                                                      hidden_layer_2])),
                  'bias':tf.Variable(tf.random_normal([hidden_layer_2]))}

hidden_3_layer = {'f':hidden_layer_3,
                  'weight':tf.Variable(tf.random_normal([hidden_layer_2,
                                                      hidden_layer_3])),
                  'bias':tf.Variable(tf.random_normal([hidden_layer_3]))}

output_layer = {'f':None,
                'weight':tf.Variable(tf.random_normal([hidden_layer_3, n_classes])),
                'bias':tf.Variable(tf.random_normal([n_classes])),}

def layers(data):
    layer_1 = tf.add(tf.matmul(data,hidden_1_layer['weight']),
                    hidden_1_layer['bias'])
    layer_1 = tf.nn.relu(layer_1)

    layer_2 = tf.add(tf.matmul(layer_1,hidden_2_layer['weight']),
                    hidden_2_layer['bias'])
    layer_2 = tf.nn.relu(layer_2)

    layer_3 = tf.add(tf.matmul(layer_2,hidden_3_layer['weight']),
                    hidden_3_layer['bias'])
    layer_3 = tf.nn.relu(layer_3)

    output = tf.matmul(layer_3,output_layer['weight']) + output_layer['bias']
```

```
    return output

def train_model(x):
    pred = layers(x)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
        logits=pred, labels=y))
    optimizer = tf.train.AdamOptimizer(learning_rate=0.001).minimize(
        cost)

    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        for epoch in range(epochs):
            epoch_loss = 0
            i=0
            while i < len(X_train):
                start = i
                end = i+batch_size
                batch_x = np.array(X_train[start:end])
                batch_y = np.array(y_train[start:end])

                _, k = sess.run([optimizer, cost], feed_dict={x: batch_x,
                    y: batch_y})
                epoch_loss += k
                i+=batch_size

            print('Epoch', epoch+1, 'completed out of', epochs, 'loss:', epoch_loss)
            correct = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
            accuracy = tf.reduce_mean(tf.cast(correct, 'float'))
            print('Accuracy:',accuracy.eval({x:X_test, y:y_test}))
    return pred
```

14.6 Exercises

1. Would the accuracy be affected if the type of optimizer used is changed from AdamOptimizer to something else?
2. Why is batch size important for training on machines with limited RAM and vRAM? What would happen if batch size is not defined?
3. Why is each one of the images encoded into a vector? And how can this value be used to make sure that the neural network can be trained for a completely different dataset of images?

4. Why do we remove very high-frequency and very low-frequency words when creating the feature set for question classification? What would happen if we did not remove them?
5. Implement TensorFlow model for Naïve Bayes classification on the spam classification case study considered in this part.

References

1. Martín, A., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., & Corrado, G. S. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. URL: <http://tensorflow.org/>. Software available from: <http://tensorflow.org/>.
2. Glauner, P. (2016). *Deep learning on big data sets in the cloud with apache spark and google tensorflow*.
3. MNIST data: <https://archive.ics.uci.edu/ml/databases/mnist/>.
4. Mehta, P., Dorkenwald, S., Zhao, D., Kaftan, T., Cheung, A., Balazinska, M., ... & AlSayyad, Y. (2017). Comparative evaluation of big-data systems on scientific image analytics workloads. *Proceedings of the VLDB Endowment*, 10(11), 1226–1237.
5. Berral-García, J. L. (2016, July). A quick view on current techniques and machine learning algorithms for big data analytics. In *2016 18th international conference on transparent optical networks (ICTON)* (pp. 1–4). IEEE.

Chapter 15

Recommendation Systems



15.1 Introduction

Along the advent like computing and Internet, large volume of information is being formed in different areas like science, automation, manufacturing, authority, and other key fields. The information produced can be categorized into three main distinct categories; structured, semi-structured, and also unstructured data. It is a challenge for the application developers to administer and gain intuitions with these different sources of information.

Recommendation systems have compelling importance in areas such as e-commerce and retail. In the late years, trade applications have evolved from traditional to Web based and also successful. A few of the examples include Amazon, e bay, Flipkart, etc. These Web-based applications employ recommender systems to naturally suggest a few of the related commodity to augment the personalized spending experience for the buyers [1]. The main function of recommender systems in distant business applications is to identify the likelihood of a user buying a product based on the suggestions. Therefore, recommendation systems play an important role in wide retail areas that provides variety of decisions for the users to buy products. The different areas where recommendation systems can be used are associated to fashion, cooking, gaming, rap, movies, etc.

The underlying principle of recommendation systems is to identify the dependencies that exist between the user and the products. For example, consider a Netflix platform where different users watch different kinds of movies. In this case, a user who watches an action movie is likely to watch another action movie and not a historical movie. Like this in many scenarios, different correlations among the items are visible and can be used for recommendations. However, in this case the recommendation is based on the category of the movie, and it can also be on the finer level of granularity of each individual item in the set of items. This fine level of granularity of recommendations can be determined with the help of ratings matrix for each item and the customers. So, for those who have not purchased the

items, the prediction of the ratings can be done depending on the amount of information. The prediction accuracy will be better if the information is larger about the ratings and groups of users who belong to a particular category can be easily determined.

In order to build a recommender system, there are two primary models used namely prediction model and ranking model. In the prediction model of recommendation systems, a typical ranking matrix of users and items is used for recommendations as shown in Table 15.1. In this Table 15.1, a set of five users and five products rating matrix is shown on a scale of 1–5, where 1 being the lowest and 5 being the highest. The main goal of the recommender system is to predict the ratings that are not rated and denoted as by ‘?’’. There are various methods and models that can be used for building prediction models for recommendation systems. These models are discussed later in the chapter.

In the case of ranking model of recommendation system, the main goal is to determine the top-k items for a particular user or for all the users. Generally, the top-k items are determined first and then use it for recommendations. A small example is as shown in Table 15.2 for the ranking version of the recommendation system. Here, the ratings of the user for an item are not taken into account rather the products purchased and ranking of the items are done for the recommendation system.

There are other models of recommendation systems such as item–item based and user–item based that fall into the category of both prediction and ranking version models of the recommendation systems [2, 3]. In this chapter, firstly the different types of recommendation systems and models of recommendation systems are discussed and then the case study is presented. The recommendation system case study presented in this chapter is based on the prediction version of the recommendation system.

Table 15.1 User–item prediction matrix

	p1	p2	p3	p4	p5
u1	?	3	2	4	5
u2	2	5	?	?	5
u3	3	?	1	3	2
u4	?	3	?	2	3
u5	5	?	?	1	2

Table 15.2 User–item ranking matrix

Users	Items purchased
u1	p1, p2
u2	p3, p4, p5
u3	p1, p2, p2
u4	p3
u5	p4, p5

15.2 Types of Recommendation Systems

In the previous section, the two primary models for recommendation systems namely ranking version and prediction version were discussed. Though these are the only primary models followed, there are different categories or types of recommendation systems such as knowledge-based recommender system, content-based recommender systems, and collaborative filtering recommendation systems. This section highlights these important types of recommendation systems.

15.2.1 Content-Based Recommender Systems

The descriptive information about the products is used in content-based recommender systems [1]. The term ‘content’ refers to the descriptive information available for a product. This descriptive information such as a review on a product can be used as an entity for the recommendation. For example, if a user in a movie platform such as netflix, gives the movie “terminator” the highest rating. However, the access to the rating matrix is restricted. In such cases, the keyword/title of the movie can be used for recommendation to the user based on the genre, i.e., action in this case.

The descriptive information present along with ratings is used for training and builds a recommendation system for the users. The training dataset for each user is the descriptive information available such as review, feedback mail. It is essentially a classification problem where the recommendation is ‘Yes’ or ‘No’ in simple words. Thus, the class-specific model is finally used to predict whether the user is like a product/item for which the rating is unknown. However, this method of providing recommendations based on the descriptive information has the following disadvantages:

- The descriptive information of a user is not known if it is the first time the user is using the system. So, the history of the user information is not known in prior to provide the recommendations. In fact, large amount of information is needed for training and make good predictions.
- One more disadvantage of content-based method is that the diverse recommendation of the products is not possible. This scenario is encountered when a user has never bought a product and recommendations on such products cannot be given. It is obvious because only certain keywords of the description are used for recommendation and not the entire description.

15.2.2 Knowledge Based Recommendation Systems

In certain situations, recommendations may not be made based on the ratings or the descriptive information of items. The examples include automobiles, real estate, tourism, financial services. In these sectors, rather than depending on the ratings,

the actual knowledge is used for recommendations and the nature of the preferences of the customer evolves over a time. So, it is difficult to understand the preferences of the customer and their preferences also evolve over a time.

15.2.3 *Collaborative Filtering Systems*

In collaborative filtering approach, the ratings of users for different items are used for recommendations. But, it should be noted that ratings for all the items are not available. The ratings of the items by different users are available for only a small fraction of the dataset; i.e., users would have liked only few of the items and not all. So, the ratings matrix of the users and items will be sparse; i.e., some of the values will be not known. The correlation between the ratings matrix of the items and users is used for the prediction of the ratings in collaborative filtering systems [4]. The two essential types of collaborative filtering based on the correlation are user based and item based.

- **User-based collaborative filtering:** A group of users is identified in this technique for recommendations. For example, consider two users A and B who have given similar ratings to the items I1 and I2. But, A has not given any rating to another item I3, but the user B has given rating for the same. In such cases, the rating of the user B can be used for user A. In general, k-users are identified first for a target user and their ratings are used for making prediction ratings.
- **Item-based collaborative filtering:** Here, a group of items are identified first for recommendations. For example, consider a user ‘A’ and two movies M1 and M2. The user ‘A’ has given the rating for both M1 and M2. If another movie M3 needed to be recommended to the user, the movie M3 must be similar or belong to the same category of M1 and M2. In general, the set ‘S’ that represents the likely item set are identified first and the new item recommendation depends on the belongingness of the set ‘S.’

In this chapter, collaborative filtering method is used for recommendations. For the collaborative filtering, there are many methods such as matrix factorization, neighborhood method, Bayesian methods, clustering. In the next section, the recommendation model based on matrix factorization is presented with example and a case study.

15.3 Collaborative Filtering Using Matrix Factorization

As discussed in the previous sections, the main aim of the recommender systems is to help users in selecting the items of their choice. The prediction of user ratings in the collaborative filtering technique comprises of history of purchases and transactions. For the matrix factorization technique [5, 6], a user-item rating matrix

$R: U \times I$, where U represents the users and I represents the items. Each value in R , i.e., R_{ui} represents the rating of a particular item by the user.

15.3.1 Matrix Factorization Example

An example of three users and three products in the category of electronics is considered in this section for matrix factorization. In Fig. 15.1, the initial rating matrix is represented by R , and the latent feature matrix for the users is represented by U ; the latent feature matrix for products is represented by V . There are three users u_1, u_2, u_3 and three categories of electronics e_1, e_2, e_3 . User u_1 has given ratings for the electronics e_2, e_3 as 4, 2, respectively. Similarly, user u_2 has given ratings for the electronics e_1, e_2 as 3, 2, respectively. User u_3 has given ratings for the electronics e_1, e_3 as 5, 3, respectively. The rating 0 indicates that the user has not given any rating to the product. The main aim is to predict the rating of this product and check whether the user has a likely interest in the product or not.

The latent feature matrices U and V are used for matrix factorization. The approximated matrix $\hat{R} \approx R$ is obtained by the product of U and V as shown in Fig. 15.1. In Fig. 15.2a, it can be observed that an approximate value of $3.8 \approx 4$ rating for the electronics e_2 from the user u_1 is obtained from the latent feature vectors. Similarly, for user 3, rating of the electronics e_3 is $2.7 \approx 3$. With this, the rating of the electronics e_1 from user u_1 can be 4.4; the rating of the electronics e_3 from the user u_2 is 1.8, and the rating of the electronics e_2 from the user u_3 is 4.4.

With the help of the predicted matrix, the ratings of the electronic items that are not rated from the users can be known and used for recommendations. From Fig. 15.2b, it can be seen that user u_1 predicted rating is 4.4 (high) for electronics e_1 and 1.4 (low) rating for electronics e_3 from the user u_2 . So, here the electronics e_1 can be recommended to u_1 , whereas the electronics e_3 cannot be recommended to user u_3 since the rating is low.

In this way, the matrix factorization method can be used build the prediction matrix \hat{R} from the latent feature matrices U and V . These latent features U and V need to be obtained by identifying the different genre of electronics considered. One of the techniques used is gradient descent analysis method for determining these latent features. It is beyond the scope of the chapter to explain in detail about identifying the gradient descent method for building the latent features.

	e_1	e_2	e_3		
u_1	0	4	2	u_1	1.98
u_2	3	2	0	u_2	1.21
u_3	5	0	3	u_3	2.30
$R_{3 \times 3}$		$U_{3 \times 1}$		$V_{1 \times 3}$	

Fig. 15.1 Product recommendation using matrix factorization

Fig. 15.2 **a** Approximated rating matrix using U and V.
b Predicted rating matrix using U and V

	e1 (2.24)	e2 (1.92)	e3 (1.18)
u1 (1.98)	?	3.8	2.3
u2 (1.21)	2.7	2.3	?
u3 (2.30)	5.2	?	2.7

	e1 (2.24)	e2 (1.92)	e3 (1.18)
u1 (1.98)	4.4	3.8	2.3
u2 (1.21)	2.7	2.3	1.4
u3 (2.30)	5.2	4.4	2.7

15.3.2 Matrix Factorization in Python

In this section, a small example on the matrix factorization in Python is discussed. For the matrix factorization, the actual rating matrix R and the latent feature matrices P and Q are needed. The alpha and beta parameters are used with default values. These values are used to minimize the error rate in the prediction. Initially, the random values of P and Q are used and iterated over a specified number of steps along with alpha and beta values.

For each row of the rating matrix, the value in the rating matrix is subtracted with latent features P and Q to obtain the error rate e_{ij} . The error rate is optimized and by again running over the iteration with P and Q values, but now with alpha and beta parameters. Finally, for each rating in the matrix R, error rate is optimized by increasing the power of the difference of actual matrix and the dot product of latent features P and Q. The optimized error rate obtained is now divided using alpha and beta parameters to determine the threshold of the error of prediction. For the example in this section, the threshold considered is less than 0.001, and if the condition is satisfied, the optimized latent features P and Q are returned to form the prediction matrix.

```
def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    Q = Q.T
    for step in range(steps):
        #print("Step", step)
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i, :], Q[:, j])
                    for k in range(K):
```

```

P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
eR = numpy.dot(P,Q)
e = 0
for i in range(len(R)):
    for j in range(len(R[i])):
        if R[i][j] > 0:
            e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
            for k in range(K):
                e = e + (beta/2) * (pow(P[i][k],2) + pow(Q[k][j],2))
if e < 0.001:
    break
return P, Q.T

```

The following snippet of code initializes the rating matrix R for the matrix factorization. A sample of 10 users and their ratings are used to carry out the matrix factorization. The latent features P and Q are initialized with random values at the beginning and then passed to the matrix_factorization() function. The optimized latent features P and Q obtained from the matrix factorization method are then used to prediction rating matrix by the dot product of P and Q. Here, nR represents the predicted rating matrix using matrix factorization method.

```

R = [
    [ 5,  3,  0,  0,  0,  0,  0,  0,  0,  3 ],
    [ 3,  3,  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 ],
    [ 4,  0,  0,  0,  3,  4,  0,  0,  0,  0 ],
    [ 4,  3,  0,  0,  0,  4,  0,  0,  0,  4 ],
    [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0 ],
    [ 3,  2.5, 0, 0,  0,  3,  0,  0,  0,  2.5 ]
]

```

```

N = len(R)
M = len(R[0])
K = 2

P = numpy.random.rand(N,K)
Q = numpy.random.rand(M,K)

```

```
nP, nQ = matrix_factorization(R, P, Q, K)
```

```
nR = numpy.dot(nP, nQ.T)
```

As discussed in this section, the matrix factorization method can be used to obtain the prediction rating matrix using the latent features P and Q. The matrix factorization method uses collaboratively the user features (P) as well as the item features (Q) to obtain the prediction rating matrix and hence the name collaborative filtering [7]. In the next section, matrix factorization method is used for recommendation on a movie dataset.

15.4 Building Recommendation Systems Using Movielens 100k Dataset

In this section, a set of movies and users are used to demonstrate recommendation systems using matrix factorization method on 100k movie lens dataset [8]. The dataset is used for multiple research works on recommendations. The dataset essentially consists of features namely userId, movieId, rating, and Timestamp. These features need to be carefully examined to build the recommendations. In the upcoming sections, a step-by-step analysis is done for building the prediction matrix and to provide recommendations.

15.4.1 Extracting the Features of Movielens 100k Dataset

In this section, first the features of the movielens dataset are extracted and a rating matrix is prepared. Initially, the dataset needs to be downloaded from [8] in csv format and put it into the working project folder. The file downloaded needs to be parsed for extracting the features of the dataset. The interested features are movieId, userId, and the rating. So, initially the parameters passed to the function are the filename, header information set to boolean value “true”, and a variable “quite”. It is used to identify the stages for extracting the features.

A set of movies and the rating matrix R is first initialised to empty sets. The file is read using the readline() method with delimiter as ‘,’ since it is a csv file. For each line, three values are read, i.e., u for userId, m for movie id, and r for rating by the user. The rating is treated as a floating point unit while reading the file. Once the rating r is read from the file, R[u][m] the actual rating matrix is updated by verifying whether the user is initially there in the previous iteration of reading.

```
def parse(filename="ratings.csv", has_header=True, quiet=False):
    """
    Input file name.
    Required format:
    userId,movieId,rating, ...
    int, int, float
```

```

Set has_header to False if the file does not have a header.
Set quiet to False to print stages.
Returns a matrix with rows representing users and columns representing
movies and mapping of movieId: column index.
Discards all movies that do not have any rating.
"""

R = {}
movies = set()
if not quiet:
    print("Reading...")
with open(filename) as f:
    if has_header:
        if not quiet:
            print("Skipping header...")
        f.readline()
    for line in f:
        u, m, r = (lambda x, y, z: (int(x), int(y), float(z)))(*line.split(',')[:3])
        if u not in R:
            R[u] = {}
            R[u][m] = r
            movies.add(m)
    if not quiet:
        print("Read %d users and %d movies" % (len(R), len(movies)))
movies = {i: j for i, j in zip(sorted(movies), range(len(movies)))}
if not quiet:
    print("Generating matrix...")
rat = [[0.0] * len(movies) for _ in range(len(R))]
for u in R:
    for m in R[u]:
        rat[u - 1][movies[m]] = R[u][m]
R = rat
if not quiet:
    print("Done...")
    print(sum(map(len, R)), "values in matrix in", len(R), "rows")
return R

```

In this way, the parse() function is used for extracting the features of movielens dataset and generates the rating matrix. The function parse() is called to from the main function. A sample of ratings by 10 users can be seen with the help of the following code.

```

if __name__ == "__main__":
    r=parse()

```

```
#print("Movies:", m[:10])
for i in r[:10]:
    print(*i[:10])

5.0  3.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  3.0
3.0  3.0  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
4.0  0.0  0.0  0.0  3.0  4.0  0.0  0.0  0.0  0.0  0.0
4.0  3.0  0.0  0.0  0.0  4.0  0.0  0.0  0.0  0.0  4.0
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
3.0  2.5  0.0  0.0  0.0  3.0  0.0  0.0  0.0  0.0  2.5
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
```

15.4.2 Training the Model for Recommendation

Once the features are extracted from the movielens dataset, then the model needs to build for recommendation. In this section, the method of building the collaborative filtering model with matrix factorization as the method is discussed. Initially, the parse module used earlier for extracting the features is imported. The rating matrix returned by the parse module is collected in ‘R’. For the matrix factorization, U and V matrices (latent features) are needed. This is initialised randomly first with P and Q as the variables. Then, the `matrix_factorization()` method is used with the parameters P, Q, and R and default values for alpha, beta, and steps as shown in the code. These values are needed for obtaining the latent features using gradient descent functions.

```
from extract_movie_rating import parse

R = numpy.array(parse())
R=R[:100, :10]
N = len(R)
M = len(R[0])
K = 2

P = numpy.random.rand(N, K)
Q = numpy.random.rand(M, K)

nP, nQ = matrix_factorization(R, P, Q, K)
```

```

nR = numpy.dot(nP, nQ.T)

print("Done")
print(R, file=open("actual_rating.txt", "w"))
print(nR, file=open("predicted_rating.txt", "w"))

```

Once the latent features are obtained and collected in nP and nQ variables, a dot product of the two matrices will give the predicted ratings nR. Since the dataset is huge, it is difficult to print the entire predicted matrix nR. Thus, the predicted ratings are transferred to a file predicted_rating.txt and can be compared with the actual_rating matrix for providing recommendations. In the next section, the matrix factorization method used for building the latent features is discussed.

15.4.3 Matrix Factorization for Building Recommendation on MovieLens Dataset

The essential method for collaborative filtering is matrix factorization. In this section, the following code demonstrates the matrix factorization method. The main aim of the method is to generate eigenvalues for the latent features P and Q that were initialized randomly. So, three for loops are used for generating the latent features P and Q. The first loop is used for iterating over the length of the actual rating matrix R, i.e., all the rows and columns of the rating matrix R. The second for loop is used for iterating each user in the rating matrix and rating for all the movies. Finally, the latent features P and Q are obtained by multiplying with alpha, beta, and e_{ij} values. Here, e_{ij} values represent the error estimation values for each of the latent feature. These values are updated for each iteration unless and until the latent features are tuned according to the equation shown below.

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = \left(r_{ij} - \sum_{k=1}^K p_{ik} q_{kj} \right)^2$$

```

import numpy
def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
    Q = Q.T
    for step in range(steps):
        #print("Step", step)
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - numpy.dot(P[i, :], Q[:, j])
                    for k in range(K):

```

```

P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
eR = numpy.dot(P,Q)
e = 0
for i in range(len(R)):
    for j in range(len(R[i])):
        if R[i][j] > 0:
            e = e + pow(R[i][j] - numpy.dot(P[i,:],Q[:,j]), 2)
            for k in range(K):
                e = e + (beta/2) * (pow(P[i][k], 2) + pow(Q[k][j], 2))
if e < 0.001:
    break
return P, Q.T

```

15.4.4 Provide Recommendations

The movie lens dataset consists of 100,000 ratings (1–5) from 943 users on 1682 movies. Each user has rated at least 20 movies. To predict the ratings that are not unknown, matrix factorization method is used. Since the dataset is large, the results of the predicted matrix need to be appended to a file for analysis. For this purpose, a matrix of eight users and ten movies for actual and predicted ratings is shown in Fig. 15.3a, b, respectively. There is a close approximation of values between the actual and predicted values.

15.4.5 Analytics on the Results of 100k MovieLens Data

The predictive rating matrix obtained can be used for analytics and see what can be ratings of movies by the users which they have not watched. From Fig. 15.4a, it can be observed that users u5 and u6 have some similarity in watching movies. Figure 15.4a depicts the ratings of the movies m1–m6 from the user u5 and u6. It can be observed that for the movie m6, the rating is 4 by both of the users. For the same movie m6, the predicted rating of user u5 is 3.92 and user u6 is 4.07. Similarly for the movie m1, the predicted rating of user u5 is 4.05 and user u6 is 4.15. Hence, the predicted ratings are approximately equal to the actual ratings.

It can be seen from Fig. 15.4a that there is no rating for the movie m2 from the user u5. But, the actual rating of the movie m3 from the user u6 is 3 and the predicted rating is 3.20. The predicted rating for the movie m2 from the user u5 is 2.96 as shown in Fig. 15.4b. Since the predicted ratings of the movie m2 for the users u5 and u6 are close, the movie m2 can be recommended to the user u5. In this way, the prediction matrices can be used for recommendations.

		Movies										
		Users	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10
(a)			[[5	3	0	0	0	0	0	0	3]
			[3	3	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]
			[4	0	0	0	3	4	0	0	0]
			[4	3	0	0	0	4	0	0	4]
			[0	0	0	0	0	0	0	0	0]
			[3	2.5	0	0	0	3	0	0	2.5]
(b)			Movies									
			[[4.72 2.92 1.42 1.83 3.21 4.39 1.46 2.04 0.76 3.37]									
			[3.08 2.83 0.85 0.69 2.55 3.17 0.72 1.09 0.79 2.96]									
			[0.23 0.25 0.06 0.02 0.21 0.25 0.04 0.07 0.07 0.25]									
			[1.03 0.89 0.29 0.26 0.82 1.04 0.25 0.37 0.24 0.94]									
			[4.05 2.96 1.18 1.32 2.98 3.92 1.14 1.63 0.80 3.26]									
			[4.15 3.20 1.20 1.26 3.14 4.07 1.13 1.62 0.87 3.48]									
			[0.15 0.12 0.04 0.04 0.11 0.15 0.04 0.06 0.03 0.13]									
			[2.93 2.47 0.83 0.78 2.32 2.94 0.75 1.09 0.68 2.63]]									

Fig. 15.3 **a** Actual rating matrix for 100k movie lens dataset analysis. **b** Predicted rating matrix for 100k movie lens dataset analysis

		m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	
(a)	u5	[4	0	0	0	3	4	0	0	0	0]
	u6	[4	3	0	0	0	4	0	0	0	4]
		m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	
(b)	u5	[4.05	2.96	1.18	1.32	2.98	3.92	1.14	1.63	0.80	3.26]
	u6	[4.15	3.20	1.20	1.26	3.14	4.07	1.13	1.62	0.87	3.48]

Fig. 15.4 **a** Actual ratings for user 5 and user 6 in 100k movie lens dataset. **b** Predicted ratings for user 5 and user 6 in 100k movie lens dataset

15.5 Movie Recommendation Using Spark

In this section, a recommendation system is built using Apache Spark. The dataset considered for the recommendation is a random dataset. The random dataset is created using the following code. A file *als_data.csv* is created with three columns where each column represents the movie rating. A sample of 20 users are created with their ratings are created. A scale of 1–5 is assumed for the rating where 1 indicates the worst and 5 indicates excellent. A sample of the data is shown in Table 15.3.

Table 15.3 Movie dataset for Spark

1	1	1
3	1	1
1	3	1
4	2	1
2	1	0
2	1	0
2	1	1
1	4	1
4	4	1
4	4	1

```
import random
with open('als_data.csv', 'w') as f:
    for i in range(20):
        f.write("{}{},{}\n".format(random.randint(1, 4), random.randint(1, 4), random.randint(0, 1)))
```

The following code demonstrates the recommendation using the Apache Spark. The modules required are Spark context, matrix factorization model, and the rating. The dataset is first loaded into the Spark and the context is created for the movie recommendation. From each of the column, the ratings are extracted one-by-one.

A matrix factorization model is built for the training set first using the latent features. The latent features are obtained using the alternate squares method. The number of iterations is fixed to 20 where the mean square error is estimated for each iteration. If the mean square error is less than the previous iterations, the method is terminated to get the final predicted matrix.

```
from pyspark import SparkContext
from pyspark.mllib.recommendation import ALS, Matrix
FactorizationModel, Rating
def mse(a,b):
    sum = 0
    for i in range(len(a)):
        sum = sum + (a[i]-b[i])**2
    return sum/len(a)
sc = SparkContext()

#load the data
data = sc.textFile('als_data.csv').map(lambda x: x.split(','))
print(data.collect())
```

```

#Get the rankings of each user input.
movie_input = data.map(lambda x : Rating(int(x[0]), int(x[1]), int(x[2])))

#Train the Alternate Least Squares Model.
latent_factors = 10
iterations = 20
movie_recommender = ALS.train(movie_input, latent_factors, iterations)

# Predict the model over the training dataset itself.
test_ds = data.map( lambda x : (x[0], x[1]))
result = movie_recommender.predictAll(test_ds)

# convert the input predictions into a tuple of length 3.
a = movie_input.map(lambda x : (x[0],x[1],x[2])).collect()
r = result.map(lambda x : x[2] ).collect()

# get the MSE loss
A = movie_input.map(lambda x : x[2]).collect()
r_forLoss = result.map(lambda x : x[2]).collect()
MSE = mse(A,r_forLoss)
print("MSE loss : {0}".format(MSE))

# Print the results.
print("User \t Movie \t actual rating \t predicted rating")
for i in range(len(r)):

    print("{0}\t{1}\t{2}\t{3}".format(a[i][0],a[i][1],a[i][2],round(r[i])))

```

Table 15.4 shows the predicted ratings of the model used. It can be seen that the predicted ratings and the actual ratings are close to each other. In this way, the prediction of ratings can be obtained in Spark.

Table 15.4 Predicted ratings for Spark

1	1	1
3	1	1
1	3	1
4	2	1
2	1	0
2	1	0
1	4	1
4	4	1
4	4	1

15.6 Exercises

1. Give examples of the real-world recommendation systems.
2. Compare collaborative filtering and content-based filtering systems.
3. Compare user-based collaborative filtering and item-based collaborative filtering methods for recommendation systems.
4. What are the other techniques of collaborative filtering other than matrix factorization?
5. Implement the recommendation case study using matrix factorization on the newsgroup dataset provided by <https://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups>
6. Implement the recommendation case study with the following rating matrix.

Movie 1	Movie 2	Movie 3
1	1	1
3	0	1
0	3	0
4	0	1

References

1. Pazzani, M. J., & Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web* (pp. 325–341). Heidelberg: Springer Berlin Heidelberg.
2. Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 4.
3. Adomavicius, G., Tuzhilin, A. (2005, June). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6).
4. Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, (2009), 19.
5. Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *ACM Conference on Knowledge Discovery and Data Mining* (pp. 426–434).
6. Keshavan, R., Montanari, A., & Oh, S. (2010). Matrix completion from noisy entries. *Journal of Machine Learning Research*, 11, 2057–2078.
7. Pradel, B., Sean, S., Delporte, J., Guérif, S., Rouveiro, C., Usunier, N., et al. (2011). A case study in a recommender system based on purchase data. In *Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining* (KDD ‘11) (pp. 377–385). ACM, New York, NY, USA.
8. Harper, F. M., & Konstan, J. A. (2015, December). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems* (TiiS), 5(4), 19, Article 19. <http://dx.doi.org/10.1145/2827872>.

Part IV

Data Visualization

Chapter 16

Introduction to Data Visualization



16.1 Introduction to Data Visualization

Data are generated by a variety of sources such as financial transactions, air quality, population, traffic which are generally numeric in nature. This information has value and meaning when the data are processed and analyzed. It helps to take informed decisions. A single picture contains a wealth of information that can be processed more quickly than a couple of words [1]. Visualization has always been the communication of information where the text cannot be interpreted. This is because the process of reading text is limited by speed of reading. In comparison, visualization on other hand can be processed by human perceptual system. Visualization may be in the form of picture, map, or graphs. In the rising edge of computing and data analytics, visualization forms a key role in enabling the analytics more interactive and understandable.

Data Visualization is a leading research area in the fields such as genomic analysis, vehicular networks, online retail, tourism. It provides a visualization of data in a graphical or pictorial way enabling the decision makers to identify new patterns in data. Hence, it provides a way to deal with data in a variety of ways and understand the domain areas in terms of clear factors of customer behavior, predicting the sales amount, and the areas that require more attention. Data Visualization can be defined as an effort to make understand the significance of data in visual context [1, 2]. Various trends, patterns, and relations that cannot be detected in text-based data can be recognized easily with data visualization.

Several datasets related to the fields of medicine, transport, and others can be visualized through various channels for understanding them more efficiently. The datasets have to be transformed into appropriate formats first and start the visualization. Data Visualization incorporates human capabilities into the visual interface and thereby combining machine intelligence and human intelligence. The major disciplines in data visualization are scientific visualization, information visualization, and visual analytics. In scientific visualization, structures and evolutions of

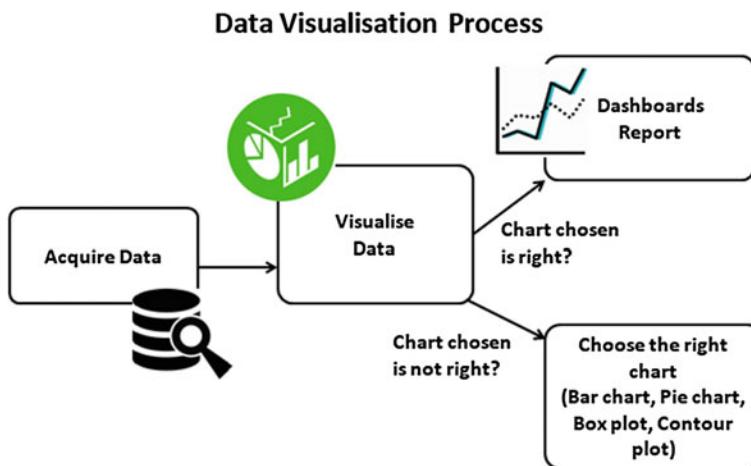


Fig. 16.1 Data visualization process

physical and chemical elements are analyzed. High-dimensional data, unstructured information, and abstract information are analyzed through information visualization. Visual analytics involves analyzing data iteratively, dynamic and interact with data [3–5].

Data visualization process involves data collection and data transformation and presenting the data in a visual form for facilitating exploration of data for further understanding. Most of the works related to visualization process involve three main steps, namely data transformation, visual transformation, and user interaction. Figure 16.1 shows the general steps involved in the data visualization process.

The very first step in the process is to collect the data from different sources. It can be either quantitative/qualitative data. The second step of the process is to transform the data into appropriate forms for visualization. The intermediate step (optional) is to put the data into data warehouse for visualization. This data in the data warehouse can be used for various purposes in the visualization process when the transformation is not in the correct representation format. From the transformed data, different set of views such as bar graphs, pie charts, histogram, box plots can be represented in the visualization step. Finally, in the last step, user can interact with the views produced during the visualization steps and change the structural parameters of it through the transformation step in the process.

16.2 Importance of Data Visualization

Data Visualization has become an integral part of business intelligence and analytics. Data Visualization tools like Tableau, Qlik, and others success has paved the way for visualization in various fields of research in computer science.

Data visualization tools help in democratizing data and provide insights into it. It provides an easy way to use compared to statistical tools for analysis. It plays an important role in Big data research area as it requires advanced analytics and processing capabilities. Visualization helps to reduce the effort in identifying the right data models for analysis.

Data visualization is being widely used in a variety of applications such as business data [4], scientific data [5], student histories [7], sports data [6], images, and videos [8]. Financial analysts and human resource managers have better understandings of data through visualization. Data visualization in these fields has been considered as one of the ways for increasing growth, productivity, and innovation. Most of the people in public examine data such as product specification, blogs, online communities seek health-related information [9]. Thus, with the recent advances in data visualization tools help such kind of people in analyzing the data more efficiently.

In the field of Big data analysis and processing, data scientist plays a key role for identifying the models and use it to predict the further outcomes. Visualization helps data scientist to monitor the results in the intermediate stages of the data analysis process. The main goal of data visualization is to identify the patterns and relationships that are contained in the data. It should not only mean to good view of the data but should be meaningful in order to make better decisions. Visualization provides a way for analyzing complex data to describe and explore efficiently by summarizing the main characteristics of the data [9]. The important goals for visualization are listed as follows:

- Finding hidden patterns
- Detecting data abnormality
- Determining the relationships between various variables in the data
- Select a preliminary model for analysis

The problems that are related to real world are intrinsically hard to solve. Machine learning approaches are preferred for data analysis problems where data are complex and hard to solve. The analytical power of the approaches of machine learning cannot be used effectively without the intervention of humans. Therefore, it is crucial to incorporate the knowledge, insight, and feedback of the human intelligence into the analytical process. Machine learning focuses on nonlinear aspects of the data and mathematical formalization, whereas visualization focuses on human perceptions and puts the user into center. The aim of modern data visualization tools is to integrate such capabilities so that user can interact with machine learning model and find insights [10].

Example of the importance of visualization

In this section, a small example demonstrates the importance of visualization. For this discussion, the example considered is as shown in Table 16.1. If in a scenario where the sales against different products need to be determined, then it is difficult to tabulate from the table. Visualizations in these scenarios help in answering such questions.

Table 16.1 Product_Years_Sales_Data

Year	Product	Sales
2001	A	10
2002	A	12
2001	B	4
2003	B	5
2004	A	10
2001	A	11
2001	C	12
2002	C	15
2001	C	11
2002	C	9
2003	A	7
2001	A	4
2001	B	6
2003	B	10
2010	C	11
2010	C	23
2009	C	11
2003	A	12
2004	A	6
2010	B	8
2010	B	7
2003	A	21
2004	A	12
2005	B	13

A graph as shown in the Fig. 16.2 can be one of the examples that can answer the question of what is the sales of each product? It can be observed from the Fig. 16.2 that the ratio of the products A and C sold is approximately equal and more than the product B in the year 2001. But, in the year 2002, no B products were sold than A and C. The plot is obtained using stacked bar chart in Python. In this way, stacked bar charts are used to answer the questions easily than the tabular way.

16.3 Principles of Data Visualization

Any visualization on data needs to be accurate and present the findings of analysis rather than reiterating the process of visualization needs. Some of the principles of visualization are listed as follows:

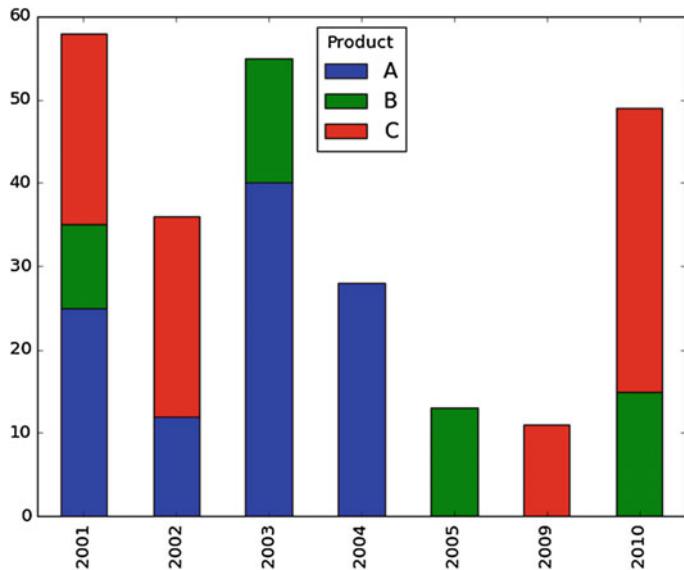


Fig. 16.2 Importance of visualization using stacked chart

- **Simple:** The visualization on data needs to be simple and includes all the necessary components of the analysis. However, it should not be oversimplified as well that explains in detail everything about the data. For example, the legends in the graph make it easy and simple to understand the data, but the legends or capturing the division of units may make it oversimplifying.
- **Comparison:** The charts or the dashboards created for visualization need to be in such a way that it can be compared with one another to evaluate the correct one. For example, comparison charts exist in two forms, namely bar graph or line graph. Both of them can be plotted for the same data to see which can be used appropriately for the visualization.
- **Diverse:** Data visualization charts need to be diversified so that different views or perspectives are possible on the same dataset. For example, in some cases of visualization, both comparison charts and composition charts can be used and aggregated to one plot signifying the views of both of them.
- **Objectivity:** Before starting the visualization, the main objective of the visualization should be framed and according to the objective the charts needs to be visualized. For example, if the objective is to determine the sales revenue for tenure of years, it can be visualized as shown in the Fig. 16.3.
- **Nimble:** The charts used for visualization should be such that it should nimble, i.e., quick and easy to understand with less information. Suppose if it contains more information, then it can be transformed into two charts aggregated into one another where on a click extra information is displayed. For example, consider the chart shown in Fig. 16.4 on the population of children in a specific region

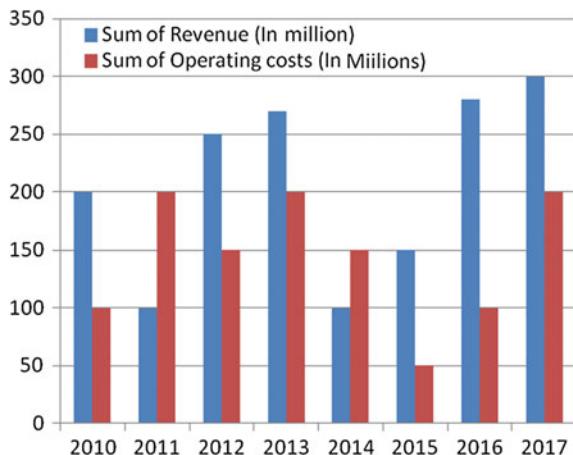


Fig. 16.3 Objectivity principle on visualization

from 2010 to 2017. In Fig. 16.4, year-wise statistics are shown. On clicking the year, month-wise statistics are shown in the Fig. 16.5. In this way, the charts can be aggregated to one and another.

- **Understanding your data:** Before starting the visualization, the data that is being used for visualization should be properly analyzed with different machine learning techniques as discussed in the previous part in this book. It can be understanding variables, which are the variables that need to be plotted on *X*-axis and *Y*-axis, what is the correlation between variables and so on.

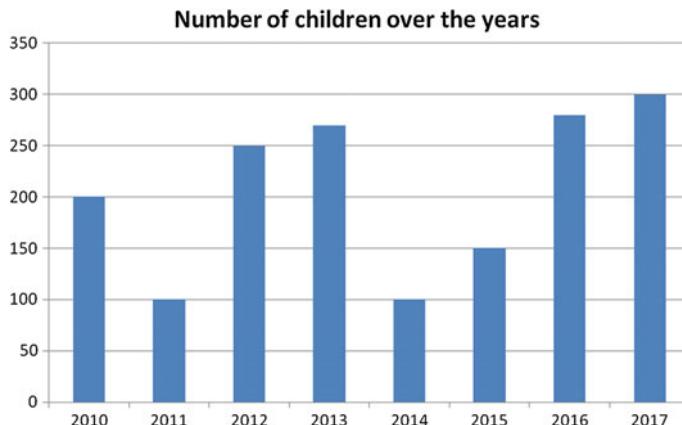


Fig. 16.4 Number of children over the years

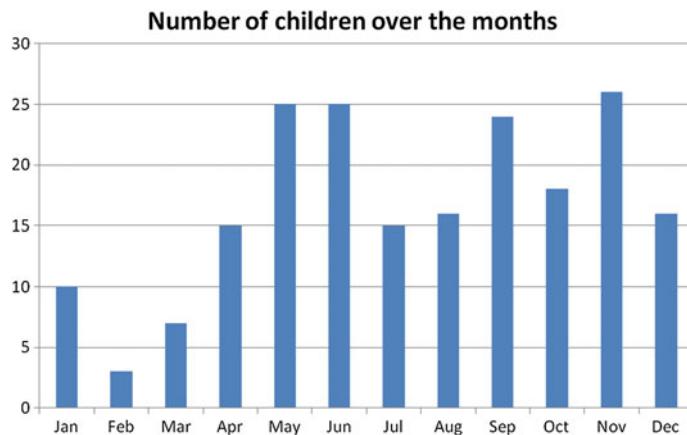


Fig. 16.5 Number of children over the months

16.4 Popular Visualization Tools

R

R can be used for visualization and supports various types of charts. It supports many applications that can be visualized in a simple and contains most of the information in the plot. For example, one of the plots is as shown in the Fig. 16.6. The plot is created using the R code and ggplot library. The library ggplot() supports visualization for various formats of data and machine learning models. The other libraries that support visualization in R are ggiraph, digraph, ggVis, rAmChart. An example of ggiraph visualization is as shown in the Fig. 16.7. Since many libraries are available for visualization in R, there is no common library that developers will use for visualization. It is difficult for the beginners in R programming to learn the different functionalities of libraries in R.

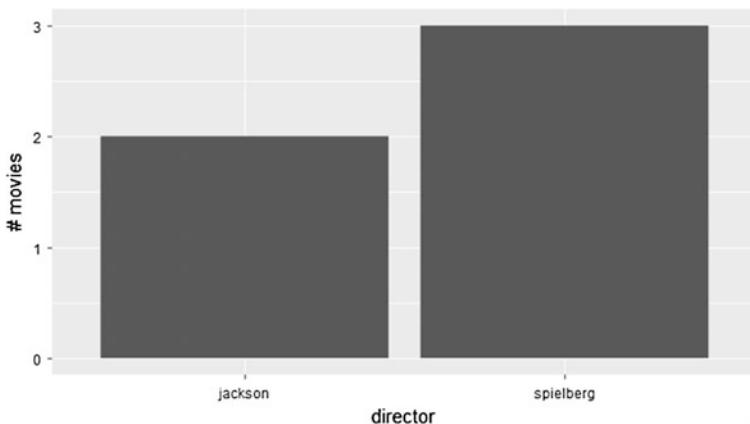


Fig. 16.6 Visualization using R

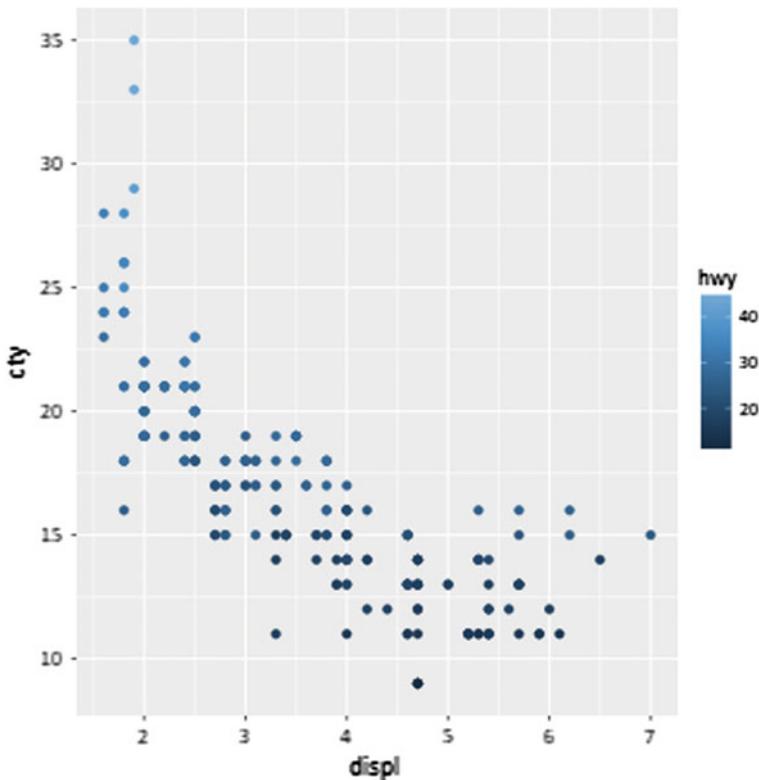


Fig. 16.7 Viusalisation using ggiraph using R

Python

Python supports visualization using many modules as explained in the machine learning (part 2) of the book. The main module that is used for visualization is matplotlib with plot() as the function. It can be used with various machine learning models like regression, clustering, classification, decision trees. For example, one of the plots using scatter function is as shown in the Fig. 16.8. It is a plot of GDP and life expectancy rate. In this book, Python language is used for demonstrating the examples of different charts of visualization. The main benefit of using Python for visualization is the base module matplotlib which is used for all the programs in visualization.

The modules supported by Python for visualization are matplotlib, pygal, graphviz, plotly, geoplotlib, bokeh, and others. One of the plots of the geoplotlib is as shown in the Fig. 16.9. The plot shows the different air fleets in the world. It provides the world map to visualized based on the coordinates of the source and destination in the map. In this way, using Python different visualizations are possible.

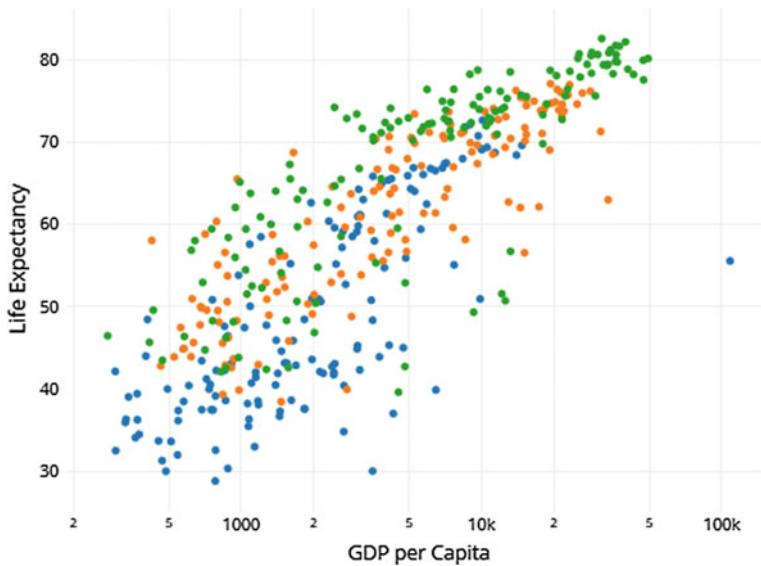


Fig. 16.8 Visualization in Python

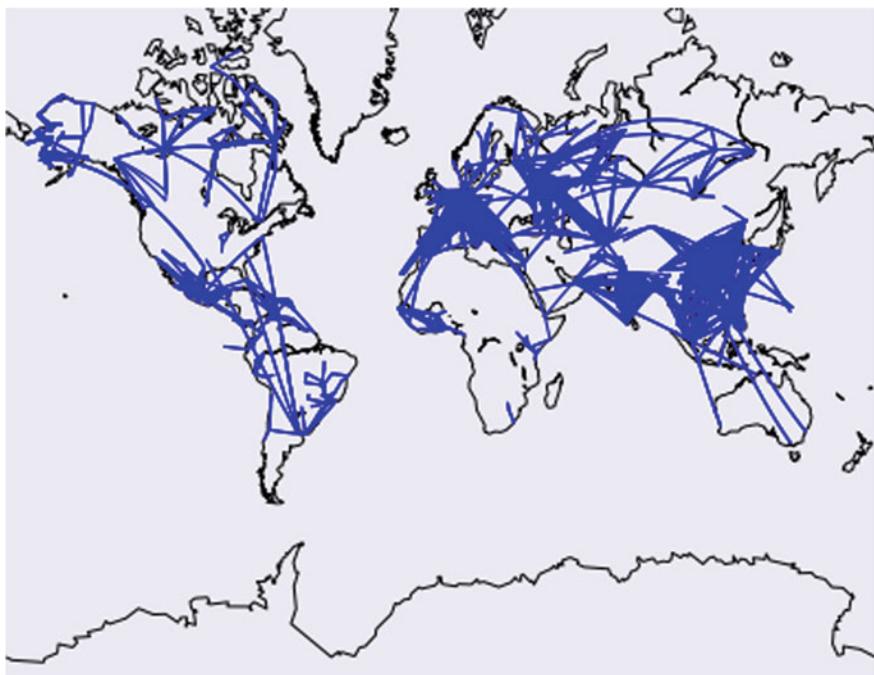
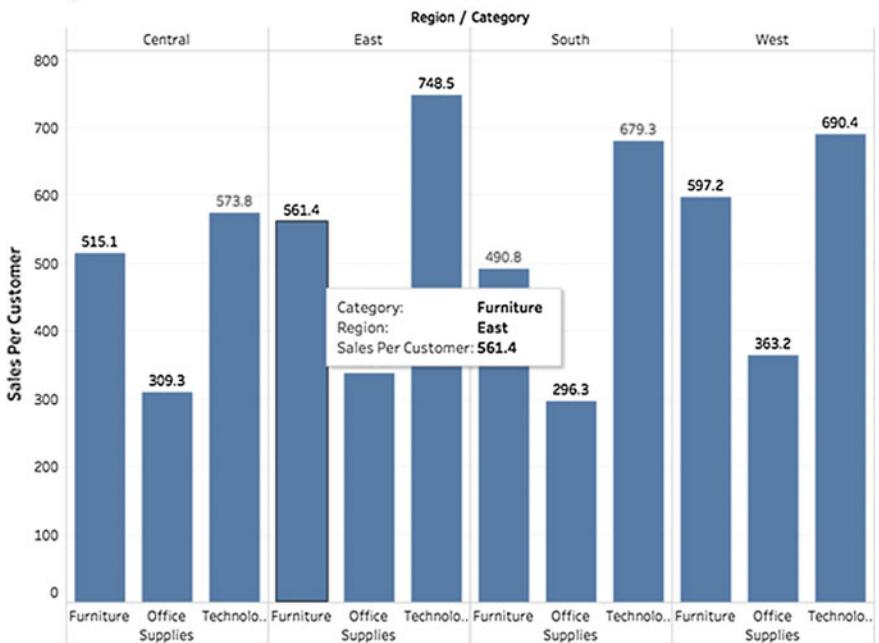


Fig. 16.9 Visualization using geoplotlib in Python

Average Sales Per Customer**Fig. 16.10** Visualization using Tableau

Tableau

Tableau is one of the advanced visualization tools that supports data visualization from various file systems. The main support of Tableau includes Hadoop, a distributed file system that is widely used for large-scale data analysis. It provides a dashboard for analysis with various charts. The main advantage is the dashboard provided automatically chooses the right chart for visualization. An example of the charts is as shown in the Fig. 16.10 in the dashboard of Tableau. It can be observed in the Fig. 16.10 that for each bar in the graph, automatically the value is placed, whereas in the case of other visualization tools, these values have to be explicitly specified.

16.5 Exercises

1. Name few research areas in data visualization.
2. Describe the steps involved in data visualization process.
3. What is the main goal of data visualization? Is data visualization needed for every machine learning technique?

4. Which area of data visualization focuses on nonlinear aspects of the data also mathematical formalization?
5. What type of data is collected during data collection phase that assists for data visualization?

References

1. Hansen, C. D., & Johnson, C. R. (2011). *Visualization handbook*. Cambridge: Academic Press.
2. Ware, C. (2012). *Information visualization: perception for design*. Amsterdam: Elsevier.
3. Cook, K. A., & Thomas, J. J. (2005). Illuminating the path: The research and development agenda for visual analytics.
4. Cao, N., Lin, Y. R., Sun, X., Lazer, D., Liu, S., & Qu, H. (2012). Whisper: Tracing the spatiotemporal process of information diffusion in real time. *IEEE Transactions on Visualization and Computer Graphics*, 18(12), 2649–2658.
5. ElHakim, R., & ElHelw, M. (2010). Interactive 3d visualization for wireless sensor networks. *The Visual Computer*, 26(6), 1071–1077.
6. Hansen, C. D., & Johnson, C. R. (2011). *Visualization handbook*. Cambridge: Academic Press.
7. Pileggi, H., Stolper, C. D., Boyle, J. M., & Stasko, J. T. (2012). Snapshot: Visualization to propel ice hockey analytics. *IEEE Transactions on Visualization and Computer Graphics*, 18(12), 2819–2828.
8. Chen, T., Lu, A., & Hu, S. M. (2012). Visual storylines: Semantic visualization of movie sequence. *Computers & Graphics*, 36(4), 241–249.
9. Elmqvist, N., Dragicevic, P., & Fekete, J. D. (2008). Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE transactions on Visualization and Computer Graphics*, 14(6), 1539–1148.
10. Ellis, G., & Mansmann, F. (2010). Mastering the information age solving problems with visual analytics. In *Eurographics* (vol. 2, p. 5).

Chapter 17

Getting Started with Visualization in Python



The visualization process discussed in this previous chapter addressed the key principles of the visualization. In this chapter, first the modules required for visualization in Python is discussed first and small examples are presented. The major libraries required for visualization are matplotlib and graphviz. Many functions such as plot(), boxplot(), subplot() are used for drawing the graphs required for visualization.

17.1 Matplotlib

Matplotlib is an extensively used package for scientific visualizations to generate publication quality images [1]. It is a 2D plotting library with MATLAB like interface to generate simple plots using the *pyplot* module. Even though, 2D plots are widely used in data science, matplotlib provides modules to generate 3D plots and animations. Matplotlib is used to generate various kinds of plots. Few of the basic features of matplotlib include the following.

- Line plots: plot()
- Scatter plots: scatter()
- Histograms: hist()
- Bar charts: bar()
- Pie charts: pie()
- Subplots: subplot()
- Legends: legend()

All these functionalities are available under the module of `matplotlib.pyplot`. Apart from these, `matplotlib` can be used to display images using `show()` module.

17.2 Graphviz

Graphviz is an open-source graph visualization software used to represent structural information as diagrams of abstract graphs and networks [2]. Graphviz is used to generate graphs using the following workflow:

1. Create a graph object.
2. Assemble the graph by adding nodes and edges.
3. Retrieve its DOT source code string.
4. Save the source code to a file and render it with the graphviz installation of your system.

Matplotlib and graphviz can be installed in Linux or Windows using a Python package manager like pip or conda. Graphviz needs an extra library to be installed in order to render the graphs. This library is called ‘dot.’

17.3 Installing Matplotlib in Windows

Once, the Python is installed in Windows as discussed in the Appendix section, matplotlib can be installed using the following steps.

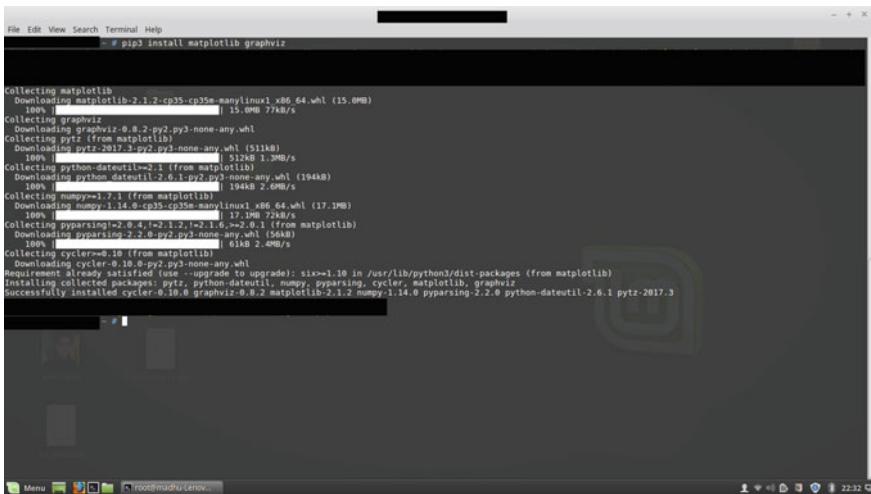
- Open command prompt.
- pip install matplotlib graphviz

17.4 Installing Matplotlib in Ubuntu

Once, the Python is installed in Windows as discussed in the Appendix section, matplotlib can be installed using the following steps.

- pip install -U matplotlib graphviz (pip package manager)
- conda install -c conda-forge matplotlib (Anaconda package manager)
- sudo apt-get install dot (in Ubuntu-based systems)

To confirm that matplotlib and graphviz modules are installed, a small example discussed in the upcoming sections can be used and the successful message needs to be displayed as shown in Fig. 17.1.



```
File Edit View Search Terminal Help
# pip3 install matplotlib graphviz

Collecting matplotlib
  Downloading matplotlib-2.1.2-cp35-cp35m-manylinux1_x86_64.whl (15.0MB)
    100% |████████████████████████████████| 15.0MB 77kB/s
Collecting graphviz
  Downloading graphviz-0.8.2-py2.py3-none-any.whl
Collecting pytz (from matplotlib)
  Downloading pytz-2017.3-py2.py3-none-any.whl (511kB)
Collecting python-dateutil==2.6.1 (from matplotlib)
  Downloading python_dateutil-2.6.1-py2.py3-none-any.whl (194kB)
Collecting numpy==1.7.1 (from matplotlib)
  Downloading numpy-1.14.0-cp35-cp35m-manylinux1_x86_64.whl (17.1MB)
Collecting pyarsing==2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 (from matplotlib)
  Downloading pyParsing-2.2.0-py2.py3-none-any.whl (564kB)
Collecting cycler==0.10 (from matplotlib)
  Downloading cycler-0.10.0-py2.py3-none-any.whl
Requirement already satisfied: typeguard<2.0,>=1.9.0 (from matplotlib)
Collecting six<1.10,>=1.5.2 (from matplotlib)
  Downloading six-1.10.0-py2.py3-none-any.whl (10kB)
Successfully installed cycler-0.10.0 graphviz-0.8.2 matplotlib-2.1.2 numpy-1.14.0 pyarsing-2.2.0 python-dateutil-2.6.1 pytz-2017.3
```

Fig. 17.1 Matplotlib and graphviz module success message

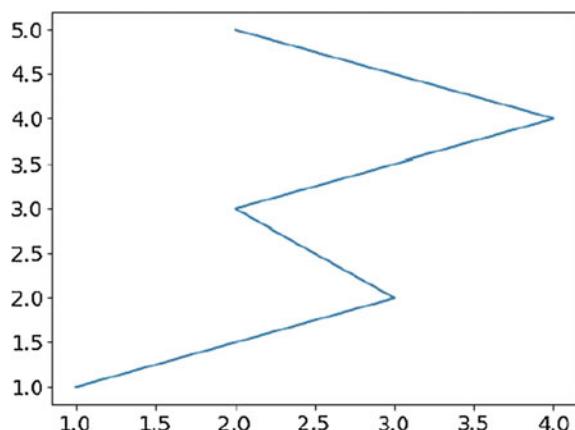
17.5 Small Examples with Matplotlib

In this section, small examples in matplotlib on line plots, subplots are shown. These examples can be used to verify the installations of the modules of matplotlib and graphviz [3–5].

Line Plots

The following code demonstrates a simple example of plotting a line given the x and y values of the data. The output of the plot is as shown in Fig. 17.2. Two list values x and y are initialised first, and the line plot is drawn for the same using the `plot()` function.

Fig. 17.2 Line plot example using matplotlib



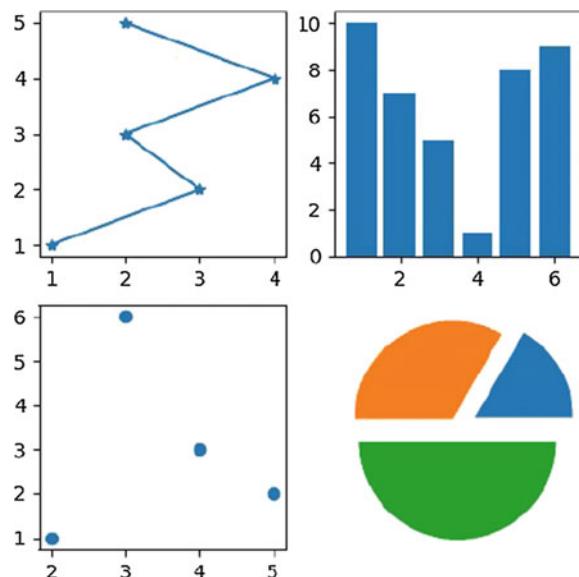
```
import matplotlib.pyplot as plt
x=[1,3,2,4,2]
y=[1,2,3,4,5]
plt.plot(x,y)
plt.show()
```

Subplots

The following code demonstrates how to incorporate multiple plots in a single Fig. 17.3 using subplot() method. The subplot() function is used to create the plots pie chart, bar chart, and the line chart as shown in Fig. 17.3.

```
import matplotlib.pyplot as plt
fig, axs = plt.subplots(2, 2, figsize=(5, 5))
x=[1,3,2,4,2]
y=[1,2,3,4,5]
#block 1
axs[0, 0].plot(x,y,marker="*")
x1 = [2,3,4,5]
y1 = [1,6,3,2]
#block 2
axs[1, 0].scatter(x1,y1)
x_bar=["Python","Java","c++","c","R","Julia"]
y_bar=[10,7,5,1,8,9]
ypos=[i for i in range(1,len(x_bar)+1)]
```

Fig. 17.3 Subplots example using matplotlib



```
axs[0, 1].bar(ypos,y_bar,align='center') #block 3  
axs[1, 1].pie([1,2,3],explode=[0.2,0.1,0.15]) #block 4  
plt.show()
```

17.6 Exercises

1. What is matplotlib module? Does Matplotlib allow users to plot 2D and 3D plots?
2. What is the workflow for generating graph in Graphviz?
3. Define the different types of charts with matplotlib module.
4. Discuss the advantages of visualization with matplotlib module.

References

1. Matplotlib: <https://matplotlib.org/>.
2. Graphviz: <https://pypi.python.org/pypi/graphviz>.
3. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95.
4. Ellson, J., Gansner, E., Koutsofios, L., North, S. C., & Woodhull, G. (2001, September). Graphviz—open source graph drawing tools. In *International symposium on graph drawing* (pp. 483–484). Springer, Berlin, Heidelberg.
5. Gansner, E. R. (2009). Drawing graphs with graphviz. *Technical report, AT&T Bell Laboratories, Murray, Tech. Rep, Tech. Rep*.

Chapter 18

Visualization Charts



Data visualization is a compelling process that involves choosing the right chart type. The charts used for the visualization should adhere to the principles of visualization as discussed in the previous section. The main steps involved in the visualization are to understand type of data, choosing the right visualization chart and crafting a story on the chart created [1]. Some of the visualization charts are listed as follows:

- **Comparison charts:** These charts are used for comparison of two or more variables [2], for example, comparing the means of height and weight of people dataset.
- **Composition charts:** These charts are used for showing the composition of different data points in a dataset, for example, the percentage of products such as TV, washing machine, refrigerator purchased by the customers.
- **Distribution charts:** These charts are used to show the distribution of data and help in understanding the underlying distribution of the data. For example, there might be normal distribution of sales over a month in customer purchase dataset.
- **Relationship charts:** These charts help in answering the questions for identifying the relations among the variables in the data. For example, before we carry out regression analysis, scatter plot is used to identify the different types of relationships among the variables.

In this section, the different types of charts that are needed for visualization are discussed with examples. These charts are basic examples of visualization, and it can be used for advanced analytics as well [3–5].

18.1 Comparison Charts

Comparison charts are used for the analysis of two datasets that exist in various forms. The different forms of the comparison charts are bar chart and line chart. These charts are generally used to study the greater and lesser values of the data.

For example, if height and weight variables are present in the data, then one of the lines can represent the height data. The other line in the chart can represent the weight data in the chart. The comparison charts are used in this way to compare two different features in the data [3].

18.1.1 Comparison Bar Charts

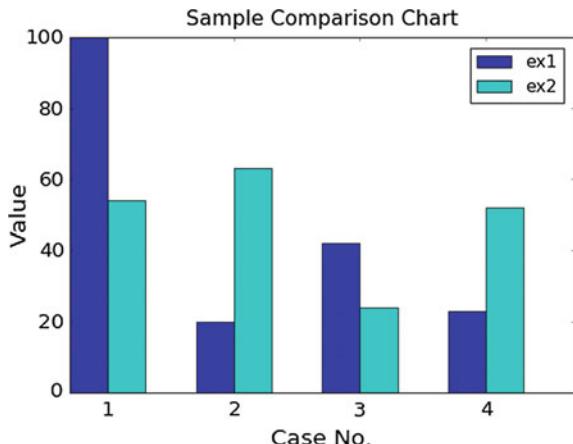
In this section, a small example on comparison bar chart is as shown below. The module matplotlib is used for visualization. Since the comparison charts are used, a number of groups are needed first. Here, the number of groups is initialized to 4. The parameters of the graph for plotting are index, bar_width, and opacity. Here, the bar_width is initialized to 0.3 and opacity is initialized to 0.85. The plt() function in matplotlib module is used to draw the graph as shown in Fig. 18.1.

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

#Number of groups of data and arbitrary data to be plotted
number_groups = 4
means_ex1 = (100,20,42,23)
means_ex2 = (54,63,24,52)

#Graph parameters
fig,ax = plt.subplots()
index = np.arange(number_groups)
bar_width = 0.3
opacity = 0.85
```

Fig. 18.1 Comparison bar chart example



```

#The definition for the two graphs
rects1 = plt.bar(index,means_ex1,bar_width,alpha=opacity,color='b',
label='ex1')
rects1 = plt.bar(index+bar_width,means_ex2,bar_width,
alpha=opacity,color='c',label='ex2')

#X and Y labels and legend definitions
plt.xlabel('Case No.')
plt.ylabel('Value')
plt.title('Sample Comparison Chart')
plt.xticks(index+bar_width,('1','2','3','4'))
plt.legend()
plt.tight_layout()
plt.show()

```

In the code, `plt.bar()` function is used to display the bar plot as shown in Fig. 18.1. It can be seen from the plot that in the first instance, the `means_ex1` has a higher value than the `means_ex2`. In this way, comparison can be done easily with the chart.

18.1.2 Comparison Line Charts

In the previous section, the comparison charts in a bar representation were discussed with an example. If the values in the data are discrete and are separated by equal intervals, then the bar representation of the comparison chart is useful. However, if the values in the data are continuous, then it might not be suitable for visualization.

In this section, a small example on comparison line chart is as shown below. The same module `matplotlib` is used for visualization. The main difference between the comparison bar chart and comparison line chart is there is no need of groups in the case of line charts. Instead here, the coordinates of x and y points are needed. Thus, the coordinates `x1`, `x2`, `y1`, `y2` are initialized for the comparison. The same `plot()` function is used to draw the line graph as shown in Fig. 18.2.

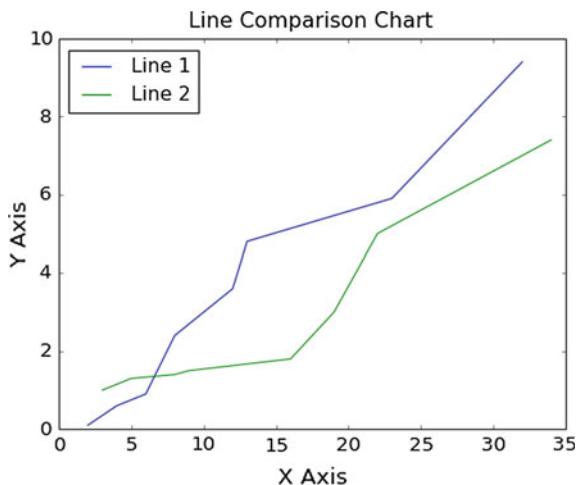
```

import matplotlib.pyplot as plt
import numpy as np

#Arbitrary X and Y values for two different items
x1 = (2,4,6,8,12,13,23,32)
x2 = (3,5,8,9,16,19,22,34)
y1 = (0.1,0.6,0.9,2.4,3.6,4.8,5.9,9.4)
y2 = (1,1.3,1.4,1.5,1.8,3,5,7.4)
plt.plot(x1,y1)
plt.plot(x2,y2)

```

Fig. 18.2 Comparison line chart example



```
#X and Y labels and legend definitions
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.title('Line Comparison Chart')
plt.legend(['Line 1', 'Line 2'], loc='upper left')
plt.show()
```

18.1.3 Comparison Charts for Brand Analysis

Small examples with random data for visualization of comparison charts were discussed in the previous section. These examples are created with random dataset. For the actual data, the values/points may vary in the graph. The applications where comparison charts are employed include brand analysis, total number bandwidth usage analysis.

In this section, a dataset example is used for comparison chart analysis. The dataset considered for this example is as shown in Table 18.1. The table displays only few rows of the dataset. It is based on the number of people using a particular brand of mobile phones. A table view of such large dataset of number of people using the mobile set may not infer interesting results. In such cases, the comparison charts help in finding the insights about the data.

The following code demonstrates the usage of comparison charts for the mobile data considered. Here, the line chart is used for the comparison. Thus, x and y coordinates are needed. x, y, y1, and y2 are the arrays that are initialized with mobile data from the csv file. Firstly, the data are read from the csv file and appended to the x, y, x1, and y1 lists for analysis. The same plot() function in the matplotlib module is used for the visualization.

Table 18.1 Mobile data for comparison charts

Country	Samsung	Nokia	LG
Alabama	4	13	136
Alaska	4	10	163
Arizona	4	8	194
Arkansas	3	8	90
California	4	9	76
Colorado	3	7	104
Connecticut	2	3	10
Delaware	4	5	138
Florida	4	15	235
Georgia	3	17	111
Hawaii	1	5	46
Idaho	2	2	20
Illinois	4	10	149
Indiana	2	7	13

The coordinates of x and y are used for Samsung mobile users. Similarly, the coordinates of x and y1 are used for Nokia mobile users. For LG mobile users, x and y2 coordinates are used. The output of the plot is shown in Fig. 18.3. From the plot, it can be clearly observed that the number of users of LG brand is more in number than the Nokia and Samsung. In this way, comparison charts can be used to compare the data among different variables.

```

import matplotlib.pyplot as my_plt
import csv
x=[]
y=[]
y1=[]
y2=[]

# Read the data from CSV
with open('mobile_data.csv') as csvfile:
    readCSV = csv.reader(csvfile, delimiter=',')
    for row in readCSV:
        print(row)
        x.append((row[0]))
        y.append((row[1]))
        y1.append((row[2]))
        y2.append((row[3]))


my_plt.plot(x,y, label='Samsung: Mobile Users')
my_plt.plot(x,y1, label='Nokia: Mobile Users')
my_plt.plot(x,y2, label='LG: Mobile Users')
my_plt.locator_params(tight=True,nbins=55)

```

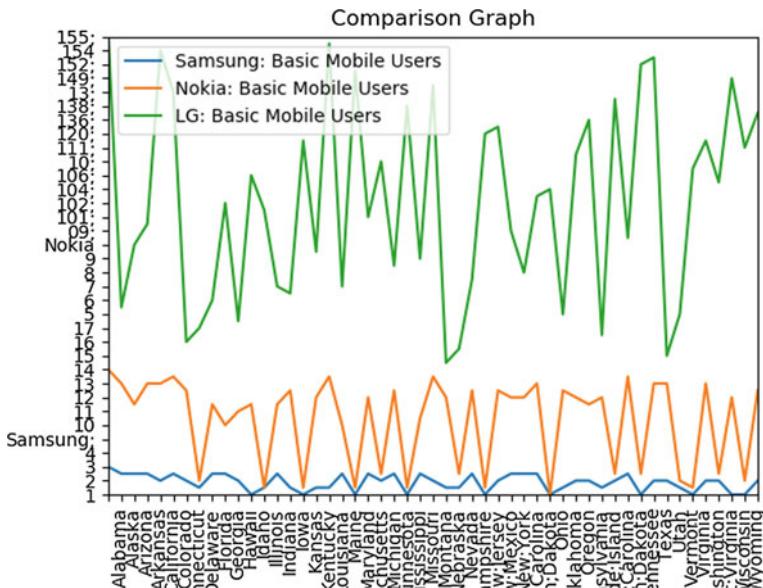


Fig. 18.3 Comparison chart for mobile data

```
my_plt.title('Comparison Graph')
my_plt.xticks(x, rotation='vertical')

# Pad margins
my_plt.margins(0)

# Adjust spacing to prevent clipping of tick labels
my_plt.subplots_adjust(bottom=0.15)
my_plt.legend()
my_plt.show()
```

18.2 Composition Charts

In the previous section, comparison charts discussed do not show the percentage of the values in a dataset. In such cases, composition charts have to be used. Composition charts can be better understood with an example as shown below. The same module matplotlib with plot() function is used for visualization. Here, in this example, random data on the ownership of the pets are considered for visualization. Instead of groups in the comparison charts, sizes are used in composition charts for specifying the percentage of values.

The pie() function of the subplot axl is used to draw the comparison chart with parameters: labels, explode, shadow, and the startangle. The explode option is used

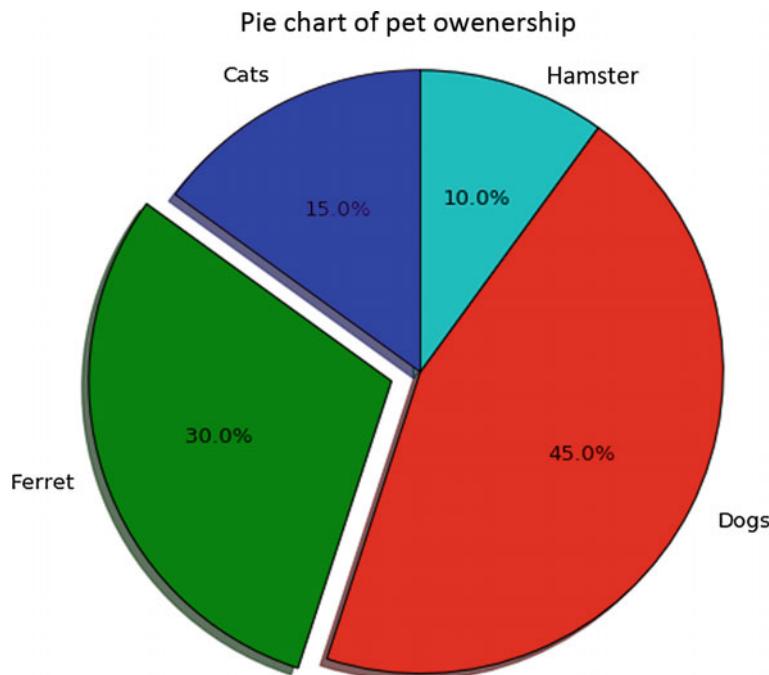


Fig. 18.4 Composition chart example

to highlight a particular slice in the composition chart, and startangle specifies the angle at which the pie chart should start. The ratio of division of axis is made equal with function axis(). The output of the plot() function is as shown in Fig. 18.4. It can be clearly seen from the plot that the percentage of pet ownership can be clearly seen; for example, 45% own dogs, 30% own ferret.

```
import matplotlib.pyplot as plt

#Random data with pet ownership
labels = ['Cats', 'Ferret', 'Dogs', 'Hamster']
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0) # only "Higlight" the 2nd slice

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%.1f%%',
shadow=True, startangle=90)

ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
plt.title("Pie chart of pet ownership")
plt.show()
```

18.2.1 Composition Chart Stacked Bar

The alternative way of obtaining composition charts can be in the form of stacked bar as shown in Fig. 18.5. The sizes are not used in the case of stacked case, but bars are used for plotting. The same module matplotlib itself is used for visualization. The main difference here is there is no display of percentage of the slices as it is displayed in the form of bar. However, a dark line separates the data and signifies the percentage use of it. It can be observed from the figure that Y scores are higher compared to X scores in the data. The black line in the bar separates the data of X and Y. In this way, the composition charts in stacked bar form can be represented.

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

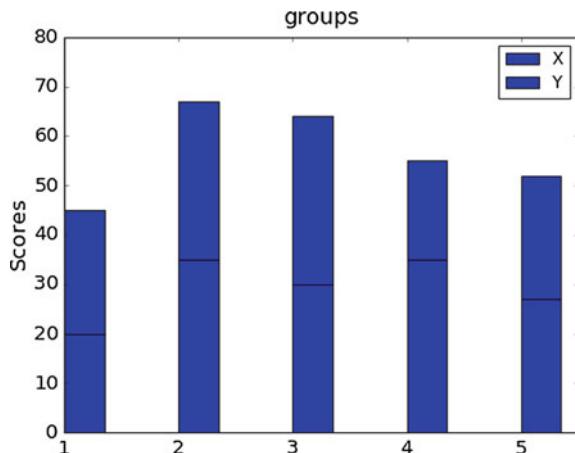
N = 5 #Number of groups
#Random scores for X and Y
X = (20, 35, 30, 35, 27)
Y = (25, 32, 34, 20, 25)
ind = np.arange(N)      # the x locations for the groups
width = 0.35            # the width of the bar

p1 = plt.bar(ind, X, width)
p2 = plt.bar(ind, Y, width,bottom=X)

plt.ylabel('Scores')
plt.title('groups')
plt.xticks(ind, ('1', '2', '3', '4', '5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('X', 'Y'))

plt.show()
```

Fig. 18.5 Composition chart stacked bar example



18.2.2 *Composition Chart on Crime Analysis*

In this section, a small example on crime rate analysis in different cities is visualized through composition chart. The code for visualizing is as shown below. Initially, the labels for different cities are created first. The crime rates for each of the cities are initialized with sizes. The same `pie()` function as earlier is used for plotting the graph. The output of the plot is as shown in Fig. 18.6.

It can be observed from the plot that Bangalore has 13% crime rate, Pondicherry has 5% crime rate, and so on. So, instead of a regular file giving the crime rate, a visual way of interpreting it will be helpful for analysis.

```
import matplotlib.pyplot as plt
# Data
labels = 'Mumbai', 'Bhopal', 'Delhi', 'Noida', 'Hyderabad', 'Pune', 'Kolkatta',
'Lucknow', 'Chennai', 'Gangtok', 'Pondicherry', 'Bangalore'
sizes = [90, 12, 15, 50, 10, 22, 30, 70, 80, 44, 24, 78]
fig1, ax = plt.subplots()
ax.pie(sizes, labels=labels, autopct="%1.0f%%", pctdistance=1.1, shadow=True,
startangle=90, labeldistance=1.3)
# Equal aspect ratio ensures that pie is drawn as a circle.
```

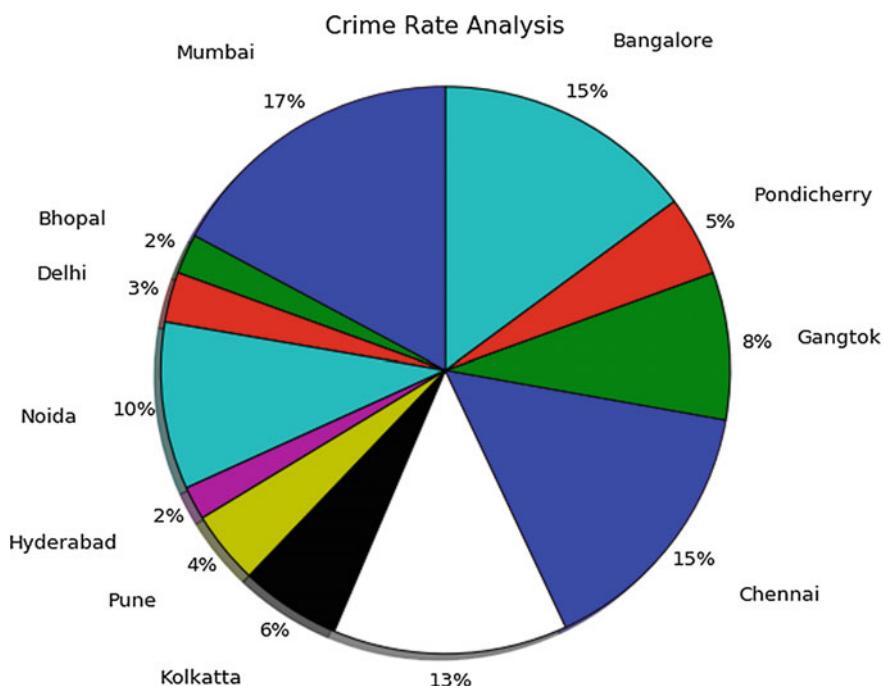


Fig. 18.6 Crime rate analysis using composition chart

```

ax.axis('equal')
# Set the title for the chart
ax.set_title('Crime Rate Analysis', y=1.08)
plt.show()

```

18.3 Distribution Charts

As described earlier, distribution charts are used to identify the underlying distribution of data. In this section, a small example on the same is discussed. Distribution charts can be visualized in two ways namely histogram and histogram with line. Both of these examples are discussed as below.

The following code demonstrates the example on distribution charts with histogram. The example data considered here are a random set of 100,000 samples with standard deviation of 15 and mean as 100. Initially, the number of bins is 20 and hist() function is used to visualize the graph. The output of the plot() function is as shown in Fig. 18.7. In the plot, it can be seen that the data points form a pattern where a line can be drawn to identify the distribution. In the next section, distribution charts with line are discussed.

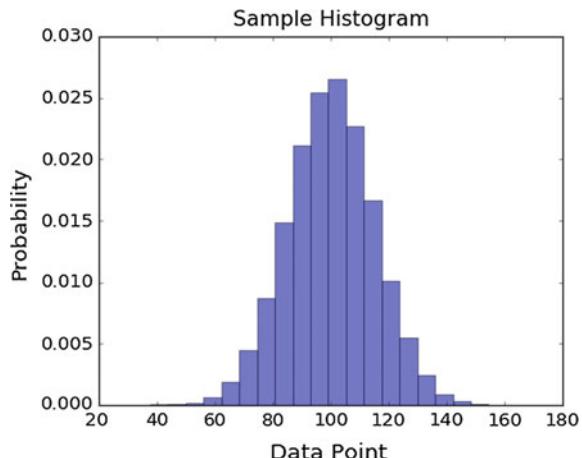
```

import matplotlib.pyplot as plt
import numpy as np

# example data
mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(100000) #Normalised data

```

Fig. 18.7 Distribution charts with histogram example



```

num_bins = 20
# the histogram of the data
n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='blue',
alpha=0.5)

# Marking the Axes
plt.xlabel('Data Point')
plt.ylabel('Probability')
plt.title(r'Sample Histogram')

# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()

```

The following code displays how to fit the line over the distribution of histogram. The function mlab() is used to draw the line over the distribution of the histogram. It will normalize the histogram to the best line as shown in Fig. 18.8. In this way, the underlying distribution of the histogram can be easily identified. In this example, it can be seen that the data points form a normal distribution curve and are uniformly distributed in the chart.

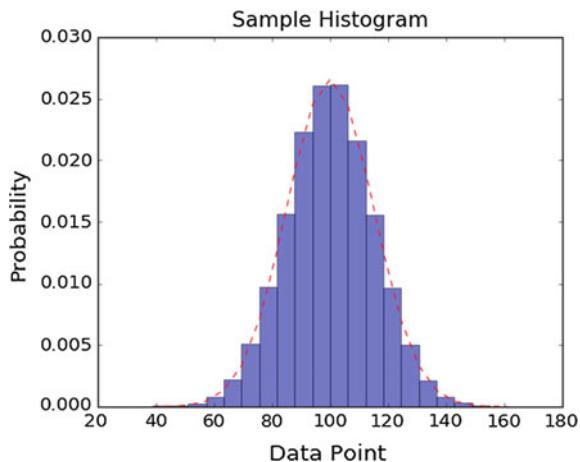
#Histogram with a best fit line

```

import numpy as np
import matplotlib.mlab as mlab #For the best fit line
import matplotlib.pyplot as plt

```

Fig. 18.8 Distribution charts with histogram line fit example



```

# Random example data
mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(100000) #Normalised data

num_bins=20
# the histogram of the data
n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='blue',
alpha=0.5)

# add a 'best fit' line
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')

#Histogram details
plt.xlabel('Data Point')
plt.ylabel('Probability')
plt.title(r'Sample Histogram')

# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()

```

18.3.1 Distribution Chart with Line Histogram Example

The distribution charts can also be formed in a way where only the line of histograms is seen. The following code demonstrates the line histogram example. The main parameter to be included to get a line histogram is ‘histtype = step’. The rest of the modules are same as the distribution charts considered in the previous section. The output of the plot is as shown in Fig. 18.9.

It can be observed from the plot that the line plot of the histogram starts from the value of 40 and ends at 60. This form of chart will give the range of the distribution of the underlying data. Hence, this chart is useful for data that consist of a large number of rows and to find the range of distribution of it.

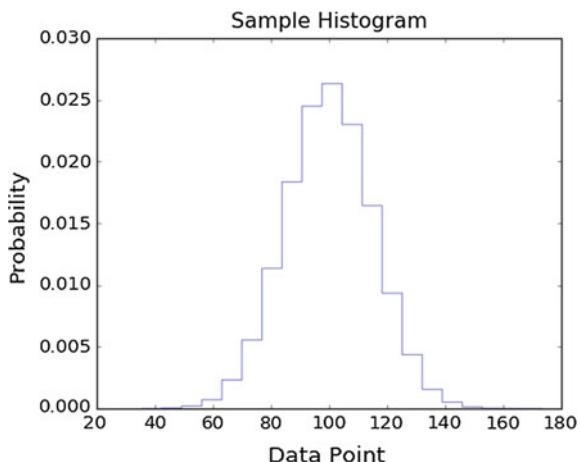
```

import matplotlib.pyplot as plt
import numpy as np

# example data
mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(100000) #Normalised data

```

Fig. 18.9 Distribution chart with line histogram



```

num_bins = 20
# the histogram of the data
n, bins, patches = plt.hist(x, num_bins, normed=1, facecolor='red',
alpha=0.5,histtype='step')

# Marking the Axes
plt.xlabel('Data Point')
plt.ylabel('Probability')
plt.title(r'Sample Histogram')

# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()

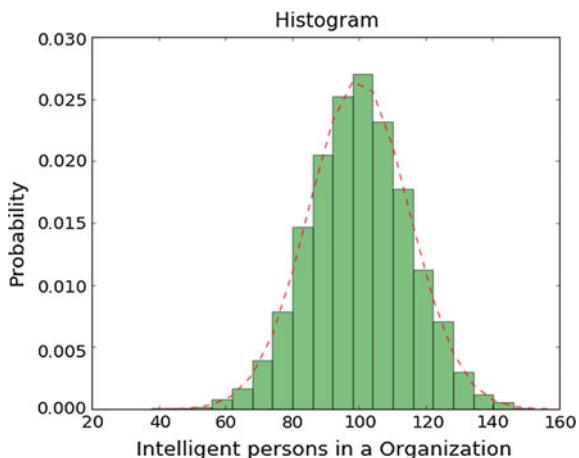
```

18.3.2 Distribution Chart with Intelligent Persons Dataset

A small example on the distribution chart on a dataset of people and their probability of carrying out intelligent work is as shown below. The code approach is as similar to the distribution charts discussed in the previous sections. The same module matplotlib and mlab modules are used for plotting the distribution chart.

The output of the plot is as shown in Fig. 18.10. It can be observed from the plot that both histogram and the line plot are combined into one. The line drawn over the histogram forms a uniform distribution over the histogram as shown in the plot. In this way, the distribution charts can be used for identifying the distribution under the curve.

Fig. 18.10 Distribution chart example dataset



```

import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as my_plt

#mean value
mean = 100
#standard deviation value
sd = 15
x = mean + sd * np.random.randn(10000)
num_bins = 20

# Histogram
n, bins, patches = my_plt.hist(x, num_bins, normed=1,
facecolor='green', alpha=0.5)

# add a 'best fit' line
y = mlab.normpdf(bins, mean, sd)
my_plt.plot(bins, y, 'r--')
my_plt.xlabel('Intelligent persons in a Organization')
my_plt.ylabel('Probability')
my_plt.title('Histogram')

# Adjusting the spacing
my_plt.subplots_adjust(left=0.15)
my_plt.show()

```

18.4 Relationship Charts

In machine learning techniques discussed in the previous section, the relation between the variables needs to be identified. Relationship charts are helpful in this aspect. The main aim of the relationship charts is to see the underlying relations among the variables and to determine which are dependent and independent variables. The relationship charts can be categorized into the following types:

- Scatter plot relationship charts,
- Bubble plot relationship charts, and
- Scatter group relationship charts.

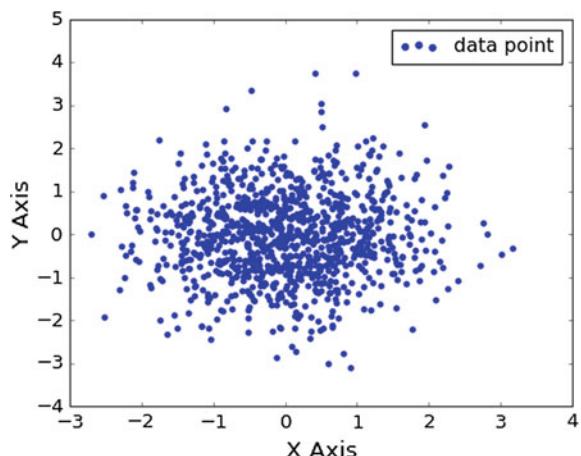
Each of these types of charts can be better understood with an example. In the upcoming sections, relationship charts for each of the categories are explained.

18.4.1 Relationship Chart with Scatter Plot

An example on the relationship chart with scatter plot is discussed in this section. The module matplotlib and the plot() function are used for the visualization. The function scatter() is used for scatter plot that depicts the relationship among the variables. The following code demonstrates the usage of scatter plot for relationship charts.

In this example, x and y are data that initialized randomly using rand() function and 1000 samples. The parameters x and y are passed to the scatter() function to plot the graph as shown in Fig. 18.11. The label for the plot is created with option ‘label’, and the axes are named with xlabel() and ylabel() functions. Optionally, the legend can be added by using the function legend(). It is a recommended practice in scatter plot to use the legends as many variables will be present in the scatter plot. The other types of the scatter plot are discussed in the next section.

Fig. 18.11 Relationship chart with scatter plot example



```

import matplotlib.pyplot as plt
import numpy as np

x = np.random.randn(1000)
y = np.random.randn(1000)

plt.scatter(x,y,label="data point",color='blue')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.legend(loc="upper right")
plt.show()

```

18.4.2 Relationship Chart with Bubble Plot

In this section, a small example on the bubble plot is discussed. The following code demonstrates the usage of bubble plot relationship chart. Here, x, y, and c are initialized with 50 random sample data points. The scatter() function as discussed in the previous relationship chart is used. But the parameter ‘s’ that specifies the size of the bubble needs to be passed to the scatter() function.

In this example, the size of the bubble is multiplied with ‘c’. The output of the plot is as shown in Fig. 18.12. It can be observed in the plot that the size of the data point is increased that specifies that there are more data points that belong to ‘c’ in this example.

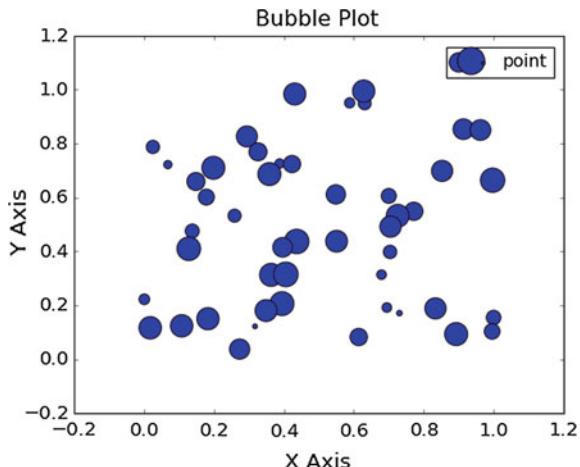
```

import matplotlib.pyplot as plt
import numpy as np

x = np.random.randn(1000)
y = np.random.randn(1000)

```

Fig. 18.12 Relationship chart with bubble plot example



```
plt.scatter(x,y,label="data point",color='blue')
plt.xlabel('X Axis')
plt.ylabel('Y Axis')
plt.legend(loc="upper right")
plt.show()
```

18.4.3 Relationship Chart Scatter Groups

The relationship charts that are discussed previously do not show any groups in the plot. In this section, a small example on the scatter groups is discussed. Here, x1, x2, and x3 variables are initialized with random values first. The same module matplotlib() is used with plot() function. A tuple ‘data’ is used to group x1, x2, and x3 inputs. A for loop is used to iterate over the data and group the three scattered data points.

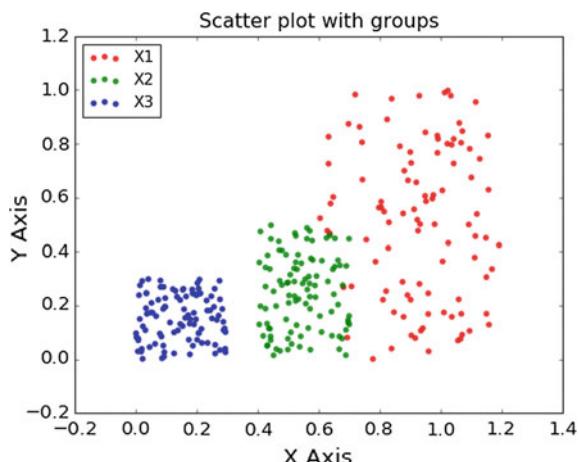
The output of the plot is as shown in Fig. 18.13. It can be observed from the plot that there are three groups in the scatter plot. From the plot, it can be easily seen that there are three groups in the data initialized. In this way, the scatter groups can be identified with the scatter plot function.

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

x1 = (0.6+0.6 * np.random.rand(100), np.random.rand(100))
x2 = (0.4+0.3 * np.random.rand(100), 0.5*np.random.rand(100))
x3 = (0.3*np.random.rand(100), 0.3*np.random.rand(100))

data = (x1, x2, x3)
colors = ("red", "green", "blue")
groups = ("X1", "X2", "X3")
```

Fig. 18.13 Relationship chart scatter groups example



```
# Create plot
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

for data, color, group in zip(data, colors, groups):
    x, y = data
    ax.scatter(x, y, alpha=0.8, c=color, edgecolors='none', s=30,
label=group)

plt.title('Scatter plot with groups')
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.legend(loc="upper left")
plt.show()
```

18.4.4 Relationship Chart for Population Analysis

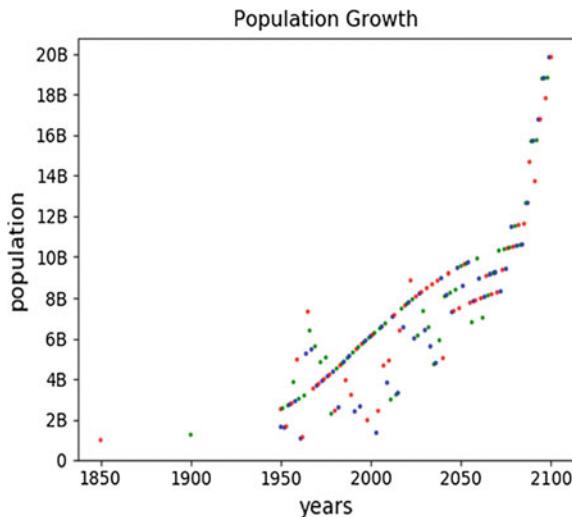
In this section, a relationship chart for a population dataset is visualized. The dataset considered here consists of population for a certain number of years. A scatter plot is used to visualize the relationship chart between the population and the years. Firstly, the modules required are matplotlib, numpy for visualization. The data are converted to a list ‘population_value’, and the range of years is initialized in ‘year_value’.

The scatter() function is used to visualize the population data with parameter s initialized with the area value. The area value implies the bubble value in the scatter plot. The output of the plot is as shown in Fig. 18.14 It can be observed from the plot that there has been a gradual increase in the population from 1950 to 2000 and is likely to increase. In this way, relationship charts can be used for visualization.

```
import matplotlib.pyplot as my_plt
import plotly.plotly as py
import numpy as np

bubbles_chart = my_plt.figure()
year_value = list(range(1950, 2101))
population_value =
[2.53,2.57,1.62,1.67,2.71,2.76,2.81,3.86,2.92,4.97,3.03,1.08,
1.14,3.2,5.26,7.33,6.4,5.47,3.54,5.62,3.69,3.77,4.84,3.92,4.,
5.07,4.15,4.22,2.3,4.37,2.45,4.53,2.61,4.69,4.78,4.86,3.95,5.
05,5.14,3.23,5.32,2.41,5.49,5.58,2.66,5.74,5.82,5.9,1.98,6.05
,6.13,6.2,6.28,1.36,2.44,6.51,6.59,4.67,6.75,3.83,4.92,3,7.08
,7.16,3.24,3.32,6.4,7.48,6.56,7.64,7.72,7.79,8.87,7.94,6.01,8
.08,6.15,8.22,8.29,7.36,6.42,8.49,6.56,5.62,8.68,4.74,4.8,8.8
```

Fig. 18.14 Relationship chart for population dataset



```

6,5.92,8.98,5.04,8.09,8.15,9.2,8.26,7.31,7.36,8.41,9.46,7.5,9
.55,8.6,9.64,9.68,9.73,7.77,6.81,7.85,7.88,9.92,8.96,7.99,7.0
3,8.06,9.09,8.13,9.16,8.19,9.22,9.25,8.28,10.31,8.33,9.36,10.
38,9.41,10.43,10.46,11.48,10.5,11.52,10.55,11.57,10.59,10.61,
11.63,12.65,12.66,14.68,15.7,15.72,13.73,15.75,16.77,16.78,18
.79,18.81,17.82,18.83,19.84,19.85]
population_value = [1,1.262,1.650]+population_value
year_value = [1850,1900,1950]+year_value

colors = ("red", "green", "blue")
area = np.pi*1

# Scatter Plot
my_plt.scatter(year_value,      population_value,      s=area,      c=colors,
alpha=0.9)

my_plt.xlabel('years')
my_plt.ylabel('population')
my_plt.title('Population Growth')
my_plt.yticks([0,2,4,6,8,10,12,14,16,18,20],
[‘0’,’2B’,’4B’,’6B’,’8B’,’10B’,’12B’,’14B’,’16B’,’18B’,’20B’])
my_plt.show()

```

18.5 Exercises

1. To analyze the sales of particular product, what kind of graph should be used?
2. Which are the useful charts for visualization? Justify.
3. Compare relationship charts and composition charts with an example.
4. Create a relationship chart with multiple lines for any random dataset.
5. Implement a composition chart for the following data.

Courses	Number of students enrolled
Computer science	125
Electrical science	87
Electronics and computers	75
Telecommunications	85
Information technology	65
Cyber-security	55
Mobile computing	65

6. Implement the stacked chart for the following data.

Number	Batch 1	Batch 2
1	25	35
2	37	32
3	45	21
4	65	25
5	35	67
6	25	34
7	55	54

7. Implement a Gaussian distribution chart in Python. What is the difference between normal distribution and Gaussian distribution?
8. How do you change the size of a plot in matplotlib? Increase the sizes of the plots obtained in questions 5 and 6.

References

1. Nelli, F. (2015). Data visualization with matplotlib. In *Python data analytics* (pp. 167-235). Berkeley, CA: Apress
2. Langtangen, H. P., Langtangen, H. P. (2009). *A primer on scientific programming with Python* (Vol 2). Berlin: Springer

3. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science Engineering*, 9(3), 90–95
4. Yu, A., Chung, C., & Yim, A. (2017). Matplotlib 2. x by example: Multi-dimensional charts, graphs, and plots in Python
5. Vaingast, S. (2014). Graphs and plots. In *Beginning Python visualization* (pp. 189–232). Berkeley, CA: Apress

Chapter 19

Advanced Visualization



Advanced visualizations are needed for data analytics for larger datasets. These visualizations are used for machine learning techniques like regression, clustering, support vector machines. In the previous section, only 2D plot examples on data visualization were discussed. There are certain situations where additional information can be gathered from the visual plots using advanced features. The advanced visualization consists of box plots, contour plots, pyramid charts, and others. In this section, some of the advanced visualization techniques are discussed with examples.

19.1 3D Bar Plot

3D bar plots are used for the analysis of the data that exists in three dimensions. In this section, a small example on the visualization of 3D bar plot is discussed. The main difference between the 2D plot and 3D plot is if there are three features in the dataset, then the plot() function discussed in the previous section on bar plot cannot be used. An example on 3D plot is as shown below. The main module required for 3D visualization is mplot3D with the function Axes3D [1].

In this example, xpos and ypos are the data points that are plotted in 2D plane. For 3D viewing, the dimension dz is added. The output of the plot is as shown in Fig. 19.1. It can be clearly observed in the plot that the bars cover all the three axes specifying the dimensions covered by the data points. 3D bar plots can be mainly used in the cases where the building height, width, and dept have to be visualized.

```
# 3D bar plot

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
```

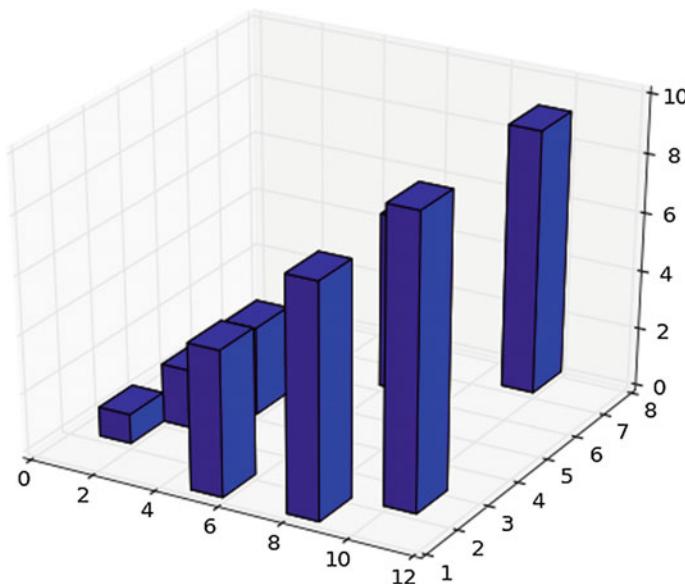


Fig. 19.1 3D bar plot example

```
fig = plt.figure()
ax1 = fig.add_subplot(111, projection='3d')

xpos = [1,2,3,4,5,6,7,8,9,10]
ypos = [2,3,4,5,1,6,2,1,7,2]
num_elements = len(xpos)
zpos = [0,0,0,0,0,0,0,0,0,0]
dx = np.ones(10)
dy = np.ones(10)
dz = [1,2,3,4,5,6,7,8,9,10]

ax1.bar3d(xpos, ypos, zpos, dx, dy, dz, color='blue')
plt.show()
```

The following code demonstrates the alternative way of visualizing the 3D plot in bar graph form. The difference between this and the earlier code is the parameter ‘zdir.’ The parameter zdir specifies the direction in which the graph needs to be aligned. The output of the plot is as shown in Fig. 19.2.

```
# 3D Barplot
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
```

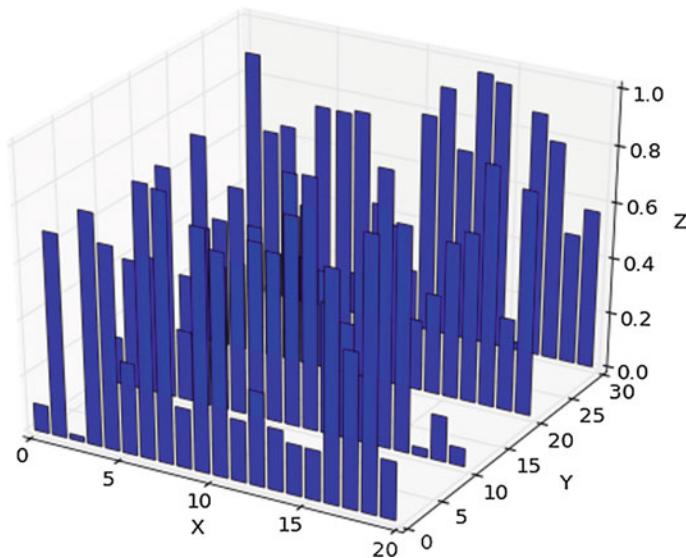


Fig. 19.2 3D bar plot example with zdir

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for c, z in zip(['r', 'g', 'b', 'y'], [30, 20, 10, 0]):
    xs = np.arange(20)
    ys = np.random.rand(20)
    ax.bar(xs, ys, zs=z, zdir='y', alpha=0.8)

ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

plt.show()
```

19.2 3D Histogram

Histogram analysis helps in getting the results pertaining to a particular data such as year, month, and days. The following code demonstrates the usage of 3D histogram analysis. Initially, the data x and y are initialized with random values. The function histogram2d is used first to create the histogram for the visualization. To build the 3D axes, bar3d() function is used with xpos, ypos, zpos, dx, dy, and dz as the parameters. The output of the plot is as shown in Fig. 19.3.

```
# 3D Histogram
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
```

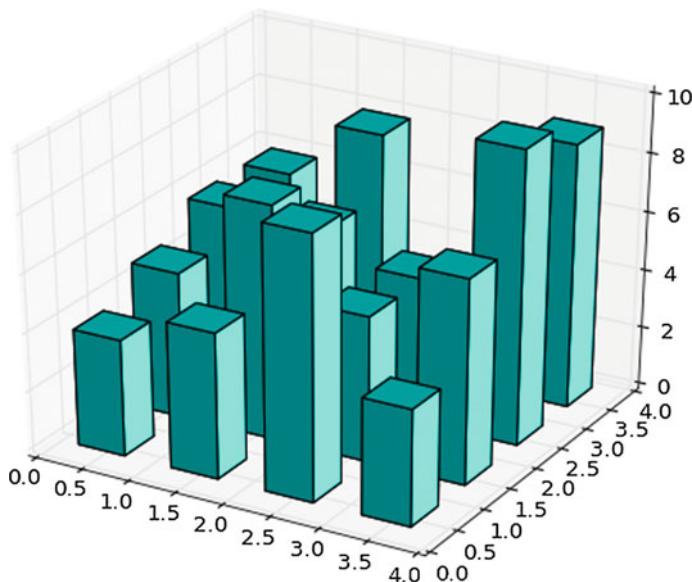


Fig. 19.3 3D histogram plot example

```

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x, y = np.random.rand(2, 100) * 4
hist, xedges, yedges = np.histogram2d(x, y, bins=4,
range=[[0, 4], [0, 4]])
xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25)
xpos = xpos.flatten('F')
ypos = ypos.flatten('F')
zpos = np.zeros_like(xpos)

dx = 0.5 * np.ones_like(zpos)
dy = dx.copy()
dz = hist.flatten()

ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='cyan',
zsort='average')

plt.show()

```

19.3 3D Contour Plot

A contour plot specifies the three-dimensional surface of the z -axis slice of the data [1, 2]. For example, in the following code a random data of X , Y , and Z are created with the function `data()`. The `contour` function is used to plot the graph as shown in

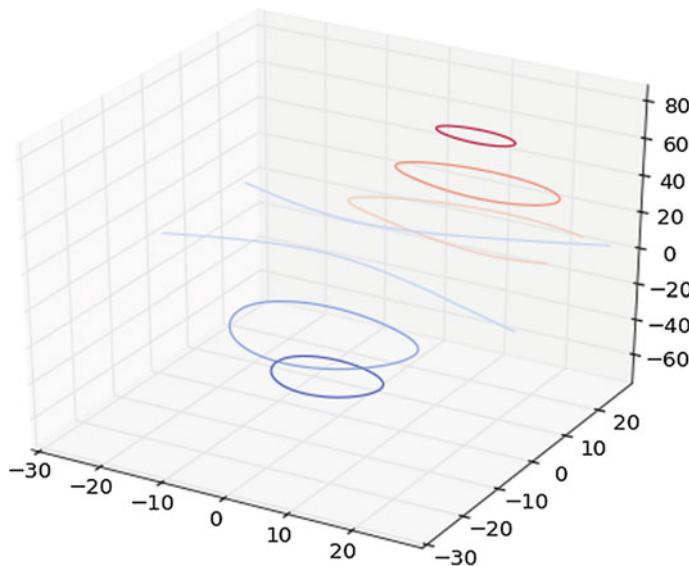


Fig. 19.4 3D contour plot example

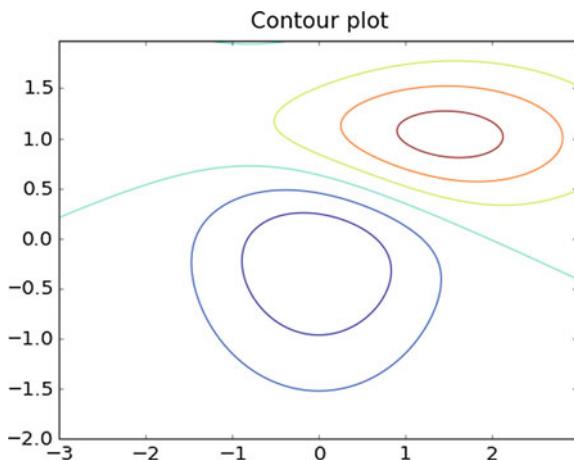
Fig. 19.4. In the plot, it can be observed that two circle surfaces are aligned along the y -axis and there are three circles that are aligned along the z -axis. In this way, contour plots can be used to distinguish easily between the surfaces of the data.

```
# 3D Contour plot
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
cset = ax.contour(X, Y, Z, cmap=cm.coolwarm)
ax.clabel(cset, fontsize=9, inline=1)

plt.show()
```

The following code demonstrates 2D contour plot as shown in Fig. 19.5. Random data are initialized with x and y values. The mlab() function is used to derive the axes direction along the z -axis with $z1$ and $z2$ as the variables. Finally, the contour plot is drawn using the function contour() and plot() function. In the plot, it can be observed that the surface area of the slice of the contours is divided along the axis value as 1.0.

Fig. 19.5 Contour plot example



```
#contour plot
import matplotlib
import numpy as np
import matplotlib.cm as cm
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

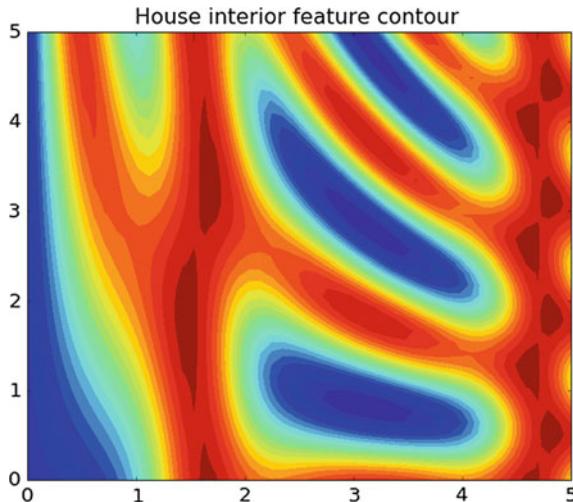
delta = 0.025
x = np.arange(-3.0, 3.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = mlab.bivariate_normal(X, Y, 1.0, 1.0, 0.0, 0.0)
Z2 = mlab.bivariate_normal(X, Y, 1.5, 0.5, 1, 1)
Z = 10.0 * (Z2 - Z1)
plt.title('Contour plot')
CS = plt.contour(X, Y, Z)
plt.show()
```

19.3.1 Contour Plot for Housing Data

In the following code, an example of housing dataset and contour plot is discussed. From the dataset, the columns ‘price_doc’, ‘full_sq’, ‘life_sq’, ‘floor’, ‘max_floor’, ‘state’, ‘kitch_sq’, ‘num_room’ are extracted for the box plot. The main aim of the contour plot is to depict the price of house and full square feet of the house.

Initially, the data are read from the csv file using pandas library and converted into a data frame. The data frame is created with selected columns from the dataset. The contourf() function is used to draw the graph as shown in Fig. 19.6. It can be observed from the plot that floor_size and the price of the house form different

Fig. 19.6 Contour plot for housing data



slices of the contour plot. In the plot, it is also clearly shown that as the full_sq increases, then the price_doc of the house also increases. In this way, the contour plots can be used for visualization.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
def f(x, y):
    return np.sin(x) ** 10 + np.cos(10 + y * x) * np.cos(x)
df=pd.read_csv("train.csv")
df_indoor=df[['price_doc','full_sq','life_sq','floor','max_floor','state',
'kitch_sq','num_room']]
df_indoor=df[df.max_floor<20]
plt.figure(figsize=(10,8))
x=df_indoor.price_doc
y=df_indoor.full_sq
x = np.linspace(0, 5, 50)
y = np.linspace(0, 5, 40)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
plt.contourf(X, Y, Z, 20, color='black')
plt.title('House interior feature contour')
plt.show()
```

19.4 3D Scatter Plot

Scatter plot is used to identify the relationship among the variables in the data or to identify the different locations of the data. The main aim of the scatter plot is to show how one variable will affect the other. The correlation among the data is gathered based on the scatter plot [4, 5]. Usually, the following points are observed for the correlation analysis:

- If the value of y -axis increases with the value of x -axis, then the correlation is positive.
- If the value of y -axis decreases with the value of x -axis, then the correlation is negative.
- If neither of the criteria is satisfied, then the correlation is zero.

The module matplotlib with the function scatter() is used for plotting the graph. Initially, xs, ys, and zs are initialized with random values, and these parameters are passed to the scatter function. The randrange() function is used to initialize the random values. The output of the plot is as shown in Fig. 19.7. It can be observed from the plot that most of the data points lie along the z-label. It means that there is a correlation among the z-label locations and x-label and y-label locations.

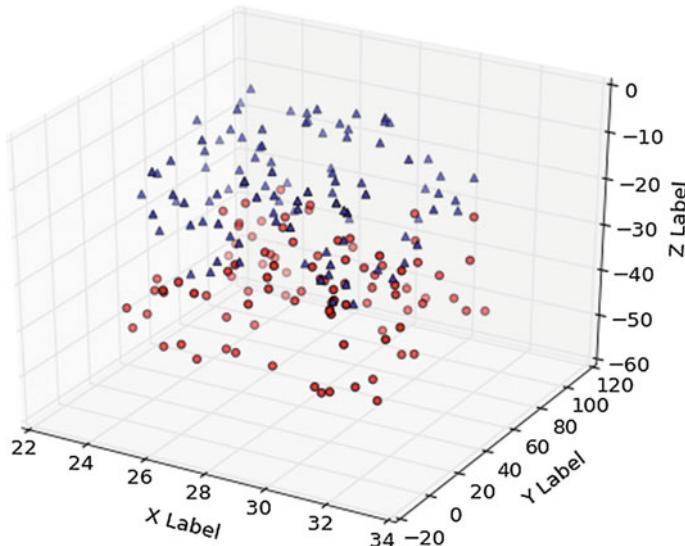


Fig. 19.7 3D scatter plot example

```

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(83232)
def randrange(n, vmin, vmax):
    return (vmax - vmin)*np.random.rand(n) + vmin

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
n = 100
for c, m, zlow, zhight in [('r', 'o', -50, -25), ('b', '^', -30, -5)]:
    xs = randrange(n, 23, 32)
    ys = randrange(n, 0, 100)
    zs = randrange(n, zlow, zhight)
    ax.scatter(xs, ys, zs, c=c, marker=m)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()

```

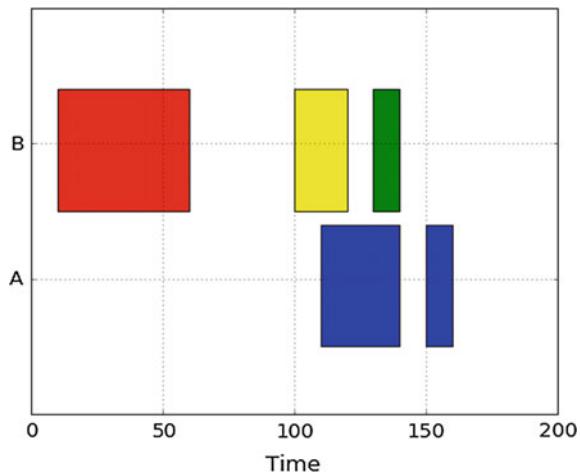
19.5 Gantt Plot

A gantt plot is used to depict the timeline of different activities typically carried out in a project [2, 3]. The following code demonstrates the gantt plot with plt() as the function. For this example, the activities considered are as shown in Table 19.1 with start and end times.

Initially, all the activities with start time and end time are initialized with broken_barh() function. The plot is first arranged as a grid so that easily the timelines can be arranged. Then, the show() method is used to plot the graph as shown in Fig. 19.8. It can be observed from the plot that the timelines can be easily identified from the gantt chart. In this way, the gantt chart is used for displaying the timelines of activities.

Table 19.1 Gantt plot example

Activity	Start	End
1	10	60
2	100	120
3	125	140
4	105	140
5	150	155

Fig. 19.8 Gantt plot example

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.broken_barh([(110, 30), (150, 10)], (10, 9), facecolors='blue')
ax.broken_barh([(10, 50), (100, 20), (130, 10)], (20, 9),
               facecolors=('red', 'yellow', 'green'))
ax.set_xlim(0, 200)
ax.set_xlabel('Time')
ax.set_xticks([15, 25])
ax.set_yticklabels(['A', 'B'])
ax.grid(True)

plt.show()
```

19.6 Box–Whisker Plot

A box–whisker plot determines the distribution of a variable with five numbers, namely minimum, lower quartile, median, upper quartile, and maximum. If any outliers are there in the plot, it is also displayed in the box plot. The detail of the box plot is as shown in Fig. 19.9. The dotted line represents the whisker line that ranges from the minimum to maximum.

The following code demonstrates the usage of box plot, and the output is as shown in Fig. 19.10. A random data are initialized first and concatenated to the variable d2. It is then appended to ‘data’ variable that is used for drawing the graph of box plot. The boxplot() function is used to draw the box plot as shown in the plot. It can be observed from the plot that the lower quartile value is negative, while the mean lies somewhere in the value of 20–30 and the upper quartile value is positive.

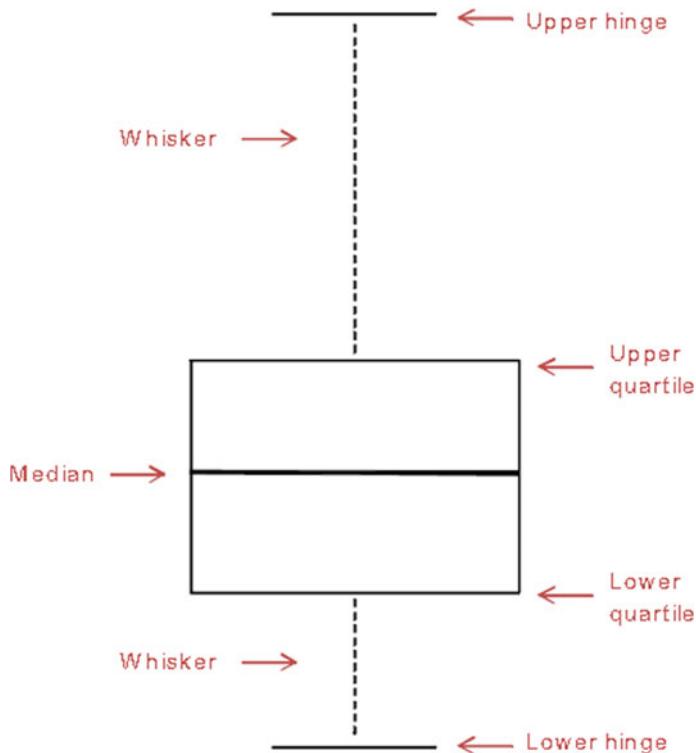
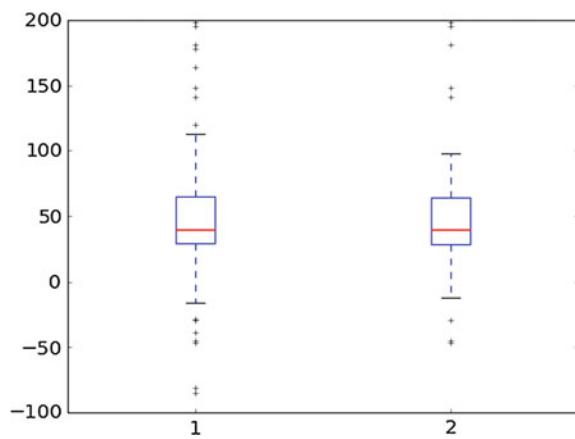


Fig. 19.9 Box plot details

Fig. 19.10 Box plot example



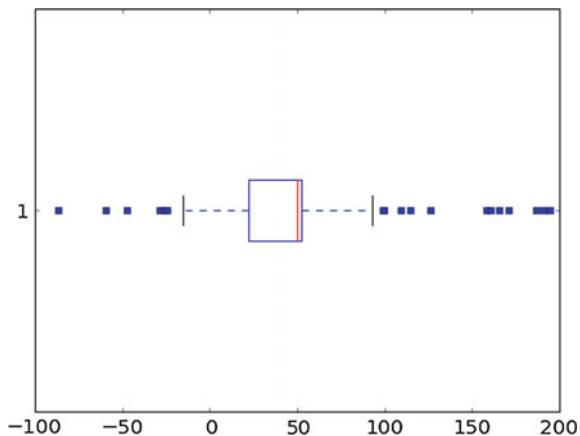
```
#Multiple box plots
import matplotlib.pyplot as plt
import numpy as np

# Random data
data = np.random.rand(1000)
spread = np.random.rand(50) * 100
center = np.ones(25) * 40
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
d2 = np.concatenate((spread, center, flier_high, flier_low), 0)
data.shape = (-1, 1)
d2.shape = (-1, 1)
data = [d2, d2[:, ::2], 0]

#Plot
plt.figure()
plt.boxplot(data)
plt.show()
```

The following code demonstrates the single box plot where the plot is drawn horizontally as shown in Fig. 19.11. The data are initialized in the same way as previously, but here, single data are used. The same boxplot() function is used to display the boxplot as shown in the plot. In the plot, it can be clearly observed that the box plot is aligned horizontally. The lower quartile is negative, and the higher quartile is more than 100 with the mean value 50.

Fig. 19.11 Single box plot example



```
#Single box plots
import matplotlib.pyplot as plt
import numpy as np

# Random data #
spread = np.random.rand(50) * 100
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data = np.concatenate((spread, center, flier_high, flier_low), 0)
#               #

plt.boxplot(data) # Vertical box plot
plt.figure()
plt.boxplot(data, 0, 'rs', 0) #Horizontal box plot
plt.show()
```

19.6.1 Box Plot on Housing Data

The following code demonstrates the example of a box plot on a housing dataset. From the dataset, the columns ‘price_doc’, ‘full_sq’, ‘life_sq’, ‘floor’, ‘max_floor’, ‘state’, ‘kitch_sq’, ‘num_room’ are extracted for the box plot. The main aim of the box plot is to depict the number of rooms and the price for that house depending on the maximum floor size.

Initially, the data are read from the csv file using pandas library and converted into a data frame. The data frame is created with selected columns from the dataset.

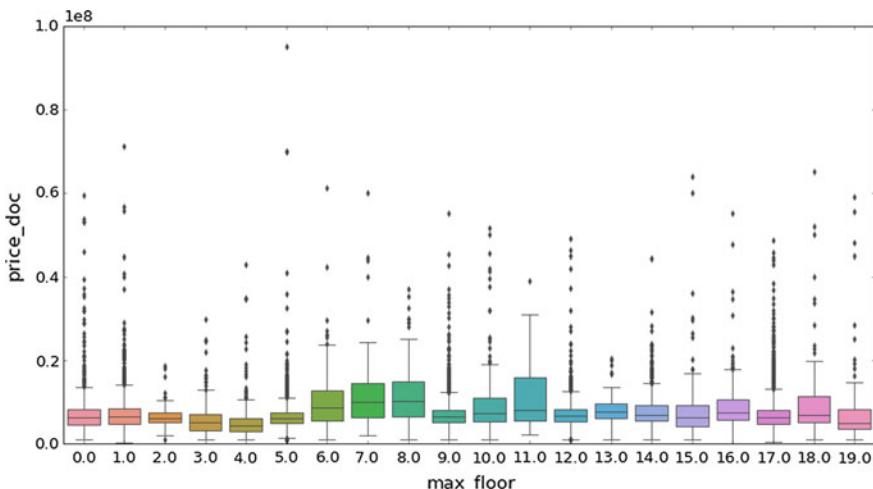


Fig. 19.12 Box plot housing data

The boxplot() function is used to draw the graph as shown in Fig. 19.12 with x -axis as the ‘max_floor’ and y -axis as the ‘price_doc.’ It can be observed from the plot that when the max_floor size is increased to 7 or 8, the mean price of the house is increased, whereas the mean price of the house for max_floor 1–3 are lesser compared to others. The lower quartile values are in the value of 0, and the higher quartile values are above 0.2. It can be seen that as the max_floor increases, the higher quartile value also increases.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
df=pd.read_csv("train.csv")
df_indoor=df[['price_doc','full_sq','life_sq','floor',
'max_floor','state','kitch_sq','num_room']]
df_indoor=df[df.max_floor<20]
plt.figure(figsize=(16,8))
sns.boxplot(x="max_floor", y="price_doc", data=df_indoor)
plt.show()
```

19.7 Polar Plot

A polar plot uses polar coordinates to draw the graph. In the polar plot, a radius function is used for the angle of projection [3]. It is circular in shape with equal intervals of degrees as shown in Fig. 19.13. The function plot() in the matplotlib can be used for drawing the graph with two parameters theta and data. The following code demonstrates the usage of polar plot in Python.

Here, theta is initialized with the circumference of the circle ($2\pi r$), and r is used as the data for the polar plot. The output of the plot is as shown in Fig. 19.13. The blue line indicates the occupancy of the data along the radius of the polar plot. The equal intervals of data from 0.5 to 2.0 are split along the radius of the plot.

```
#Polar plot
import numpy as np
import matplotlib.pyplot as plt

r = np.arange(0, 5, 0.01)
theta = 2*np.pi*r

ax = plt.subplot(111, projection='polar')
```

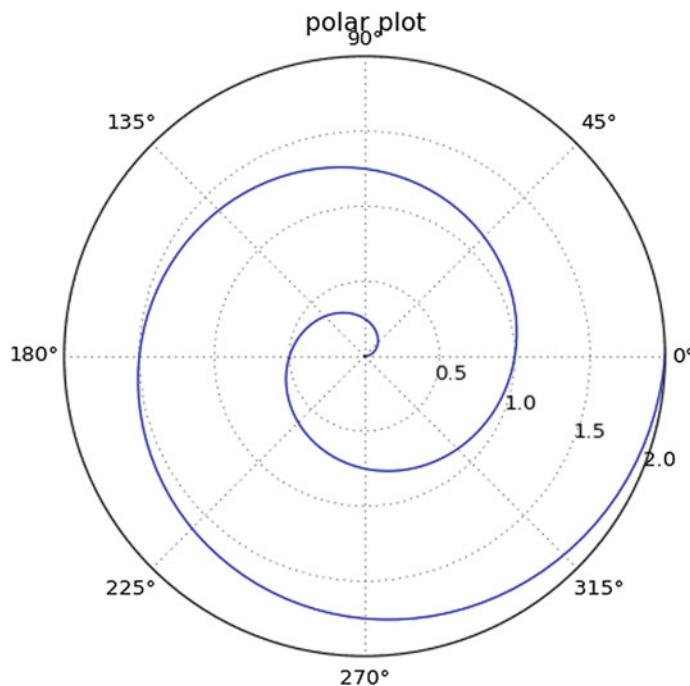


Fig. 19.13 Polar plot example

```

ax.plot(theta, r)
ax.set_rmax(2)
ax.set_rticks([0.5, 1, 1.5, 2]) #Radial Ticks
ax.set_rlabel_position(-22.5)
ax.grid(True)

ax.set_title("polar plot")
plt.show()

```

19.8 Dendrograms

Dendrogram plots are used to draw a tree-based plot that identifies the taxonomical classifications. It can be used to deduce the hierarchy of the nodes in data [2, 6]. In this section, a small example is discussed for the dendrogram plot. The dataset considered for the dendrogram plot consists of rows for a random game where there are two types with type 1 as grass and type 2 as poison. Depending on these types, the dendrogram plot specifies whether attack, defense, special attack need to be taken.

The following code demonstrates the dendrogram plot usage in Python. Initially, the data are read from the csv file using pandas library. The `augmented_dendrogram()`

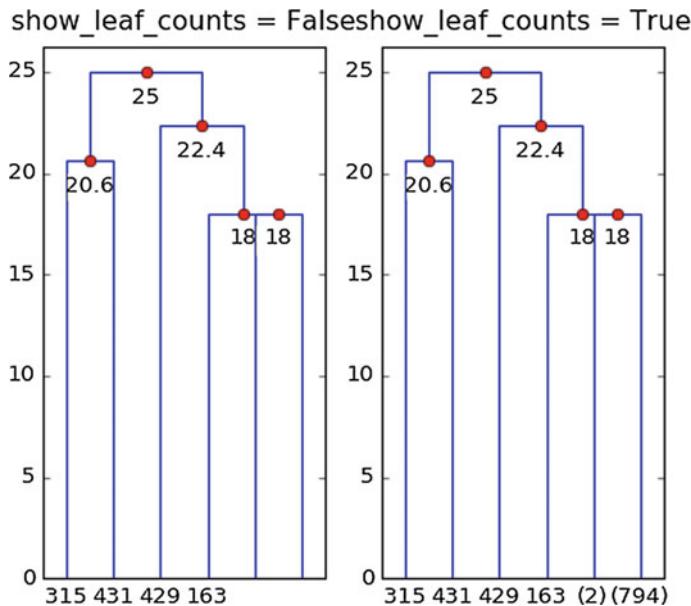


Fig. 19.14 Dendrogram plot example

function is used to prepare the plot with linkage matrix. The linkage matrix specifies the data that need to be projected in the dendrogram plot. Inside augmented_dendrogram() function, the plot function is used to plot the graph as shown in Fig. 19.14.

The plot shows the leaf counts on the left side with false, and the right part shows the leaf counts with true condition. The upper part begins with the value 25 and descends down to 18. The left part depicts the Type1 attack, and the right part depicts the Type2 attack. In this way, the dendrograms can be used for visualization.

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
from scipy.cluster.hierarchy import dendrogram
from scipy.cluster.hierarchy import linkage
def augmented_dendrogram(*args, **kwargs):
    ddata = dendrogram(*args, **kwargs)
    if not kwargs.get('no_plot', False):
        for i, d in zip(ddata['icoord'], ddata['dcoord']):
            x = 0.5 * sum(i[1:3])
            y = d[1]
            plt.plot(x, y, 'ro')
```

```
plt.annotate("%.3g" % y, (x, y), xytext=(0, -8),
            textcoords='offset points',
            va='top', ha='center')

return ddata

df=pd.read_csv("mon.csv")
df=df[['Attack','Speed']]
x=df.values
plt.figure(1, figsize=(6, 5))
plt.clf()
plt.scatter(x[:, 0], x[:, 1])
plt.axis('equal')
plt.grid(True)

linkage_matrix = linkage(x, "single")

plt.subplot(1, 2, 1)
show_leaf_counts = False
ddata = augmented_dendrogram(linkage_matrix,
                               color_threshold=1,
                               p=6,
                               truncate_mode='lastp',
                               show_leaf_counts=show_leaf_counts,
                               )
plt.title("show_leaf_counts = %s" % show_leaf_counts)

plt.subplot(1, 2, 2)
show_leaf_counts = True
ddata = augmented_dendrogram(linkage_matrix,
                               color_threshold=1,
                               p=6,
                               truncate_mode='lastp',
                               show_leaf_counts=show_leaf_counts,
                               )
plt.title("show_leaf_counts = %s" % show_leaf_counts)

plt.show()
```

19.9 Heatmaps

Heatmap charts are used to depict the data in colors depending on the values associated with them. Each value in the data is associated with a color on the heatmap. A range of values are used to create a color for the heatmap initially first, and these colors are used for the plot [2, 6]. The usefulness of the heatmap is some of the values where small

differences are easily categorized based on the colors. It is used in the applications related to weather, pollution, stock market pricing, and energy.

An example on the housing data considered for the box plots earlier in this chapter is used for the heatmaps as well. The function `heatmap()` is used for drawing the heatmap as shown in Fig. 19.15. The same set of columns as in the box plot is used for the heatmap as well. The output of the plot is as shown in Fig. 19.15. It can be observed from the plot that certain colors are used for the values in the data ranging from 0.0 to 1.0. All the diagonal elements are initialized to 1 automatically in the heatmap. For the rest of the values, the heatmap is considered for plotting the values. In this way, the values can be easily differentiated using the heatmap rather than the earlier plots like box plot and scatter plot.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
```



Fig. 19.15 Heatmap plot example

```

import warnings
warnings.filterwarnings('ignore')
df=pd.read_csv("train.csv")

#Heatmap of Indoor Characteristics
df_indoor=df[['price_doc','full_sq','life_sq','floor','max_floor','state',
'kitch_sq','num_room']]
corrmat = df_indoor.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat,cbar=True, annot=True, square=True);
df_indoor.fillna(method='bfill',inplace=True)
plt.show()

```

19.10 Pyramid Charts

The charts that are discussed for visualization earlier show the relation or the plot for two/three variables in the dataset. However, in certain scenarios, charts are needed for comparison of two variables in side-by-side manner so that it is easy to compare and distinguish the data values [3, 6]. Pyramid charts help in these scenarios that allow side-by-side plot and allows comparison of values.

One of the examples considered for pyramid charts in this section is a set of countries with number of staff and sales associated with each country. Initially, a set of 20 states and for each state, the number of staff and sales are initialized using a list in Python. The list of the staff are sorted and aggregated to the respective states. The subplots function is used to plot the graphs separately using two axes, where axes[0] is used for number of staff and the country and axes[1] is used for sales and the country. These two axes are used to plot the graph as shown in Fig. 19.16 by iterating over one another. Now, from the plot, we can clearly identify the number of staff in each country and the sales associated with each country. The comparison can also be done for each country based on the number of staff and sales. In this way, the pyramid charts can be used for side-by-side comparison of each of the variables in the data.

```

import numpy as np
import matplotlib.pyplot as plt

# Data
states = ["AK", "TX", "CA", "MT", "NM", "AZ", "NV", "CO", "OR", "WY", "MI",
          "MN", "UT", "ID", "KS", "NE", "SD", "WA", "ND", "OK"]
staff = np.array([20, 30, 40, 10, 15, 35, 18, 25, 22, 7, 12, 22, 3, 4, 5, 8,
                 14, 28, 24, 32])
sales = staff * (20 + 10 * np.random.random(staff.size))

```

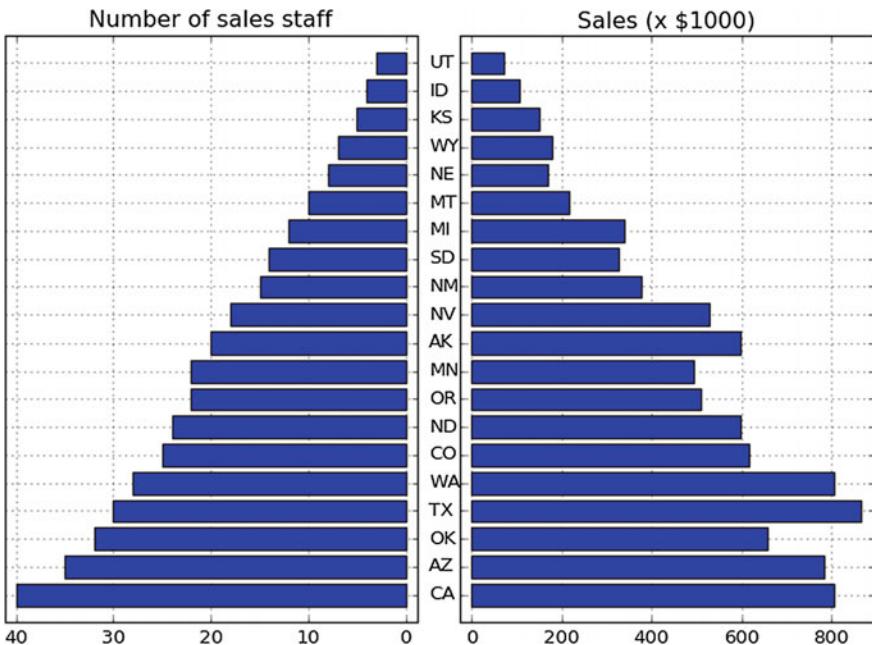


Fig. 19.16 Pyramid charts plot example

```
# Sort by number of sales staff
idx = staff.argsort()
idx = idx[::-1]
states, staff, sales = [np.take(x, idx) for x in [states, staff, sales]]

y = np.arange(sales.size)

fig, axes = plt.subplots(ncols=2, sharey=True)
axes[0].barh(y, staff, align='center', color='blue', zorder=10)
axes[0].set(title='Number of sales staff')
axes[1].barh(y, sales, align='center', color='blue', zorder=10)
axes[1].set(title='Sales (x $1000)')
axes[0].invert_xaxis()
axes[0].set(yticks=y, yticklabels=states)
axes[0].yaxis.tick_right()

for ax in axes.flat:
    ax.margins(0.03)
    ax.grid(True)
```

```
fig.tight_layout()  
fig.subplots_adjust(wspace=0.09)  
plt.show()
```

19.11 Radar Plots

A radar chart is used to display the multivariate data in the form of 2D plot. It is also known as spider chart or Web chart where the axes are aligned to a common point first. It can be understood with the help of dataset easily. The dataset considered for the example here is same as the dendrogram plot. Initially, the column values required for data analysis are extracted first and the radius of the plot is initialized. The following code demonstrates the usage of radar plots in Python.

The plot() function with parameter ‘polar=True’ is used to plot the radar chart. The ‘stats’ and ‘angles’ are used for the projections of the plot. The output of the plot is as shown in Fig. 19.17. It can be observed from the plot that a common angle of the plot with range of values from 10 to 80 is aligned along the common axis. The variables ‘Attack,’ ‘Defense,’ ‘Special attack,’ ‘Special defense,’ and ‘speed’ are aligned to the range of values in the radar plot. The radar in the plot is marked in the form of Web. Any data point inside this Web can be used to see which action needs to be taken. In this way, the radar plots can be used for identifying the necessary actions with data values.

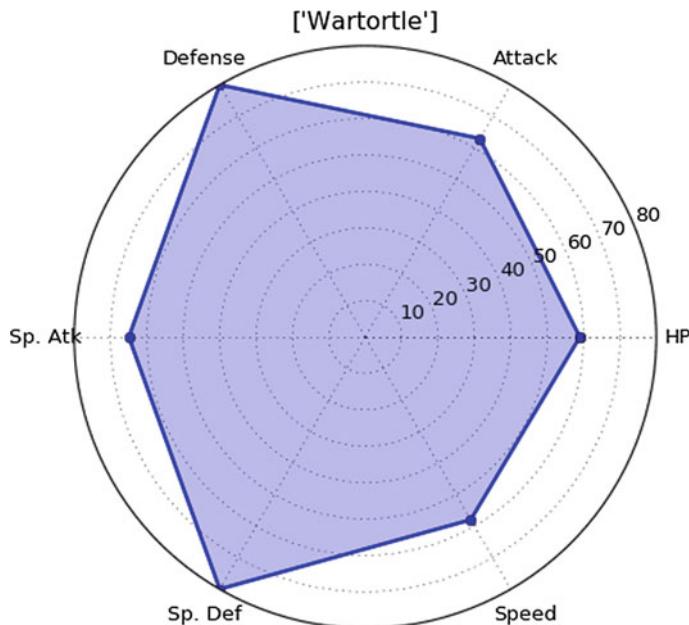


Fig. 19.17 Radar plot example

```

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
df=pd.read_csv("mon.csv")
df.head()
labels=np.array(['HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed'])
stats=df.loc[10,labels].values
angles=np.linspace(0, 2*np.pi, len(labels), endpoint=False)
stats=np.concatenate((stats,[stats[0]]))
angles=np.concatenate((angles,[angles[0]]))
fig=plt.figure()
ax = fig.add_subplot(111, polar=True)
ax.plot(angles, stats, 'o', linewidth=2)
ax.fill(angles, stats, alpha=0.25)
ax.set_thetagrids(angles * 180/np.pi, labels)
ax.set_title([df.loc[10,"Name"]])
ax.grid(True)
plt.show()

```

19.12 Exercises

1. Implement a Python program that draws two histograms together.
2. What is the significance of bins in histogram charts? Change the number of bins in the histogram in the program considered in this chapter.
3. What is the difference between 2D and 3D plots? Where are 3D plots mainly used?
4. What is the use of scatter plots? Which plot determines the distribution of a variable with numbers?
5. Create a box plot for the data [1, 2, 3, 4, 5, 6, 7, 100, 101, 102] and determine the fliers in the box plot.
6. Create a contour plot for the iris data in Python.
7. Create a heatmap for the following data by taking the days on the x -axis and the sessions on the y -axis.

```

values=[[1, 20, 30, 50, 1], [20, 1, 60, 80, 30],
[30, 60, 1, -10, 20]],
days=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'],
sessions=['Morning', 'Afternoon', 'Evening']

```

8. Create a radar plot for the following data that show the different groups one by one.

```
'group': ['A', 'B', 'C', 'D'],  
'var1': [38, 1.5, 30, 4],
```

9.

```
'var2': [29, 10, 9, 34],  
'var3': [8, 39, 23, 24],
```

10.

```
'var4': [7, 31, 33, 14],  
'var5': [28, 15, 32, 14]
```

11. Create a vertical histogram chart for the iris dataset with sepal length as the feature.

12. Create a scatter plot for the following data.

Variable 1	Variable 2	Group
1.3	2.1	A
3.4	5.3	B
2.3	3.1	B
3.5	8.4	A
2.2	8.8	B

References

1. Johansson, R. (2015). Plotting and visualization. In *Numerical python* (pp. 89–123). Berkeley, CA: Apress.
2. Langtangen, H. P., & Langtangen, H. P. (2009). *A primer on scientific programming with Python* (Vol. 2). Berlin: Springer.
3. Barrett, P., Hunter, J., Miller, J. T., Hsu, J. C., & Greenfield, P. (2005, December). matplotlib—A portable python plotting package. In *Astronomical data analysis software and systems XIV* (Vol. 347, p. 91).
4. Foreman-Mackey, D. (2016). corner. py: Scatterplot matrices in Python. *The Journal of Open Source Software*, 2016.
5. <https://plot.ly/python/line-and-scatter/>.
6. <http://www.scipy-lectures.org/intro/matplotlib/matplotlib.html>.

Appendix

Installing Python

Python supports cross platform installation and working such as Windows, Ubuntu and Mac OS. In this section, the steps for the installation of Python in Ubuntu are listed as follows.

- Update the Ubuntu using the command `sudo-apt get update`
- Install the Python using the command `sudo-apt get install Python3.6`
- Now using the command `Python` the interpreter can be started.
- If already a version of Python is available then using the following command, the Python configurations can be switched.
`sudo update-alternatives—config Python3`

A.1 Installing Anaconda

For machine learning in Python, the default interpreter has limited modules for programming. Hence, Anaconda interpreter needs to be configured for machine learning in Python. In this section, the steps for installing Anaconda in Ubuntu are listed as follows.

- Download the anaconda shell script file from the Anaconda website, <https://www.anaconda.com/download/>.
- Run the shell script file, in the end add the Anaconda to the path variable.
- Run the command `conda` to verify the successful installation of Anaconda interpreter.

A.2 Installing NLTK

For text analytics, sentimental analysis the text content can be pre-processed in Python with the help of NLTK module. In this section, the steps for installing the NLTK in Ubuntu are listed as follows.

- Install nltk using the command `sudo pip install nltk`.
- Import the module nltk to verify the installation of nltk.

A.3 Python Basics

Python is one of the popular languages that are available for programming. In this section, the basics of Python programming is discussed with examples. The main concepts discussed with Python are lists, disctionary, sets and tuples. These basic concepts are necessary for getting started with machine learning in Python.

Lists

A list is a basic structure in Python that stores a sequence of elements. Anything that is enclosed with [] is called a list.

An example on creating a list is as shown below.

```
#Initialize a list
a=[1,2,3]
b=['one','two','three']
c=[1,'one','two',3]

#Print a list
print(a)
print(b)
print(c)

#Print the length of the list
#Use len() function

# Create a new list of names
names=[]
num=int(input('Enter the number of elements'))
for i in range(num):
    ele=input('Enter the element')
    names.append(ele)
print(names)
```

Output:

```
[1, 2, 3]
['one', 'two', 'three']
[1, 'one', 'two', 3]
```

```
Enter the number of elements3
Enter the element ramesh
Enter the element ranjan
Enter the element anjali
['ramesh', 'ranjan', 'anjali']
```

List slicing

A list of elements can be extracted using the list slicing as shown below.

```
#This program demonstrates list slicing
```

```
a_list=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

print('Initial list is ')
print(a_list)

print('List slicing for [1:4]')
print(a_list[1:4])

print('List slicing for [3:]')
print(a_list[3:])

print('List slicing for [:5]')
print(a_list[:5])

print('List slicing for [:]')
print(a_list[:])
```

```
#List slicing on the names
```

Output:

```
List slicing for [1:4]
['b', 'c', 'd']
List slicing for [3:]
['d', 'e', 'f', 'g', 'h']
List slicing for [:5]
['a', 'b', 'c', 'd', 'e']
List slicing for [:]
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
```

Appending to a list

An element can be added to a list using append operation. The append() method is used to add the element into the list.

Extending a list

An element can be added to a list using extend operation. The extend method is used to add the elements into the list.

The following example demonstrates the use of append() and extend() to the list.

```
#Initial list
nums=[1,2,3,4,5]

print('Initial list is')
print(nums)

#Append operation in a list
#Append [6,7,8] and print the length of the list
nums.append([6,7,8])
print('After appending')
print(nums)
print(len(nums))

#Extend operation in a list
nums2=[1,2,3,4,5]
print('Initial list is')
print(nums2)

#Extend the list and print the length of the list
nums2.extend([6,7,8])
print('After extending')
print(nums2)
print(len(nums2))

nums3=[10,20,30]
nums4=[40,50]
#Please fill the code here
new_nums=nums3[:]
print('After Appending')
nums3.append(nums4)
print(nums3)
print('After extending')
new_nums.extend(nums4)
print(new_nums)

[1,2,3,4,5]
After extending
[1,2,3,4,5,6,7,8]
8
After Appending
[10,20,30,[40,50]]
After extending
[10,20,30,40,50]
```

List Methods

The different methods of the list are listed as follows.

- `del()`: It is used to delete the element in the list.
- `append()`: It is used to add the element into the list.
- `sort()`: It is used to sort the elements in the list.
- `reverse()`: It is used to reverse the elements in the list.

```
#Create a list of fruits
fruit=['banana','apple','cherry']

print('Initial list is')
print(fruit)

#Changing the value of a list to demonstrate mutable list
fruit[2]='coconut'

print('After changing the value at 2nd position')
print(fruit)

#Delete an element into the list
del fruit[1]

print('After deleting the value at the position 1')
print(fruit)

#Insert an element into the list
fruit.insert(2, 'pear')
print('After inserting the value at the position 2')
print(fruit)

#Append an element to the list
fruit.append('apple')
print('After appending the value to the list')
print(fruit)

#Sort a list
fruit.sort()
print('Sorted fruit list is')
print(fruit)
```

```
#Reverse a list
fruit.reverse()
print('Fruit list in the reverse order is')
print(fruit)

matrix=[[1,0,0],[0,1,0],[0,0,1]]
print(matrix[0])
print(matrix[0][0])
print(matrix[1][1])

Initial list is
['banana', 'apple', 'cherry']
After changing the value at 2nd position
['banana', 'apple', 'coconut']
After deleting the value at the position 1
['banana', 'coconut']
After inserting the value at the position 2
['banana', 'coconut', 'pear']
After appending the value to the list
['banana', 'coconut', 'pear', 'apple']
Sorted fruit list is
['apple', 'banana', 'coconut', 'pear']
Fruit list in the reverse order is
['pear', 'coconut', 'banana', 'apple']
[1, 0, 0]
1
1
```

Dictionary

A dictionary is basic structure in Python that stores key value pairs, where the value associated with each key can be searched easily. The syntax for declaring a dictionary is enclosed within {} and key-value pairs are separated by : In this section, small examples on dictionary are discussed.

Example on Dictionary

The following code demonstrates a small example on dictionary and the iteration over the elements of the dictionary.

```
#Dictionary initialization
daily_temps={'sun':78.9,'mon':67.8,'tue':65.4,'wed':66.8,'thurs':76.89,
'fri':75.88}
```

```
#To access the values
print('Temperature on sunday')
print(daily_temps['sun'])

print('Temperature on monday')
print(daily_temps['mon'])

print(daily_temps['thurs'])
print(daily_temps['fri'])
```

Temperature on sunday

78.9

Temperature on monday

67.8

76.89

75.88

Dictionary methods

The following code demonstrates an example for a dictionary in Python.

```
scientist_to_birthdate = {'Newton' : 1642, 'Darwin' : 1809, 'Turing' : 1912}

print('All dictionary elements')
print(scientist_to_birthdate)

print('Items() method')
print(scientist_to_birthdate.items())

print('Keys() method')
print(scientist_to_birthdate.keys())

print('Get method')
print('The birthdate of Newton is' +str(scientist_to_birthdate.get
('Newton')))

#Get the birthdate for Darwin
print(scientist_to_birthdate['Darwin'])

#Add a new scientist Edison with birthdate as 1867
scientist_to_birthdate['Edison']=1867
```

```

#Print the updated dictionary
print(scientist_to_birthdate)

researcher_to_birthdate = {'Curie' : 1867, 'Hopper' : 1906,
                           'Franklin' : 1920}

print('Before update the dictionary elements are')
print(scientist_to_birthdate)

#Updating the dictionary
scientist_to_birthdate.update(researcher_to_birthdate)

print('After update the dictionary elements are')
print(scientist_to_birthdate)

All dictionary elements
{'Turing': 1912, 'Darwin': 1809, 'Newton': 1642}
Items() method
dict_items([('Turing', 1912), ('Darwin', 1809), ('Newton', 1642)])
Keys() method
dict_keys(['Turing', 'Darwin', 'Newton'])
Get method
The birthdate of Newton is 1642
1809
{'Turing': 1912, 'Darwin': 1809, 'Newton': 1642, 'Edison': 1867}
Before update the dictionary elements are
{'Turing': 1912, 'Darwin': 1809, 'Newton': 1642, 'Edison': 1867}
After update the dictionary elements are
{'Turing': 1912, 'Darwin': 1809, 'Franklin': 1920, 'Hopper': 1906,
 'Newton': 1642, 'Edison': 1867, 'Curie': 1867}

```

Phone book example

```

phone_book={}

def create_phone_book():
    #global word_dict
    #word_dict={}
    ch="y"
    while (ch=="y") or (ch=="Y"):
        print("\nEnter name:", end="")
        name=input()

```

```

print("\nEnter number:", end="")
number=input()
# Add a new entry to the phonebook dictionary
#----- Please fill the code-----
phone_book[name]=number
print("\nDo you want to continue adding (y or n):", end="")
ch=input()

def add_new_entry():
    #global word_dict
    print("\nEnter name:", end="")
    name=input()
    print("\nEnter number:", end="")
    number=input()
    # Add a new entry to the phonebook dictionary
    #----- Please fill the code-----
    phone_book[name]=number
def find_phone_number(name):
    print('Finding the phone number of '+name)
    # -- Please fill the code---

    print(phone_book.get(name))
def get_all_contacts():
    for name,no in phone_book.items():
        print("%s ==> %s" %(name,no))

```

Tuples

Tuples are similar to lists but they are enclosed within ‘()’. The following examples show the working of tuples in Python.

Example on Tuple

A string can be initialised as a tuple where the tuple is considered as a list of characters in the tuple.

```

>>> t = tuple('Python')
>>> print t
('P', 'y', 't', 'h', 'o', 'n')

```

The following example, shows how to sort a list of words from longest to shortest:

```
txt = 'but soft what light in yonder window breaks'
txt = 'Welcome to Python Programming'
words = txt.split()
t = list()
for word in words:
    t.append((len(word), word))

thone book example.sort(reverse=True)

res = list()
for length, word in t:
    res.append(word)

print res
```

[Programming, Welcome, Python, to]

Sets

A set represents an unordered collection of elements. The following code demonstrates a small example in Python for a set.

```
# initialize my_set
my_set = {1,3}
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing

#my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)
```

```
# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2,3,4])
print(my_set)

# addlist and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4,5], {1,6,8})
print(my_set)

{1, 3}
{1, 2, 3}
{1, 2, 3, 4}
{1, 2, 3, 4, 5, 6, 8}
```

Index

Numbers

3D bar plots, 361

A

Accuracy, 134

Activity, 277

Advanced visualizations, 361

Air pollution monitoring, 270

Analytical architecture, 8

Analytics, 3–5, 10, 11, 27, 219, 234, 265, 267, 268, 281, 314

Apriori algorithm, 191

Architecture of Hadoop, 30

Association rule mining, 191

Attributes, 133

B

Bag Of Words (BOW), 224

Bayes theorem, 155

Big data, 24, 25

Big data analytics, 269

Big data ecosystem, 25, 27

Box-whisker plot, 370

Bubble plot, 353

Business intelligence, 322

C

Call log analytics, 42

Categorical data, 6

Classification, 155

Cloud computing, 4

Clustering, 177

Comparison bar chart, 340

Comparison charts, 339

Composition charts, 339

Confusion matrix, 159

Contour, 361, 364

D

Data, 5

Data conditioning, 18

Data features, 14, 15

Data nodes, 30, 31

Data sources, 14

Data visualization, 321

Decision tree, 164

Dendrogram plots, 375

Descriptive analytics, 4, 5

DF, 245

Distribution charts, 339, 348

Domain, 12

E

Elbow curve, 178, 179

F

Fault-tolerant architecture, 31

Flume, 95

Frequency, 222

G

Gantt plot, 369

Google File System (GFS), 29

Graphviz, 333

H

Hadoop, 29

Heatmap, 377

Histogram, 363

Hive, 55

Hive DDL, 59

Hypothesis, 14

I

Internet of Things (IoT), 265

K

k-means, 177, 178

L

Lifecycle of data analytics, 11

Linear regression, 140

Logistic regression, 206

M

Machine learning, 127, 128

MapReduce, 33

Matplotlib, 333

Model, 131

Modified National Institute of Standards and Technology database (MNIST), 288

Mtcars, 151

N

Naïve Bayes, 155

NameNode, 30, 31

Natural language processing, 234

Network log analytics, 48, 49

Numerical data, 6, 7

O

Objective definition, 12

Ordinal data, 7

Overfitting, 146

P

Pig, 85, 86

Polar plot, 374

Polynomial regression, 142, 143

Principal Component Analysis (PCA), 195, 196

Principles of visualization, 324

Problem formulation, 13

Pyramid charts, 379

Python, 328

Q

Question classification, 249, 250

R

R, 327

Radar chart, 381

Random forest, 200, 201

Real-time analytics, 270

Regression, 139

Relationship charts, 339

Resilient Distributed Datasets (RDD), 77, 78

Retail analysis, 39

Ridge regression, 153

S

Scatter, 368

Scatter plot, 353

Scikit-learn, 135, 136

Sentence extraction, 236

Sentimental analysis, 254, 255

Spam classification, 240

Spam filter, 240, 244

Spark, 73

Stakeholders, 13

Stemmer, 247

Stopwords, 220

Storm, 109

Summarization, 234

Supervised learning, 132

Support Vector Machines (SVM), 169

T

Tableau, 330

TensorFlow, 283

Term Frequency (TF), 245

Text analytics, 219

Text collection, 220

Tokenizer, 247

U

Unsupervised learning, 132, 133

V

Visualization charts, 339

W

Word clouds, 231

Word count, 37

Word to vector, 227

Within Sum of Squares (WSS), 178